

**UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**MODELO DE APLICAÇÃO PARA RASTREADORES UTILIZANDO
PROTÓTIPO CRIADO COM ARDUINO DUE**

por

Lucas Alves Selliach

Itajaí (SC), novembro de 2013

**UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**MODELO DE APLICAÇÃO PARA RASTREADORES UTILIZANDO
PROTÓTIPO CRIADO COM ARDUINO DUE**

Área de Geoinformática

por

Lucas Alves Selliach

Relatório apresentado à Banca Examinadora do
Trabalho Técnico-científico de Conclusão do
Curso de Ciência da Computação para análise e
aprovação.

Orientador: Carlos Henrique Bughi, M.Sc.

Itajaí (SC), novembro de 2013

Dedico essa obra a meus pais, fonte da minha força e alegria.

AGRADECIMENTOS

Agradeço meus amigos e familiares que me deram força para continuar a lutar, aos meus pais Ivan Boardmann Selliach e Vanderci Alves Selliach que me deram mais do que apoio, me deram amor e carinho durante fases difíceis na conclusão desse projeto. Agradeço ao Joel Medeiros, Roberto Luis do Santos e Rafael de Santiago pelo incentivo e agradeço principalmente meu orientador M.sc Carlos Henrique Bughi, que me apoiou desde o início e me ajudou a concluir com sucesso o projeto.

RESUMO

SELLIACH, Lucas Alves. **Modelo de Aplicação para rastreadores utilizando protótipo criado com Arduino Due**. Itajaí, 2013. 87 folhas. Trabalho Técnico-científico de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2013.

O presente trabalho de conclusão de curso aborda um estudo sobre o desenvolvimento de um modelo de aplicação para rastreadores. Estes aparelhos servem para realizar o monitoramento de informações logísticas e geográficas que permite localizar pessoas, carros, barcos e até animais. Primeiramente foi realizada uma análise contextual através da leitura dos manuais de alguns modelos de rastreadores utilizados no mercado brasileiro, os aspectos históricos e conceituais, suas diversas aplicações práticas, normas e procedimentos. Em seguida foi elaborada uma pesquisa qualitativa de caráter exploratório cujo objetivo geral era reunir informações sobre o grau de satisfação e dificuldades dos desenvolvedores e técnicos na utilização e manuseio dos rastreadores. Com a interpretação e tabulação dos dados e a análise dos rastreadores utilizados elaborou-se um protótipo de rastreador e de um Sistema Operacional para o mesmo, realizando assim as alterações necessárias, podendo criar uma base para auxiliar nas soluções dos problemas levantados, principalmente os relativos à falta de padronização de SO para rastreadores. A partir da modelagem do protótipo e da modelagem do sistema operacional realizou-se o protótipo de um rastreador verificando a viabilidade da solução através de testes e resultados coletados.

Palavras-chave: Rastreadores. Arduino. Sistema Operacional.

ABSTRACT

This undergraduate project covers a study and development of an application model for trackers, being these devices used to monitoring geographical and logistical information, that can locate people, cars, boats and even animals. Firstly, we made a contextual aspects through readings manuals of some trackers used in the Brazilian Market, historical aspect, conceptual and its various practical applications, standards and procedures. Secondly, we drawn up a qualitative exploratory research whose overall objective is to gather information about the rating of satisfaction and difficulties of developers and technicians in the use and handling trackers. With interpretation and data synthesis and analysis of trackers used, we elaborated modeling of a prototype tracker and modeling of an operating system, using a real time Operating System, performing necessary changes, creating procedures to resolve problems founded, especially the related at lack of standardization of operating system for tracking. From the modeling of the prototype and the operating system modeling, we made a prototype of a tracker and an operating system for even allowing checking the feasibility of the proposed solution through testing and results collected.

Keywords: Trackers. Arduino. Operating System.

LISTA DE FIGURAS

Figura 1.	Rastreadores quanta.....	25
Figura 2.	Rastreadores maxtrack.....	26
Figura 3.	Rastreadores skypatrol.....	27
Figura 4.	Rastreadores quecklink.....	27
Figura 5.	Arduino due	29
Figura 6.	Pinos conectores usb	36
Figura 7.	Celular shield	39
Figura 8.	Gps shield	41
Figura 9.	Alterações no gsm shield	51
Figura 10.	Modelo de rastreador.	52
Figura 11.	Modelo de rastreador (visão interna).	53
Figura 12.	Arquitetura chibios.....	56
Figura 13.	Diagrama microkernel	59
Figura 14.	Diagrama sincronização	60
Figura 15.	Uso da memória	61
Figura 16.	Diagrama de gerenciamento de memória	62
Figura 17.	Fluxograma de aplicação.....	63
Figura 18.	Diagrama de sequência do sistema.....	64
Figura 19.	Padrão nmea	65

LISTA DE QUADROS

<i>Quadro 1. Hardware Arduino Due.....</i>	<i>30</i>
<i>Quadro 2. Pinos microcontrolador Cortex-M3.....</i>	<i>31</i>
<i>Quadro 3. GSM/GPRS shield.....</i>	<i>40</i>
<i>Quadro 4. Esquema Eletrônico.</i>	<i>50</i>
<i>Quadro 5. Código fonte inicial do modelo.....</i>	<i>57</i>
<i>Quadro 6. Criação das threads com prioridades diferenciadas.</i>	<i>58</i>
<i>Quadro 7. Utilização de tabela com sincronismo entre threads.....</i>	<i>59</i>
<i>Quadro 8. Criação de thread.....</i>	<i>60</i>
<i>Quadro 9. Serial com GPS.....</i>	<i>62</i>
<i>Quadro 10. Testes de hardware.</i>	<i>66</i>
<i>Quadro 11. Testes de Sistema Operacional.</i>	<i>66</i>
<i>Quadro 12. LOG rastreador.....</i>	<i>67</i>
<i>Quadro 13. LOG servidor.....</i>	<i>68</i>

LISTA DE ABREVIATURAS E SIGLAS

AC/DC – *Alternating Current / Direct Current*
API – *Application Programming Interface*
ARM – *Advanced RISC Machine*
CAN – *Controller Area Network*
E/S – *Entrada / Saída*
FCC – *Federal Communications Commission*
FDMA – *Frequency Division Multiple Access*
GND – *Graduated Neutral Density Filter*
GPRS – *General Packet Radio Service*
GPS – *Global Positioning System*
IDE – *Integrated Development Environment*
NMEA – *Nation Marine Electronics Association*
RPM – *Rotação Por Minuto*
SIM – *Subscriber Identity Module*
SMS – *Safety Management System*
SPI – *Serial Peripheral Interface Bus*
SO – *Sistema Operacional*
UART – *Universal Asynchronous Receiver/Transmitter*
USB – *Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO.....	15
1.1 PROBLEMATIZAÇÃO.....	17
1.2 FORMULAÇÃO DO PROBLEMA	17
1.2.1 Solução Proposta.....	19
1.3 OBJETIVOS.....	19
1.3.1 Objetivo Geral	19
1.3.2 Objetivos Específicos.....	19
1.4 METODOLOGIA.....	20
1.5 ESTRUTURA DO TRABALHO.....	21
2 FUNDAMENTAÇÃO TEÓRICA	22
2.1 INTRODUÇÃO AOS RASTREADORES.....	22
2.1.1 Rastreadores no mercado brasileiro	24
2.1.2 Sistemas Embarcados	28
2.2 ARDUINO DUE E SEUS SHIELDS	28
2.2.1 Hardware do Arduíno Due	29
2.2.2 Software do Arduíno Due	36
2.2.3 Shields para Arduino Due.....	39
2.3 SISTEMAS OPERACIONAIS DE TEMPO REAIS	41
2.3.1 Sistemas Operacionais comerciais	42
2.3.2 Comparativo entre os Sistemas Operacionais.....	44
2.4 PROJETOS SIMILARES.....	45
2.4.1 Sistemas de rastreamento embarcados.....	45
2.4.2 Sistemas de monitoração veicular via GPRS	46
3 DESENVOLVIMENTO	49
3.1 DESENVOLVIMENTO DO HARDWARE.....	49
3.1.1 Esquema eletrônico.....	49
3.1.2 Realização e montagem do protótipo	51
3.2 DESENVOLVIMENTO DO SOFTWARE	53
3.2.1 Análise Requisitos.....	54
3.2.2 Modelagem e Desenvolvimento	55
3.2.3 Plano de Testes	65
4 CONCLUSÃO.....	69
4.1 TRABALHOS FUTUROS.....	70
APÊNDICE A. QUESTIONÁRIO SOBRE RASTREADORES.....	73
A.1 ENTREVISTA 1	73
A.2 ENTREVISTA 2	74
A.3 ENTREVISTA 3	75
APÊNDICE B. CÓDIGO FONTE DO RASTREADOR MODELO	77

<i>ANEXO A.</i>	<i>CÓDIGO FONTE DO CHIBIOS.C E CHIBIOS.H</i>	<i>84</i>
------------------------	---	------------------

1 INTRODUÇÃO

O progresso científico tecnológico vem transformando incessantemente o modo como se percebe o planeta. A busca da eficiência, da eficácia, da produtividade, da qualidade ocasionou um aprofundamento com ênfase em novas tecnologias, como a que ocorreu no dia quatro de outubro de 1957, um marco para a história mundial. Nesta data, a União Soviética colocou o satélite Sputnik em órbita. “Este foi o primeiro satélite artificial lançado ao espaço e marcou a corrida espacial que era travada entre os soviéticos e os Estados Unidos” (ARAÚJO, 2012). Assim deu-se início a novas técnicas de localizações utilizando satélites como referências.

No ano de 1973, o “Departamento de Defesa dos Estados Unidos desenvolveu um sistema de posicionamento de alvos para fins militares” (ALBUQUERQUE; SANTOS, 2003). O projeto NAVSTAR GPS “é um sistema rádio com um sentido, emitindo duas frequências a partir de uma constelação de 24 satélites, com as quais é mensurável a distância entre as posições conhecidas dos satélites e um receptor” (PRATES, 2004).

“O sistema atingiu sua configuração final somente a partir de 1994, [...]. Alavancado pelas necessidades apresentadas pela sociedade, o sistema *Global Positioning System* GPS (Sistema de Posicionamento Global) tornar-se-ia um forte concorrente dos meios tradicionais de levantamentos e ferramentas eficaz de apoio à navegação marítima e aérea. Com a chegada do GPS ao País, criou-se novas frentes de trabalho, assinalando-se a abertura e operação de empresas especializadas no uso e aplicação desse sistema e representações técnicas e comerciais, voltadas a venda e manutenção dos receptores GPS” (ALBUQUERQUE; SANTOS, 2003).

Através da citação de Albuquerque e Santos (2003), percebeu-se que com os receptores GPS (*Global Positioning System*) modernos e precisos ficou mais exato a localização no meio geográfico, sendo assim, o homem desenvolveu novas técnicas realizando a localização de veículos, embarcações, aviões e qualquer objeto ou ser vivo que precise ser localizado e rastreado, agora, não só pelas forças armadas, mas também pelo uso civil. Junto com o avanço dos GPS e com a corrida espacial travada, evoluiu a tecnologia chamada telemetria.

“Telemetria, que mais evidentemente significa uma medição realizada a distância e que tem como característica ser o mais exigente no que diz respeito às especificações de construção e desempenho dos equipamentos. [...] A telemetria surgiu devido à necessidade de realizar medições em locais inacessíveis, e evoluiu em uma ciência complexa capaz de realizar medições em qualquer local remoto” (MATTOS, 2004).

Segundo Mattos (2004), “testes de veículos foi uma das primeiras aplicações da telemetria”. Aplicada em veículos tripulados e não tripulados a telemetria veio a contribuir para o desenvolvimento dos rastreadores modernos. Portanto, a partir dessa contribuição

desenvolveram-se rastreadores que além de obter informações de localizações podem também obter informações e interagir com o veículo através de sensores (onde o objetivo é “medir alguma grandeza física, como temperatura” (MATTOS, 2004) e atuadores, quando é enviado “sinais aos carros para fazer algumas mudanças pequenas em algumas partes, por exemplo, no motor.” (MATTOS, 2004). “O dispositivo depende de um chip GPS conectado a um transmissor para enviar dados de localização (muitas vezes por rede de celulares GSM/GPRS) para servidores das companhias de rastreamento” (MCNAMARA, 2008).

Exemplificando, os rastreadores são aparelhos eletrônicos que surgiram com o avanço e a popularização da telemetria a fim de realizar o monitoramento de informações logísticas e geográficas, ele utiliza o sistema GPS para realizar a localização e as tecnologias como GSM/GPRS, SMS (*Safety Management System*) ou satelital para realizar o tráfego de dados e informações por meio da Internet. Entre as adaptações dos rastreadores se destaca a capacidade de E/S controlada para realizar interações com diversos componentes como bloqueio de veículo, alerta de sirene, controle de portas, leitura de RPM (Rotação Por Minuto), leitura de velocidade e outros.

Portanto, percebe-se que existem equipamentos no mercado denominados rastreadores e eles possuem tecnologias como telemetria, GPS e comunicação por redes telefônicas. Essas tecnologias estão dispostas em periféricos e estes estão conectados a um microcomputador denominado Microcontrolador.

Assim, segundo Oliveira e Andrade (2006):

“Os microcontroladores, em geral, possuem todos os periféricos necessário em um único chip. Seu tamanho também é muito pequeno, mesmo contendo vários periféricos como: memórias, barramentos, *Timer's*, porta de comunicação, conversores de sinais analógicos para digitais e etc”.

Com a quantidade de recursos a serem gerenciados e a conexão de diversos periféricos, os rastreadores necessitam de um Sistema Operacional que possa realizar tal tarefa. Para Oliveira, Carissimi e Toscani, (2008) “o Sistema Operacional é uma camada de software colocada entre o hardware e os programas que executam tarefas para o usuário”, ou seja, no contexto de rastreadores “o Sistema Operacional é responsável pelo acesso aos periféricos” (OLIVEIRA; CARISSIMI; TOSCANI, 2008) de GPS, GSM/GPRS, E/S, acelerômetro e outros.

Atualmente a maioria dos rastreadores possui um Sistema Operacional, mas o que se pode observar mediante pesquisa, é que não existe uma padronização entre eles para o seu

manuseio. Portanto, percebe-se que cada um possui um Sistema Operacional próprio criado pela fabricante do rastreador.

A partir desse fato observado, no qual fundamenta-se esta pesquisa de conclusão de curso em Ciência da Computação, foi realizado um estudo observando conceitos nos manuais, lançando o olhar para alguns modelos existentes de rastreadores utilizados em rastreamento de veículos, seus aspectos históricos e conceituais, aplicações práticas, normas e procedimentos.

Depois, foi realizada pesquisa qualitativa exploratória através de aplicação de questionários para observar-se a satisfação e ou descontentamento dos desenvolvedores e técnicos na utilização e manuseio dos rastreadores. Com o fim desta etapa, realizou-se uma pesquisa sobre o funcionamento do Sistema Operacional para rastreadores, destacando-se as adaptações realizada no SO (Sistema Operacional) para o funcionamento nos rastreadores.

Em quarto, com a interpretação e tabulação dos dados, será elaborado um protótipo de rastreador baseado em micro controladores ARM (*Advanced RISC Machine*), propostas e planos de ação, sugerindo melhorias nos SO para rastreadores, modelando o Sistema Operacional com intuito de definir seu funcionamento e ou abstraindo o que não for necessário, adaptando o Sistema Operacional de acordo com a modelagem desenvolvida.

1.1 PROBLEMATIZAÇÃO

1.2 FORMULAÇÃO DO PROBLEMA

Os rastreadores são dispositivos eletrônicos que foram adaptados com o surgimento de tecnologias como o GPS e a Telemetria, a quantidade de periféricos e a variedade de funções que ele possui e pode desempenhar.

“O Sistema Operacional é uma camada de software colocada entre o hardware e os programas que executam tarefas para o usuário. [...] O Sistema Operacional é responsável pelo acesso aos periféricos. Sempre que um programa necessita de algum tipo de entrada e saída, ele a solicita ao Sistema Operacional. Dessa forma o programador não precisa conhecer os detalhes do hardware. [...] Ao mesmo tempo, como todos os acessos aos periféricos são feitos através do Sistema Operacional, ele pode controlar qual programa pode acessar qual recurso. É possível, então, obter uma distribuição justa e eficiente dos recursos” (OLIVEIRA; CARISSIMI; TOSCANI, 2008).

Observando a citação de Oliveira, Carissimi e Toscani (2008) verifica-se que um rastreador para ter seu funcionamento completo precisa de um Sistema Operacional que possa controlar e realizar a distribuição de acesso aos periféricos de forma eficiente.

Conforme Tanenbaum (2010), o Sistema Operacional realiza duas funções básicas não relacionadas que são: estender a máquina e gerenciar os recursos. Como máquina estendida o Sistema Operacional deve apresentar uma forma de programar mais fácil do que hardware, ou seja, ele ficará responsável por realizar a comunicação direta com os periféricos. Como gerenciador de recursos o SO deverá manter o controle sobre quem está usando qual recurso, garantido suas requisições de recursos, controlando as contas e mediando conflitos de requisições entre diferentes programas e usuários. Portanto, pode se observar mediante pesquisa, que o sistemas operacionais disponíveis nos rastreadores possuem essas características mas cada fabricante possui o seu SO. Isso ocorre pelo fato de não existir um SO padrão para uso dos fabricantes. Sendo assim é visível que esse fato agregaria problemas de utilizações dos rastreadores, sendo que cada Sistema Operacional possui características únicas variando de rastreador para rastreador.

Para analisar melhor os problemas dos rastreadores, foi proposto um questionário (Apêndice A: Questionário sobre rastreadores) aplicado como entrevista a três entrevistando, desenvolvedores na área de rastreamento e telemetria. As entrevistas foram realizadas individualmente, mas as questões propostas aos entrevistados foram as mesmas, em número de sete, elaboradas a partir da pesquisa bibliográfica e a leitura realizada nos manuais. As perguntas estavam direcionadas a profissão, área de atuação, tempo de serviço e ou experiência, tipo de rastreadores utilizados na área de atuação, benefícios encontrado nos rastreadores utilizados na área de atuação, dificuldades encontradas no manuseio dos mesmos e o que poderia em sua visão ser melhorado para uma maior eficiência na utilização desses rastreadores, por eles manuseados.

Ao verificarem-se as respostas, constatou-se que os entrevistados estavam envolvidos na área da pesquisa proposta, ocorrendo certa unanimidade quanto à utilização dos rastreadores, pois de acordo com os entrevistados os diferentes equipamentos (modo de comunicação *Over The Air*) dificulta bastante a integração destes ao sistema e ainda a falta de uma rede de comunicação inteligente (TIA/EIA-485) ou um CAN-BUS¹ foram fatores apontados durante a entrevista. Durante a análise técnica dos rastreadores utilizados pelas empresas pesquisadas, foi detectado que, fora a família Tetros todos eles dispõem somente de interfaces seriais TIA, EIA-232, o que limita muito o desenvolvimento. Mesmo rastreadores Quanta Tetros, com portas

¹ Protocolo de comunicação serial síncrono.

485, possuem protocolos limitados, principalmente no controle de tráfego de dados conforme afirma um dos pesquisados.

Dentre as dificuldades encontradas nos rastreadores mais utilizados na área de atuação dos pesquisados, apontam como melhoria a criação de um *Firmware* que padronize a comunicação dos equipamentos, o recebimento dos comandos *Over The Air*, de modo que facilite a integração destes equipamentos pelos diversos sistemas de telemetria existentes. A adição de uma comunicação decente com periféricos, uma configuração melhor interna, porém simplificada, mirando o que o Skypatrol² realiza, mas com algumas interfaces de programação nos rastreadores que facilita-se na tarefa de homologar e configurar esses rastreadores.

1.2.1 Solução Proposta

Mediante a pesquisa bibliográfica, análise dos manuais dos rastreadores citados pelos entrevistados, as dificuldades por eles citadas e as melhorias por eles propostas, observou-se a necessidade de buscar uma solução desenvolvendo um protótipo de rastreador com micro controlador e adicionando um Sistema Operacional, modelando-o com intuito de definir seu funcionamento e ou abstraindo o que não for necessário, buscando assim fornece um modelo de aplicação padrão que poderá ser utilizado por diferentes empresas de rastreamento.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Desenvolver um protótipo de rastreador e adaptar um Sistema Operacional para o mesmo.

1.3.2 Objetivos Específicos

- Levantamento de modelos de rastreadores mais comum no mercado;
- Análise de configuração e funcionamento dos rastreadores para melhor entendimento dos problemas neles encontrados;
- Realizar estudo de caso, *in loco* demonstrando os problemas encontrados.

² Rastreador fornecido pela empresa americana de mesmo nome (SKYPATROL, 2013).

- Realizar um estudo de sistemas operacionais que já estão disponíveis em rastreadores;
- Realizar estudo sobre os protocolos de comunicações envolvidos no processo de rastreamento (GPS, GSM);
- Modelar um rastreador analisando o hardware a ser utilizado;
- Modelar o Sistema Operacional com intuito de definir os protocolos de comunicações, interfaces e seu funcionamento;
- Realizar a adaptação/implementação de um Sistema Operacional de acordo com a modelagem desenvolvida;
- Analisar os resultados obtidos com a aplicação do sistema nos rastreadores, através de implantação e uso; e
- Documentar o estudo desenvolvendo um artigo técnico-científico através de resultados.

1.4 METODOLOGIA

A metodologia deste projeto é dividida em cinco partes:

- Conceituação: Nesta etapa foi realizado um estudo sobre os principais conceitos de eletrônica embarcada e Sistemas Operacionais, focando principalmente em sistemas embarcados com base em ARM e Sistemas Operacionais embarcados, utilizando-se livros e manuais para aquisição de conhecimento;
- Estudo e análise de SO embarcados: Nesta etapa foi realizado o estudo e análise dos SO disponíveis para a arquitetura ARM, verificando quais se aproximam de uma melhor adaptação para uso em rastreadores;
- Montagem do protótipo de rastreador: Será realizado a montagem de um protótipo de rastreador utilizando periféricos necessário para seu funcionamento.
- Especificação do SO embarcado: Nesta etapa foi analisado e apresentado a adaptação de um Sistema Operacional embarcado para uso no rastreador, alterando e modificando conforme a necessidade;

- Implementação: Nesta etapa foi implementado o sistema operacional mencionado na etapa anterior; e
- Documentação: Onde foi realizada a documentação do processo existente no trabalho, deste a conceituação até a validação, incluindo também a criação de um artigo científico.

1.5 ESTRUTURA DO TRABALHO

Este projeto está dividido em quatro partes. O primeiro capítulo é a Introdução, o qual foi apresentado uma visão geral sobre o assunto do trabalho. O segundo capítulo contempla a Fundamentação Teórica, que apresenta os conceitos necessários englobando o projeto. O terceiro capítulo é o projeto, onde descreve o processo de adaptação do sistema operacional no protótipo gerado. O quarto capítulo são as considerações finais em que será realizada uma visão geral sobre o assunto abordado, e as consequências geradas através do estudo e realização do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTRODUÇÃO AOS RASTREADORES

Com o desenvolvimento do uso da telemetria surgiram novas tecnologias, como os GPS. “Os GPS oportunizaram serviços de rastreamento e monitoramento remotos, sendo estes alvos muito explorados no momento do surgimento da rede GSM [...]” (ALEGRE; OLIVEIRA, 2008). Esses foram primordiais para o desenvolvimento dos rastreadores que vieram facilitar a vida da sociedade em toda a área de conhecimento.

Os rastreadores são equipamentos que servem para monitorar informações e desempenhando um papel importante.

Os rastreadores veiculares surgiram da integração de três tecnologias pré-existent: a do GPS, a cartografia digital e as comunicações sem fio, tecnologias que contavam com vários anos de existência ao momento de sua integração. A rápida evolução do setor de comunicações ajudou na proliferação dos rastreadores, os quais conseguiram um grande desenvolvimento através principalmente nas inovações de componentes de hardwares. (FULMAR, 2013)

Portanto, percebe-se que a disponibilidade dos GPS e a evolução das redes telefônicas sem fio, permitiu o nascimento dos rastreadores e com a evolução do mesmo aumentou o uso da telemetria.

O GPS, ou Navstar (*Na Viglion Satellite With Time And Ramging*), “é um sistema de rádio navegação desenvolvido pelo departamento dos Estados Unidos – DOD (*Department Of Defense*), com o intuito de ser o principal sistema de navegação das forças armadas americanas” (MONICO, 2000).

O governo americano financiou dois programas para que fosse desenvolvido um sistema de navegação que tivesse uma abrangência global, esta ficou sobre a responsabilidade da marinha e das forças armadas americanas. O sistema foi declarado totalmente operacional apenas em 1995. Seu desenvolvimento custou 10 bilhões de dólares. Consiste numa “constelação” de 24 satélites. (LEÃO, 2004)

O GPS passou a ser liberado para a comunidade civil e assim tornou-se um “Sistema de abrangência global”, uma vez que o mesmo proporciona facilidades para todo o ramo de atividades que forem necessários acompanharem posicionamento. “A concepção do sistema GPS permite que o usuário, em qualquer lugar da superfície terrestre, ou próximo a ela, tenha a sua disposição no mínimo, quatro satélites para serem rastreados”. Assim, com a tecnologia

desenvolvida nos satélites monitorar ficou muito mais fácil e preciso, sendo que o GPS poder ser utilizado sobre quaisquer condições climáticas.

Há dois tipos de serviços os quais são conhecidos como: SPS, serviço de posicionamento padrão e PPS, serviço de posicionamento preciso. O SPS é um serviço de posicionamento padrão que está disponível a todos os usuários. O PPS proporciona maiores resultados, mas é restrito ao uso militar e a usuários autorizados (MONICO, 2000).

Analisando as colocações de Monico, pode-se perceber que a utilização do PPS seria o ideal pois este iria facilitar mais ao usuário, mas este ainda é restrito a uma classe mais elitizada. O GPS consiste de três seguimentos principais: espacial, controle e de usuário

O seguimento espacial consiste de 24 satélites distribuídos em seis orbitais igualmente espaçados [...]. O seguimento de controle, com sua principal tarefa de: monitorar e controlar o sistema de satélites, determinar o sistema no campo do GPS, prever as efemérides dos satélites, calcular as correções dos relógios dos satélites e atualizar periodicamente as mensagens de navegação de cada satélite[...]. O seguimento de usuários é constituído pelos receptores GPS, que devem ser apropriados para os propósitos a que se destinam. (MONICO, 2000)

Em razão da alta tecnologia proporcionada pelo GPS e do desenvolvimento da tecnologia nos receptores GPS, após a liberação para o meio civil, surgiu uma gama de usuários dos mais variados seguimentos da comunidade civil, quer seja nas navegações, posicionamento geográfico, agricultura e no controle de frotas.

A primeira Geração de telefonia foi desenvolvida no início dos anos 80, nela estabeleceu-se a estrutura e as funções básicas da telefonia celular como, por exemplo, *roaming* e *handover* entre células. “Com a primeira geração, que possuía sistemas analógicos, era possível apenas transmitir voz no mesmo país. Para isso era usado acesso múltiplo por divisão de frequência (FDMA – *Frequency Division Multiple Access*)” (OLIVEIRA; PITOMBO; ALEGRE, 2008). Nessa geração foram utilizados dois sistemas:

- NMT – “Esse sistema foi a primeira rede da telefonia móvel do mundo, utilizado por países escandinavos, alguns países europeus, partes da Rússia, Oriente Médio e Ásia” (OLIVEIRA; CARISSIMI; TOSCANI, 2008)
- AMPS – Tecnologia que utilizar radiofrequência para comunicação. Este “Aloca um canal de voz que permanece dedicado durante toda a duração de uma chamada” (OLIVEIRA; PITOMBO; ALEGRE, 2008)

Já a segunda geração foi substituído os sistemas analógicos por digitais. Com o surgimento dos sistemas digitais oportunizou-se a melhoria no aproveitamento da utilização da

frequência, a parceria entre operadoras diferentes (*roaming*), a melhora do “serviço e a adição de serviços de dados, como mensagens de texto e multimídia” (OLIVEIRA; PITOMBO; ALEGRE, 2008). Na segunda geração foram adicionadas as tecnologias como:

- GSM – “Esse sistema digitaliza e comprime os dados e, logo após, envia e estabelece um canal com dois outros fluxos de dados de usuários, cada um no seu próprio *slot* de tempo”; (OLIVEIRA; PITOMBO; ALEGRE, 2008)
- CDMA – Sistema em que “todas as máquinas transmitem e recebem na mesma frequência”; e (OLIVEIRA; PITOMBO; ALEGRE, 2008)
- TDMA – “Divide a rádio frequência em *slots* de tempo e aloca os *slots* para múltiplas chamadas. [...] A tecnologia compartilha um canal de comunicação entre vários usuários”.

A Segunda Geração e Meia, a 2,5G renova na tecnologia dos pacotes, sendo que neste a cobrança é feita por dados transmitidos. Com essa geração surgiu as tecnologias:

- GPRS – O GPRS facilita com conexões instantâneas, as informações são divididas em pacotes denominado dados e estes são transmitidos através da operadora entre o destino e origem, assim tornando possível a cobrança por dados e não por tempo de conexão. Assim tornar viável também a utilização de um canal de frequência para vários clientes; e (OLIVEIRA; PITOMBO; ALEGRE, 2008)
- O EDGE – Com essa tecnologia “é possível transmitir dados em alta-velocidade e serviços como *streaming* de vídeos, rádios ao vivo e transferência de arquivos”.

Na terceira geração, a 3G, surgiu o melhoramento das tecnologias que possibilitou o aumento de velocidades no transporte dos pacotes de dados. O UMTS e o CDMA2000 vieram com esta geração.

2.1.1 Rastreadores no mercado brasileiro

Com a evolução do GPS e da Telemetria os rastreadores foram sendo criados e adaptados e ao decorrer do processo e evoluíram para o que se tem atualmente disponível no mercado brasileiro. Entre centenas e milhares de rastreadores já disponíveis destacam-se marcas

como Quanta, Maxtrack, Skypatrol e Queclink. O motivo pelo qual tais produtos possui destaque é que cada um possuem características superior ou equivalente a outros modelos de outras empresas e por esse fator mais o fato de elas serem mencionadas nos questionários torna essas marcas foco da análise a ser realizada neste projeto.

Segundo o entrevistado 1 (Apêndice A: Questionário sobre rastreadores) cada fabricante possui um modelo e este uma característica que o torna único, sendo que essas particularidades variam entre preços e funcionalidades, logo, todas as fabricantes possui modelos diferenciados que buscam suprir necessidades dos usuários. A empresa Quanta possui como produtos de venda um total de seis rastreadores, variando entre um produto de preço barato com funções reduzidas até um produto com maior preço e mais conjunto de funções. Na Figura 1 é possível analisar tais produtos e suas características.







	Nome	Possui Telemetria Interna?	Possui Comunicação GSM/GPRS?	Possui GPS?	Possui Rede TIA/EIA 485	Características
	Radar Duo	Sim (acelerômetro)	Sim	Sim	Não	Projetado para atender veículos de passeio, motocicletas, frota de ônibus e caminhões.
	Tetros Auto	Sim (acelerômetro)	Sim	Sim	Não	O Tetros Auto foi projetado para atender integralmente a resolução 245/253 do Denatran em veículos de carga e de passeio.
	Tetros Maxi	Não	Sim	Sim	Sim	O Tetros Maxi é um equipamento utilizado em veículos de carga disponibilizando funções para soluções de logística interestadual simplificada.
	Tetros Midi	Não	Sim	Sim	Sim	O Tetros Midi é um equipamento utilizado em veículos de carga leve, disponibilizando funções para aplicações de segurança e logística simplificada.
	Tetros Moto	Sim (acelerômetro)	Sim	Sim	Não	Produto desenvolvido para veículos de 2 rodas, projetado para atender integralmente a resolução 245/253 do Denatran.
	Tetros Plus Ibutton	Sim (varios)	Sim	Sim	Sim	Produto completo para soluções de logística interestadual, com telemetria embarcada, funções das gerenciadoras de risco (GR) e entrada para identificação de motorista.

Figura 1. Rastreadores Quanta

Através das análises realizada nos equipamentos da Quanta, foi possível verificar que os mesmos possuem como padrão a comunicação por GSM e a detecção de posição por GPS, assim validando o produto como rastreador. Esses modelos analisados possuem telemetria devido ao fato de realizarem leituras de velocidade e RPM e opções de atuação no veículo como bloqueio do mesmo. Em alguns modelos apresenta-se características diferenciadas de telemetria como acelerômetro embarcado o que possibilita a análise de dados como aceleração, freada brusca e curva brusca. Um modelo revelou possuir mais condições de verificação de telemetria,

segundo a (QUANTA, 2013), o Tetros Plus *Ibutton* tem a capacidade de identificar banguela, velocidade mínima e máxima, RPM mínimo e máximo entre outros. Os modelos que possuem rede de comunicação TIA/EIA 485 apresentam maior capacidade de integração com periféricos que possuam a mesma interface.

A empresa Maxtrack possui um total de oito rastreadores como produtos, seis são rastreadores veiculares e dois são rastreadores pessoais. Como o foco neste projeto é somente rastreadores veiculares, realiza-se uma análise de seus modelos automotivos na Figura 2.







	Nome	Possui Telemetria Interna?	Possui Comunicação GSM/GPRS?	Possui GPS?	Possui Rede TIA/EIA 485	Características
	IDP-780	Sim (acelerômetro)	Sim	Sim	Sim	Equipamento de rastreamento com dual sim, autonomia de programação (lua), integração com periféricos através de rede sem fio e diversas funcionalidades extendidas.
	MTC-550 FULL	Sim (Varios)	Sim	Sim	Sim	Equipamento com módulo de ações embarcadas com análise de giro alto, banguela, valor do hodômetro, tempo de rotação por faixa, tempo motor ligado, velocidade máxima e outros,
	MTC-780	Sim (acelerômetro)	Sim	Sim	Sim	Equipamento de rastreamento com dual sim, autonomia de programação (lua), integração com periféricos através de rede sem fio, rede de comunicação RS-232/485/CAN e diversas funcionalidades extendidas.
	MXT-140A	Sim (acelerômetro)	Sim	Sim	Não	Rastreador projetado para veículos menores, possui o necessários para um rastreador.
	MXT-140B	Sim (acelerômetro)	Sim	Sim	Não	Rastreador projetado para veículos menores, possui o necessários para um rastreador com algumas características extras como controle de ignição, 03 entradas e 02 saídas.
	MXT-151	Sim (acelerômetro)	Sim	Sim	Não	Equipamento com alto foco em integração de periféricos através de redes wireless,

Figura 2. Rastreadores Maxtrack

Todos os produtos da Maxtrack possuem telemetria interna com a utilização do acelerômetro. Um modelo possui alta customização através de todas suas possibilidades de ações embarcadas e mais a utilização da linguagem interpretada LUA que é um grande diferencial na customização/programação destes equipamentos.

As empresas Skypatrol e Queclink são estrangeiras com produtos frequentemente utilizados no Brasil, a Skypatrol possui um total de seis rastreadores e a Queclink um total de cinco rastreadores conforme se pode observar nas Figura 3 e Figura 4.







	Nome	Possui Telemetria Interna?	Possui Comunicação GSM/GPRS?	Possui GPS?	Possui Rede TIA/EIA 485	Características
	TT 1150	Não	Sim	Sim	Não	Equipamento utilizado em motocicletas, possui pouca customização porém um bom custo benefício.
	TT 8750	Não	Sim	Sim	Não	Rastreador que possui inteligência embarcada, alta customização capaz de gerar alertas.
	TT 8750+	Não	Sim	Sim	Não	Possui as mesmas características do TT8750 mais um incremento da Segunda geração do EDDIE (protocolo de customização).
	TT 8850	Não	Sim	Sim	Não	Equipamento utilizado em monitoramento de cargas, possui um tamanho reduzido e foco na duração da bateria.
	TT 8950	Sim (acelerômetro)	Sim	Sim	Não	Equipamento com alto controle de telemetria, sendo possível realizar uma alta customização com diversos controles de E/S.
	TT 9150	Não	Sim	Sim	Não	Possui as mesmas características do TT 8750, porém com um designer ultra fino.

Figura 3. Rastreadores Skypatrol

Com base na tabelas apresentadas nota-se que os produtos da Skypatrol e os produtos da Quecklink possuem menos funcionalidades e integrações com periféricos, mas ainda mantem as características fundamentais para serem considerados rastreadores. O diferencial dos produtos da Skypatrol é a capacidade das grandes customizações possíveis através da sintaxe AT&T.






	Nome	Possui Telemetria Interna?	Possui Comunicação GSM/GPRS?	Possui GPS?	Possui Rede TIA/EIA 485	Características
	GMT100	Sim	Sim	Sim	Não	Equipamento utilizado em motocicletas.
	GV 200 GV 200G	Sim	Sim	Sim	Não	Equipamento que realiza integração com periféricos através da rede RS232.
	GV 300	Sim	Sim	Sim	Não	Equipamento que realiza integração com periféricos através da rede Rs232 e possui maior capacidade de customização.
	GV 55	Sim	Sim	Sim	Não	Rastreador que garante a posição através de GPS e CELL ID além de ter um detector de colisão.
	GV 55 LITE	Sim	Sim	Sim	Não	Possui mesmas características que o GV 55, ajudando assim no custo benefício.

Figura 4. Rastreadores QueckLink

2.1.2 Sistemas Embarcados

Com a análise da história dos rastreadores e dos modelos percebe-se que estes são constituídos de vários periféricos como o periférico de comunicação, localização e a telemetria através de entradas e saídas. Além dessas funcionalidades, o rastreador se comporta como um sistema embarcado instalado em veículos ou simplesmente colocado junto ao objeto a ser localizado.

Segundo Oliveira e Andrade, define-se sistemas embarcados como sistemas que possuem uma capacidade de processamento de informações vinda de um software que está sendo processado internamente nesta unidade e estes são compostos por uma unidade de processamento, que é um circuito integrado, fixado a um circuito impresso.

Na arquitetura geral de um sistema embarcado é notável a utilização de vários componentes como memórias, processadores, periféricos entre outros. Para um rastreador é normalmente utilizado os microcontroladores. Os microcontroladores, em geral, possuem todos os periféricos necessário em um único chip, esse periféricos são memórias, barramentos, *timer's*, portas de comunicação, conversores de analógico para digital e etc. “Eles possuem desempenho menor que os microprocessadores, mas são ideais em aplicações que necessitam de menores dimensões, preços e custos” (OLIVEIRA; ANDRADE, 2006).

2.2 ARDUINO DUE E SEUS SHIELDS

Arduino é uma plataforma de prototipagem eletrônica *open-source* que possui características flexíveis em seu *Hardware* e *Software*. Utilizado para a construção de objetos interativos (como sistemas embarcados), o Arduino pode perceber o ambiente por uma variedade de sensores e atuar nele com *leds* (*Light Emitting Diode*), controladores, motores e outros dispositivos físicos conhecido como atuadores (ARDUINO, 2013). A Figura 5 apresenta um dos modelos de Arduino com um bom custo/benefício.

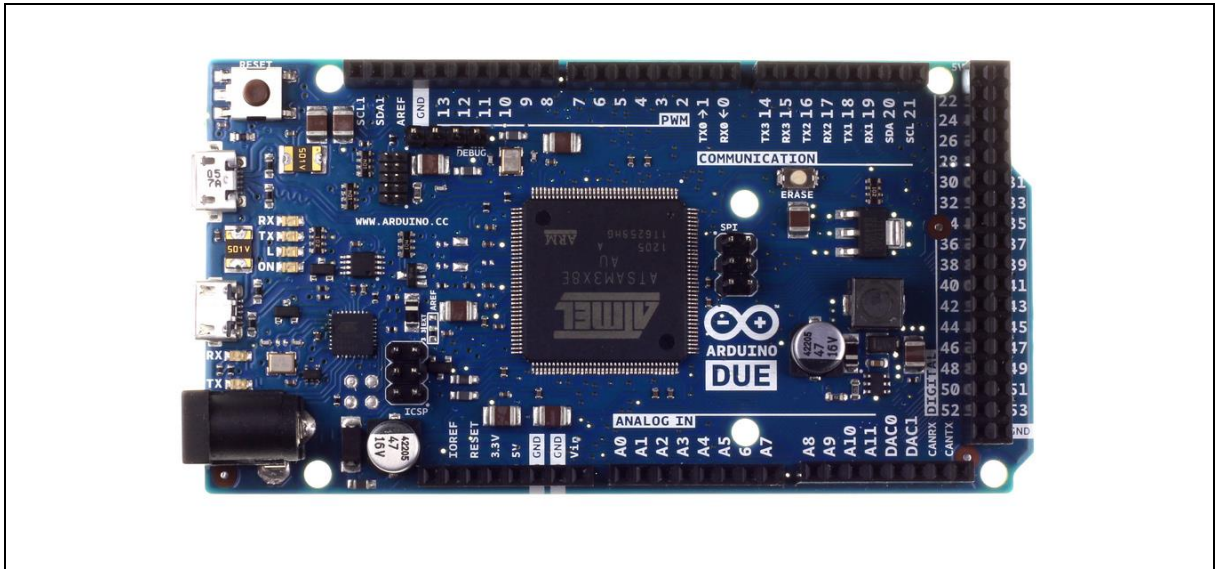


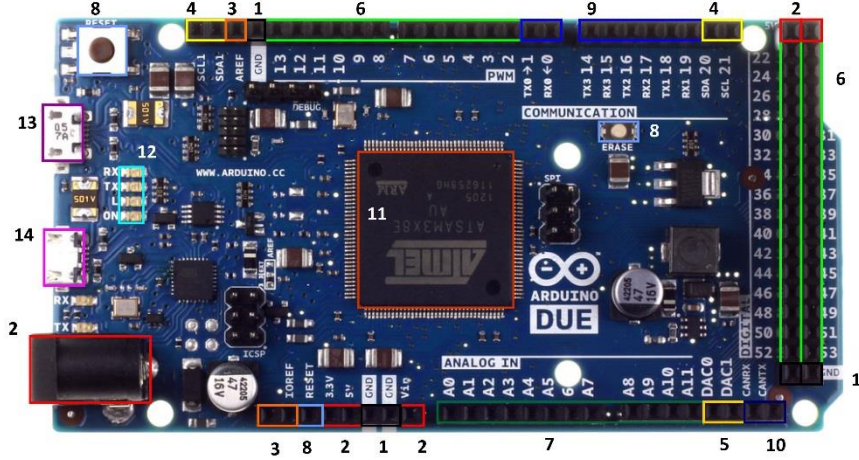
Figura 5. Arduino Due

Fonte: ARDUINO, 2013

Variando entre modelos com micro controladores menos potentes até modelos com alto poder de processamento, o Arduino é uma ótima opção para a prototipagem de um rastreador, por exemplo, com um Arduino Due é possível realizar a integração do GSM e GPS *Shield* além de ainda poder realizar telemetria no meio em que ele atua. (ARDUINO, 2013)

2.2.1 Hardware do Arduinio Due

“O Arduino Due é uma placa de prototipagem com um micro controlador baseado no ATMEL SAM3X8E ARM Cortex M3. Esta é a primeira placa Arduino com um microcontrolador ARM 32 Bits” (ARDUINO, 2013). Segundo o grupo a placa contém tudo que é necessário para suportar o microcontrolador ARM, precisa-se apenas ser conectado a um computador (caso queira realizar a programação) com cabo micro USB (*Universal Serial Bus*) ou liga-lo a um adaptador AC/DC (*Alternating Current / Direct Current*). Esta placa possui um ótimo desempenho e uma grande variedade de E/S (Entrada/Saida) que podem ser analisados no Quadro 1.

Imagem	
Legenda	<ol style="list-style-type: none"> 1. <u>Preto</u>: GND (<i>Graduated Neutral Density Filter</i>) conhecido como terra; 2. <u>Vermelho</u>: Representa Fonte externa de alimentação, Tensão 3.3V 800 mA, Tensão 5V e Pino de alimentação (Atua com <i>Shields</i>); 3. Laranja: IOREF e AREF que representam o leitor de referência para as entradas analógicas respectivamente; 4. Amarelo Claro: Pinos TWI 1 e 2 utilizados na comunicação WIRE; 5. Amarelo Escuro: Pinos que fornecem saídas. 6. Verde Claro: Pinos digitais de Entrada/Saída. 7. Verde Escuro: Pinos Analógicos que 8-bits de resolução. 8. Azul Claro: Utilizado para reiniciar e apagar o conteúdo da memória. 9. Azul Escuro: Pinos seriais num total de 4, sendo duplas de RX/TX, usados para transmitir dados seriais. 10. Azul Marinho: Pinos que suportam o protocolo CAN de comunicação. 11. Laranja Escuro: Microcontrolador Atmel SAM3X8E baseado em ARM Cortex-M3, núcleo de 32 bits que permite operações de 4 bytes de dados dentro de um único nível, com <i>clock</i> de 84Mhz e 96 Kbytes de SRAM ainda possui 512 Kbytes de memória flash para o código. 12. Azul Aqua: Leds de representação da placa. 13. Púrpura: Porta universal serial de utilização. 14. Magenta: Porta universal serial de programação, está porta é utilizada para programar o microcontrolador.

Quadro 1. Hardware Arduino Due

Fonte: ARDUINO, 2013

2.2.1.1 Microcontrolador

Como já mencionado, a placa de prototipagem Arduino Due possui um microcontrolador da ATMEL baseado nos microcontroladores ARM Cortex-M3, esse trabalha numa frequência de 84MHz e possui um conjunto de periféricos interno que apresentam uma grande variedade de conexão através das interfaces *Ethernet, dual CAN, HS, miniHost USB*. A diversas características que se pode ser observado no Quadro 2.

Microcontrolador	AT91SAM3X8E
<i>Clock</i>	84 MHz
Memória Flash	512 Kbytes
Quant. Pinos	144 pinos
Interface USB	<i>HOST, device</i>
Quant. SPI	04
Quant. TWI (I2C)	02
Quant. UART	05
Quant. CAN	02
Quant. LIN	03
Quant. SSC	01

Quadro 2. Pinos microcontrolador Cortex-M3

Fonte: (ATMEL, 2013)

“Arduino Due pode ser alimentado através de um cabo USB ou uma conexão externa. A fonte de alimentação é selecionada automaticamente” (ARDUINO, 2013). Segundo Oliveira e Andrade (2006), todo equipamento eletrônico precisa ser alimentado (termo utilizado para o ato de fornecer energia elétrica para o sistemas). Como fonte de energia há a possibilidade de uso de transformadores, fontes, pilhas e baterias, assim todos os meios de energia passam pelo regulador de tensão do dispositivo que vai prover a tensão correta para todo o circuito.

No caso do Arduino Due é possível também realizar a alimentação da placa adicionando a adaptação de uma bateria nos pinos GND e Vin, embora essa opção não seja aconselhada. O ideal é realizar a alimentação com uma fonte externa de 6 a 20 volts, pois esta energia ao passar no regulador de tensão será transformada em 3.3V e 5V que são o ideal para se trabalhar no protótipo. Se for fornecido uma energia abaixo de 6V a placa se torna instável e poderá reiniciar várias vezes, se for fornecido uma energia superior a 20 volts o regulador de tensão aquecerá e danificará o circuito.

Segundo Arduino (2013), como se pode observar no Quadro 1, os pinos de energia do Arduino Due são:

- VIN: Este pino é uma entrada para alimentação direta no circuito, geralmente utilizado quando a alimentação está ativa em um *Shield*.
- 5V: Este pino gera tensão 5V. A placa pode ser alimentada com energia a partir da tomada DC (7 – 12V), conector USV (5V) ou do pino VIN (7 – 12V). Fornecimento de alimentação direto nos pinos de 5V e 3.3V pode danificar a placa.
- 3.3V: Tensão de alimentação 3.3V, com corrente máxima de 800 mA. Segundo Oliveira e Andrade (2006), a tensão pode ser considerado a força da energia e a corrente como a quantidade de energia consumida. Sendo assim se for adicionado algum periférico que consuma mais que 800mA de corrente, este pode não funcionar.
- GND: Pinos terra ou negativos.

- **IOREF:** Este pino fornece a referência de tensão com qual o microcontrolador opera. Assim *Shields* podem identificar a tensão que devem trabalhar para não danificar o microcontrolador.

2.2.1.2 Memória

“O SAM3X tem 512KB (dois blocos de 256 KB) de memória flash para armazenamento de código. O *bootloader*³ é produzido na fábrica da *Atmel* e é armazenado em uma memória ROM dedicado” (ARDUINO, 2013).

A memória flash é aonde será gravado o *firmware*, ou seja, é aonde será gravado o sistema operacional que terá como limitação o tamanho de memória flash, isso será explicado mais à frente. Sendo assim a memória só será limpa quando for acionado o botão *ERASE*.

2.2.1.3 Entrada e Saída

Conforme Arduino (2013) a plataforma dispõem de 54 pinos digitais sendo que 12 oferecem saída PWM com resolução de 8 bits, esses pinos denominados pinos digitais se comportam recebendo e transmitindo níveis digitais como se fossem ativados e desativados, assim trabalhando com atuadores, *leds*, motores e sensores. A capacidade de operação deles é de 3,3V, cada pino pode oferecer de origem uma corrente de 3mA a 15mA e suporta uma corrente de 6mA a 9mA.

Como pode ser observado no Quadro 1, alguns pinos possuem funcionalidades especiais tais como:

- **Pinos Seriais:** Os pinos seriais são utilizados para receber (RX) e transmitir (TX) dados seriais. O pino 0 e 1 são diretamente ligados ao microcontrolador para realizar a programação e comunicação direta.
- **Pinos SPI:** Utilizado para realizar comunicação direta com outros microcontroladores.

³ Código de inicialização.

- Pinos CAN: pinos RX e TX que realizam a comunicação com periféricos utilizando o protocolo CAN de comunicação.
- Pino L: Pino que define o status do LED L acoplado a placa. Quando o pino se encontra ativo o LED estará acesso, quando o pino estiver desativado LED está apagado.
- Pinos TWI: Pinos 20 (SDA), 21(SCL) e SDA1, SCL1 que são utilizados na comunicação WIRE que é padrão em alguns periféricos.
- Pinos Analógicos: São pinos que permitem a leitura analógica, geralmente com 10 Bits de resolução que equivale um valor igual a 1024 unidades, ou seja 0V é igual 0 e 3,3V igual 1024. Se passar de 3,3V poderá danificar o microcontrolador.
- Pinos DAC: Pinos que fornecem saídas PWM (analógicas) com resolução de 12 Bits (valor igual a 4096).
- Pino AREF: Pino utilizado para obter a referência de tensão analógica, adquirindo o valor de tensão e calculando ele para o valor máximo em Bits.
- Pino RESET: Pino utilizado para reiniciar o microcontrolador, geralmente utilizando por *Shields* que obstruem o botão de *reset*.

2.2.1.4 Comunicação

O Arduino Due possui diversos facilitadores na comunicação com o computador e outros periféricos, havendo a possibilidade de se conectar por diversos padrões seguindo vários protocolos. Entre o Arduino Due e o computador realiza-se uma conexão USB como se fosse um celular, câmera ou *Tablet*. “A porta de programação está conectada a um Atmega16U2, que fornece uma porta COM virtual para o software em um computador conectado. [...] O 16U2 também está ligado ao hardware UART do SAM3X” (ARDUINO, 2013). A comunicação realizada entre o computador e o SAM3X passa pelo microcontrolador 16U2 que será responsável por tratar especificamente disto, assim, realizando tarefas como:

- Identificar uma conexão;
- Criar um porta virtual COM para conexão;

- Preparar o SAM3X para o tipo de comunicação que se deseja realizar;
- Realizar a conversão do padrão USB para o serial.
- Enviar Bits de dados.

Quando ocorre uma comunicação que ativa o microcontrolador 16U2, os Leds RX e TX piscam ao realizar a troca de dados entre os microcontroladores.

A placa ainda dispõe de outra porta USB nativa que está diretamente ligada ao microcontrolador SAM3X. “Ela permite a comunicação serial através do USB” (ARDUINO, 2013). Além também de pinos que tornam possíveis as comunicações através dos padrões USARTS, SPI, TWIS e I2S.

Comunicação Serial

“A comunicação serial é a forma de transmissão de dados mais comum entre dispositivos embarcados e um computador pessoal. [...] Ela possui dois canais de transferência de dados, um para envio (Tx) e o de recebimento (Rx)” (OLIVEIRA; ANDRADE, 2006). No Arduino Due pode ocorrer a comunicação serial com outros periféricos, logo é necessário o uso de um cabo *cross-over* que segundo Oliveira e Andrade (2006), é um cabo que coloca o pino de transmissão (Tx) do sistema embarcado com o pino de recepção (Rx) do outro dispositivo e vice-versa.

Comunicação USB

O Arduino Due possui duas comunicação USB, uma para programação e outra nativa.

A comunicação USB acontece sempre entre dois dispositivos, o Host e a aplicação. O primeiro é responsável por detectar a inserção ou remoção de algum dispositivo (aplicação). Ele gerencia o fluxo de dados e a interface elétrica entre ele e a aplicação. Cada Host suporta 127 dispositivos. (ARDUINO, 2013)

O Due utiliza conectores micro USB, conforme na Figura 6, sendo que “os cabos conectores USB possuem quatros fios condutores, dois para alimentação e dois para transmissão de dados. Os fios de alimentação (Vbus e GND) são utilizados para alimentar a aplicação em +5V, quando for necessário” (OLIVEIRA; ANDRADE, 2006).

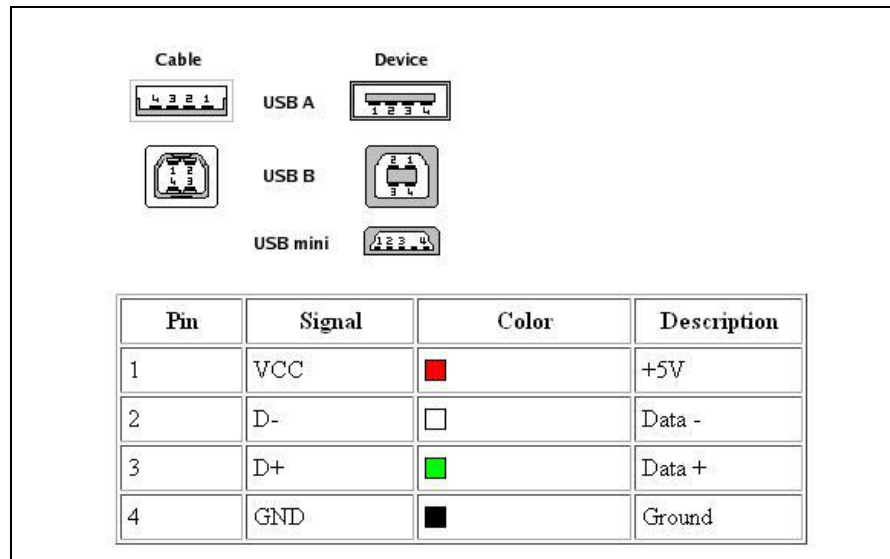


Figura 6. Pinos conectores USB

Fonte: (UCONTROL, 2013)

Segundo Arduino, (2013) a porta USB nativa do Arduino também pode atuar como um *HOST* USB para periféricos conectados como mouses, teclados e *Smartphones*. Mas segundo Oliveira e Andrade, (2006) para o host USB ter um bom funcionamento recomenda-se cabos com tamanho máximo de cinco metros, quando passar disso é preciso usar outros protocolos entre a extremidade, como o *Ethernet* e o RS-485.

Universal Asynchronous Receiver Transmitter (UART)

“É um método de comunicação *full-duplex*, ou seja, com dois canais de comunicação independentes que podem operar ao mesmo tempo” (OLIVEIRA; ANDRADE, 2006). Ambas operam independentemente, porém, possuem a mesma taxa de transferência e tamanho de dados, sendo assim, a comunicação funciona de forma serial. Esse método possui alguns Bit acrescentados que controlam um início de dado e um fim de dado.

2.2.2 Software do Arduinio Due

O Arduino Due pode ser programado utilizando a IDE (*Integrated Development Environment*) do Arduino que é “um software livre no qual você escreve o código na linguagem que o Arduino compreende (baseado na linguagem C)” (MCROBERTS, 2011). A IDE do Arduino permite que seja escrito um conjunto de instruções que ao ser carregadas na memória do microcontrolador serão executadas.

2.2.2.1 Linguagem de Programação

A linguagem utilizada na programação do Arduino é a C/C++ com algumas diferenciações. “Um sketch do Arduino deve ter uma função *setup()* e uma função *loop()*, do contrário, não funcionará” (MCROBERTS, 2011). A função *setup()* somente é executado no início do programa e a *loop()* é uma funções constantemente executada até que seja anulada. Além dessa característica possui outras variação que são as adaptações de métodos para o Arduino como os controles dos pinos de entrada e saída.

2.2.2.2 Controle dos Pinos de Entrada e Saída

Como o Arduino Due é uma placa com diversas entradas e saídas é natural que exista um modo de controla-las através de linha de códigos, sendo assim é possível realizar a análise de diversas funções mostradas a seguir:

- **PinMode (pino, modo):** Realiza a configuração do pino desejado para se comportar tanto como entrada ou como saída, neste caso são passados dois parâmetros que são o número do pino no Arduino Due e o modo (INPUT/OUTPUT);
- **Digital Read (pino):** Comando que realiza a leitura doe estado o pino (HIGH/LOW), o Due possui 54 pinos digitais que é possível realizar a leitura;
- **DigitalWrite(pino, valor):** Comando que realiza a escrita no pino, passando o valor desejado, neste contexto, o valor é um valor lógico (HIGH/LOW);
- **AnalogRead (pino):**em 12 pinos é possível realizar a leitura analógica, ou seja, eles trabalham com leitura de tensão e consegue reconhecer valores de 0V à 3.3V que equivale a 1024. Para realizar a leitura basta usar o comando indicando o pino analógico;
- **AnalogWrite (pino, valor):** em 14 pinos do Due é possível realizar saídas de tensões tipo analógicas, 12 pinos possuem resolução de 10 Bits e 2 resolução de 12 Bits. Para realizar a escrita no pino basta usar esse comando passando o pino e o valor (0~1024 ou 0~4096);

- `AnalogReadResolution` (valor): Configura a resolução em Bits para a leitura analógica no pinos analógicos do Arduino. Por padrão é 10 Bits (valor de até 1023) o que o torna compatível com outros Arduino.
- `AnalogWriteResolution` (valor): Configura a resolução em Bits para as saída analógicas com resolução de 8 Bits (255) por padrão.

2.2.2.3 Bibliotecas

Existe milhares de bibliotecas disponíveis para o Arduino, e todas incorporam padrões de comunicação ajudando no trabalho com hardware e na manipulação dos dados. Para Mcroberts, (2011) uma biblioteca é simplesmente um conjunto de código, oferecendo funcionalidades que, do contrário, seriam criado do zero. Tal prática representa o princípio da reutilização de código e ajuda a acelerar no processo de desenvolvimento. No Arduino Due e especificamente neste projeto de rastreador, possivelmente serão utilizados bibliotecas como:

- `GSM.h`
- `SD.h`
- `SPI.h`
- `WIRE.h`
- `USBHOST.h`

A biblioteca `GSM`, segundo Arduino, (2013) permite que uma placa Arduino possa realizar a maioria das operações iguais a de um telefone celular, ou seja, realizar e receber ligações, enviar e receber SMS e conectar-se a internet através do GPRS. A biblioteca possui o resumo de comunicação de baixo nível entre o modem e o cartão SIM (*Subscriber Identity Module*) além de comandos AT usados no módulo. Dividida em 10 classes, possui todos os métodos necessários para realizar a abstração de montagem dos comandos.

A biblioteca `SD` possui todas os comandos e operações necessárias para realizar a comunicação com um cartão SD e suporta arquivos de sistemas tipo FAT16 e FAT32. A biblioteca realiza a comunicação entre o microcontrolador e o cartão SD através de barramento SPI, assim essa biblioteca também utiliza o `SPI.h`.

A SPI é uma biblioteca que permite a comunicação com periféricos usando o Arduino como dispositivo mestre, sendo assim, essa biblioteca é “um protocolo síncrono de dados em série utilizado por microcontroladores para comunicar com um ou mais dispositivos periféricos rapidamente ao longo de distâncias curtas” (ARDUINO, 2013). Também através dele é possível realizar a comunicação entre dois microcontroladores. A biblioteca WIRE é mais utilizada na comunicação com sensores e atuadores, ela “permite a comunicação com dispositivos I2C”.

2.2.3 Shields para Arduino Due

O Arduino Due possui diversas entradas e saídas o que possibilita a conectividade com diversos periféricos de diversos gêneros. Segundo Mcroberts, (2011) o Arduino também pode ser estendido utilizando os *Shields*, que são placas de circuitos contendo outros dispositivos, (por exemplo, receptores GPS, *display* de LCD, módulos de Ethernet etc.) assim estendendo as funcionalidades. Entre a diversidade de *Shields* existentes serão utilizados basicamente dois tipos neste projeto, são eles o GSM/GPRS *Shield* e o GPS *Shield*.

2.2.3.1 GSM/GPRS shield

O *Shield* GSM/GPRS é um responsável pela a comunicação dos dados com as operadoras telefônica, “o celular *Shield* para Arduino inclui todas as peças necessárias para realizar a interface entre o Arduino e um módulo de celular [...]. Isso permite adicionar facilmente funcionalidades como SMS, GSM/GPRS, TCP/IP ao projeto baseado em Arduino” (SPARKFUN, 2013). Para tonar possível a utilização das funcionalidades é necessário um chip telefônico das operadoras denominado SIM e uma antena como pode ser visto na Figura 7.

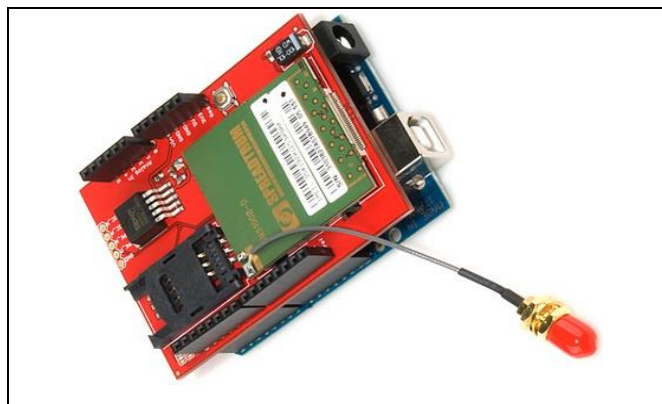
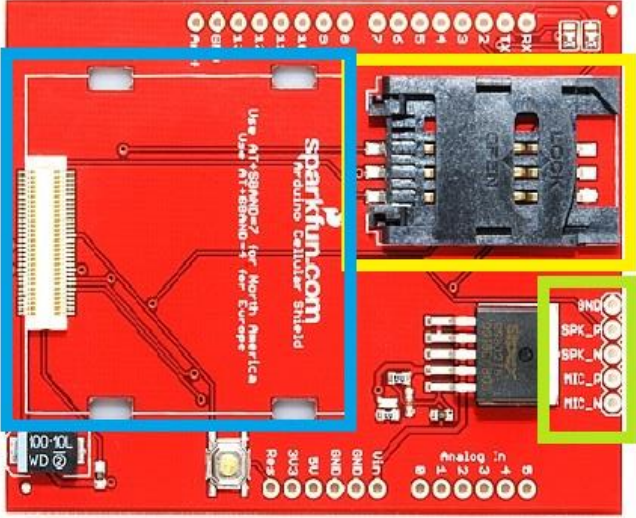


Figura 7. Celular Shield

Fonte: (SPARKFUN, 2013)

Segundo Arduino, (2013) para haver uma conexão com a rede telefônica é necessário a assinatura com uma operadora telefônica (pré-pago ou pós-pago), um dispositivo compatível com a rede GSM/GPRS e um módulo de identificação de Assinante (SIM *card*), este último possui o número e o DDD de funcionamento.

Há diversos Shields como GSM/GPRS, 3G e 4G. Como este projeto apenas necessita de um conexão GSM/GPRS e visto que este possui maior cobertura, foi escolhido como padrão a análise dos *Shields* GSM/GPRS e como modelo de exemplo o *Shields* celular da Sparkfun (SPARKFUN, 2013).

Imagem	
Legenda	<ol style="list-style-type: none"> 1. <u>Azul</u>: Local reservado para módulo de comunicação; 2. <u>Amarelo</u>: Local reservado para chip SIM; 3. <u>Verde</u>: Conectores de microfone e alto falantes.

Quadro 3. GSM/GPRS *Shield*

Fonte: (SPARKFUN, 2013)

Pelo Quadro 3 é possível ver a localização dos componentes fundamentais para o funcionamento de um *Shield* GSM/GPRS, percebe-se que a única coisa que vai diferenciar entre os diversos *Shields* do mesmo seguimento é o módulo, sendo que o módulo é o responsável pelos protocolos de comunicações realizado entre o Arduino e as operadoras.

2.2.3.2 GPS Shield

Os *Sshields* GPS atuam adquirindo informações de satélites e realizando monitoramento na localização, no tempo e na velocidade. Acoplado sobre um Arduino, esses *Shields* são necessário para realizar a aquisição de informações sobre a localização atuando como um rastreador.

Há diversos *Shields* como nos *GSM/GPRS Shield*, a única diferença entre eles é os módulos de GPS que praticamente atuam igualmente. O modelo escolhido como exemplo foi o EM406 da *Sparkfun* (SPARKFUN, 2013).

Conforme na Figura 8 é possível notar a estrutura de um *GPS Shield*.

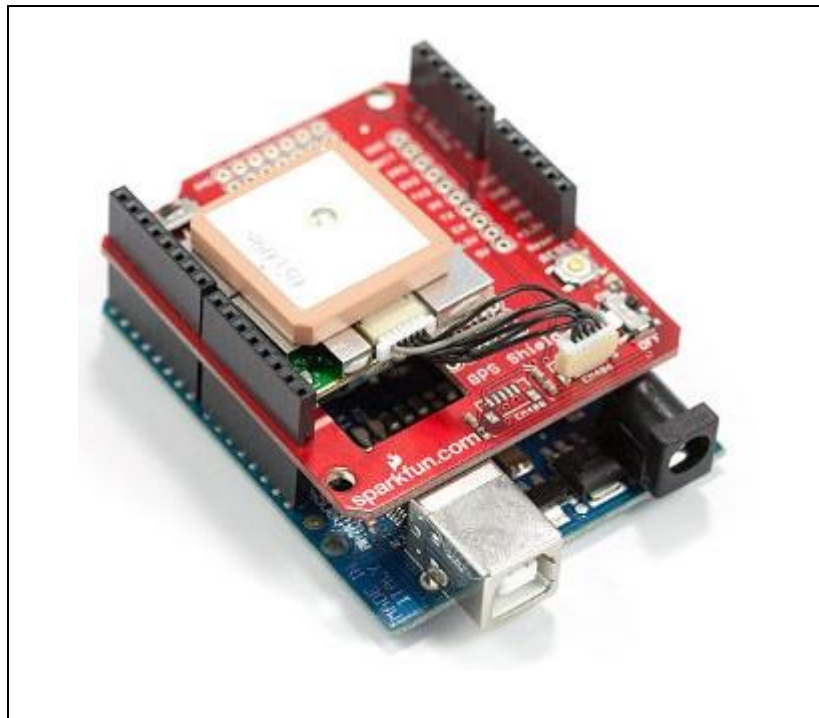


Figura 8. GPS Shield

Fonte: SPARKFUN, 2013

2.3 SISTEMAS OPERACIONAIS DE TEMPO REAIS

O sistema operacional “é uma camada de software colocada entre o hardware e os programas que executam tarefas para os usuários. [...] O sistema operacional é responsável pelos periféricos” (OLIVEIRA; CARISSIMI; TOSCANI, 2008). Analisando a capacidade do Arduino Due e moldando o mesmo como um rastreador tendo em vista que este possui várias

E/S e a conexão com diversos periféricos, há a necessidade de um melhor gerenciamento dos recursos com a utilização de um SO.

Os Sistemas Operacionais de tempo real são, segundo Tanenbaum, (2011) sistemas caracterizados por terem o tempo como um parâmetro fundamental. “O processamento em tempo real acontece num sistema cujo funcionamento se dá não apenas pela execução correta das operações realizadas pelo processador, mas também por um tempo ideal de processamento dessas operações” (OLIVEIRA; ANDRADE, 2006). O grupo ARM, (ARM, 2013) considera que a maioria dos sistemas embarcados necessitam de um software responsável em tratar as entradas de eventos dentro de um período de tempo definido e que estes podem ser caracterizados como sistemas de tempo real críticos e não críticos. “Tais sistemas podem ser categorizadas como *Hard Real-Time*, onde perder um prazo de resposta é inaceitável [...] e *SoftReal-Time*, onde bater um prazo é desejável, mas não essencial. Em ambos os tipos de sistema, um grau de determinismo é importante” (ARM, 2013) Com a análise dessas afirmações é possível notar que no caso dos rastreadores são utilizados Sistemas Operacionais não críticos, estes são sistemas de tempo real que possuem certas tolerâncias no descumprimento do tempo.

Há diversos Sistemas Operacionais de tempo real comerciais disponíveis no mercado e como o protótipo será constituído de um microcontrolador ARM e busca utilizar somente códigos-fontes abertos, serão listados os mais conhecidos que atende a esses critérios a seguir.

2.3.1 Sistemas Operacionais comerciais

Segundo o grupo ARM, (ARM, 2013) todos os principais fornecedores de Sistemas Operacionais de tempo real possuem apoio a microcontroladores ARM e a maioria possuem funções de nível além do *kernel*, funções como gerenciamento de dispositivos (USB, UART, Ethernet, LCD, etc.), sistemas de arquivos, pilha de protocolos (CAN, TCP/IP, HTTP e etc.) e interfaces gráficas com usuários (GUI). Na tabela 1 é possível notar os seis Sistemas Operacionais mais conhecidos disponíveis para microcontroladores ARM córtex-M3

Tabela 1. SO que suportam o ARM córtex-M3

Companhia	RTOS	Compatível
Altreonic	OpenComRTOS	SIM
eForce	μC3	SIM
Express Logic	ThreadX	SIM
FreeRTOS.org	FreeRTOS	SIM
ChibiOS/RT	ChibiOS	SIM
RT-Thread.org	RT-Thread RTOS	SIM

Fonte: (ARM, 2013)

Entre os seis SO apresentados, 2 possuem o maior destaque pelo fato de possuírem mais documentações e mais exemplos de utilização, são eles, o FreeRTOS e ChibiOS/RT.

Segundo o grupo FreeRTOS, (2013) este é um sistema operacional de tempo real pequeno e objetivo utilizado em vários sistemas embarcados. Ele tem a capacidade de gerenciar múltiplas threads, sinalizações com semáforos e temporizadores. Este SO de tempo real ainda realiza a o gerenciamento de memória e escalonamento de tarefas em dois tipos, a preemptiva onde se tem prioridades na execução das tarefas (variando de 0 que é a espera ociosa e 3 com maior prioridade) e a cooperativa onde a troca de tarefa é dado apenas quando uma é bloqueada. O FreeRTOS possui uma API com um total de 89 funções que são representadas a seguir:

- Criação de tarefa;
- Controle de tarefa;
- Utilidades de tarefas;
- Controle de *Kernel*;
- Controle de MPU();
- Controle de filas;
- Controle de conjuntos de filas;
- Controle de semáforos e Mutex;
- Controle de Timer de Software;
- Funções de co-rotinas.

O ChibiOS/RT “é um Sistema Operacional de tempo real completo, portátil, de código-fonte aberto, compacto e extremamente rápido” (CHIBIOS, 2013). Entre as principais características do SO estão a realização *MultiThreading*, 128 níveis de prioridades, Escalonamento *Round-Robin* para *thread* com o mesmo nível de prioridade, gestão de memória, suporte a mensagens síncronas e assíncronas e abstração de hardware. O chibiOS possui uma API com mais de 100 funções divididas em:

- Controles de registros;
- Controles internos;
- Serviços do *kernel*;
- Fluxos e arquivos;
- Sincronizações;
- Tipos;
- Número de versões e identificação;
- Configurações;
- Debug;
- Gerenciamento de memória.

2.3.2 Comparativo entre os Sistemas Operacionais

Para realizar a escolha de um sistema operacional é necessário a avaliação de critérios pré-definidos que venha a ter características que torne possível o desenvolvimento de um rastreador.

“Os principais componentes do *kernel* de qualquer Sistema Operacional são a gerencia de processador, a gerencia de memória, os sistemas de arquivos e a gerencia de entrada e saída” (OLIVEIRA; CARISSIMI; TOSCANI, 2008). Com base nessa afirmação os critérios de avaliação são o agendamento de tarefas, o tratamento de memória e gerenciamento de hardware.

Na comparação do agendamento de tarefa entre o FreeRTOS e o ChibiOS os dois modelos apresentam nível de hierarquia na escolha de tarefas, sendo que o ChibiOS possui 128 níveis de prioridade contra 3 do FreeRTOS, ainda, os dois modelos possuem escalonamento tipo Round-Robin para tarefas com mesma prioridade. Nessa verificação o ChibiOS apresentou melhores resultados.

No tratamento de memória nota-se que os dois sistemas trabalham de formas diferentes, o freeRTOS trabalha com alocação dinâmica, já o ChibiOS trabalha com tamanhos fixos no uso da memória. Apesar dessa diferenças ambos apresentam igualdade na comparação.

Em questão de gerenciamento do hardware o ChibiOS apresentou melhor abstração e adaptação com protocolos de comunicações, realizando criação de camadas para realizar o tratamento delas. Assim com a análise desses parâmetros, o Sistemas Operacional que melhor se adapta com o modelo de rastreador proposto é o ChibiOS.

2.4 PROJETOS SIMILARES

Vários grupos de pesquisa trabalham em projetos de rastreamentos, realizando a análise desses projetos foi possível avaliar exemplos de aplicações similares a proposta e pode-se verificar que existem no mercado variados tipos de rastreadores, que foram surgindo conforme as pesquisas foram avançando e as necessidades se apresentando.

2.4.1 Sistemas de rastreamento embarcados

O objetivo deste projeto é mostrar a funcionalidade de um sistema de rastreamento veicular. Assim inicialmente, os rastreadores de sistema GPS mostraram-se limitados quanto à precisão de mais ou menos 100 metros, não obtendo uma posição exata. Através da pesquisa este quadro foi se alterando, a tecnologia evoluiu e o sistema de rastreamento tornou-se mais preciso e eficazes.

Conforme Adriano, Herzog e Lopes, (2009) em seu artigo intitulado Sistema de Rastreamentos Embarcados já existe no mercado quatro grupos de tecnologia de rastreamento, que são:

- a) Satélite+GPS:O posicionamento do veículo é feito pelo GPS e a transmissão do posicionamento é feita por meio de um satélite específico;

- b) GSM² + GPRS³ + GPS: O posicionamento é feito via GPS, porém as transmissões das informações são feitas por meio de banda de telefonia celular GSM /GPRS;
- c) Rádio frequência: triangulações feitas a partir do percurso do veículo e que transmitem para uma centra de processamento;
- d) Pager/RDS4: Também funciona a base de antenas ao longo do percurso do veículo, sendo somente utilizada para envio de sinal da central para bloqueio do veículo. Atualmente devido sua vasta cobertura e custo mais acessível, utiliza-se a tecnologia GPS+GSM;

Quanto ao funcionamento do sistema existe uma unidade embarcada que contém o receptor GPS, transmissor/receptor GSM, que seria equivalente a um celular para transmissão de dados e um ou mais microcontrolador para gerenciar todas as informações. Este módulo possui interface com outros subsistemas do veículo, tais como alarme, bateria, bloqueio, etc. E por final deve haver um servidor que execute a recepção desses sinais emitidos e ou enviados com a unidade de monitoração, através, por exemplo, em forma de e-mails, SMS ou através da internet.

Quanto aos equipamentos, para um bom funcionamento, seria um sistema GPS/GSM que viria a necessitar basicamente de uma unidade (módulo) de processamento, que contém o receptor GPS e o receptor/transmissor GSM, além de outras interfaces com subsistemas do veículo (alarme, bloqueio etc.). A resolução atual que diz respeito ao sistema de rastreamento veicular é a Resolução do CONTRAN Nº 245 de 27 de julho de 2007 (ADRIANO; HERZOG; LOPES, 2009).

Os autores mostram também neste artigo o objetivo do sistema de rastreamento que seria a: monitoração de um veículo, empresas logísticas e proteção de furtos com a utilização de rastreadores já disponíveis no mercado.

2.4.2 Sistemas de monitoração veicular via GPRS

Uma área tecnológica que está em constante atualização e que envolve inúmeras aplicações é a rede de telefonia celular. Esta área foi tema de um trabalho monográfico desenvolvido por Oliveira, Pitombo e Alegre, (2008). Segundo os autores essa rede possui as

várias gerações sendo que a primeira geração (1G) que é analógica e de banda estreita. A segunda geração (2g) que é digital e de banda estreita, utilizando GSM, CDMA e TDMA. A terceira geração (3g) que é digital e de banda larga que trouxe grandes mudanças na internet móvel.

Os autores tiveram por objetivo propor um sistema de monitoramento veicular via GPRS, apresentando em primeiro lugar as tecnologias envolvidas. Observaram que o GPRS é uma das tecnologias da 2,5g que possibilita uma conexão instantânea sendo que a informação é rapidamente enviada ou recebida segundo a necessidade de utilização. Nessa tecnologia o meio de transmissão está sempre disponível para o usuário. O módulo XT65 da Siemens agrega as tecnologias GPRS e GPS, assim desenvolve soluções de telemetria veicular através da linguagem Java, com esse equipamento é possível receber e enviar informações para o veículo equipado, sabendo, por exemplo, a sua coordenada e velocidade em tempo real.

Conforme os autores, para atingir o objetivo principal os seguintes passos deveria ser realizados: pesquisar o *Datasheet* do módulo XT65 e demais documentos para a correta utilização dos recursos oferecidos pelo Java, aprendendo a programação Java, pois o módulo XT65 permite a utilização dessa plataforma e entende-se como sendo uma forma mais acessível e rápida de programá-lo. Adquirir conhecimento sobre a plataforma Netbeans Ide 6.0, para o desenvolvimento da aplicação embarcada no módulo, tomando conhecimento da tecnologia de comunicação GRPS para o envio de informações via telemetria e interpretando as coordenadas vindas do GPS (protocolo NMEA), assim como os sinais de velocidade, documentação do motorista e chassi do veículo provenientes do sistema micro processado. Desenvolver uma pesquisa sobre a ferramenta Wireless Toolkit 2.5.2 for CLDC, pois serviria para simular a aplicação do módulo além de configurar um servidor WEB, com o objetivo de sustentar a aplicação Web que seria desenvolvida.

Outro fator referente à pesquisa é prender a linguagem de programação PHP, devido a sua utilização no desenvolvimento das aplicações web, tomando conhecimento da ferramenta *dbdesigner*, que seria utilizado para construção da modelagem do banco de dados para a aplicação web e assim aprender sobre desenvolvimento de banco de dados com o *postgresql*, pois o sistema utilizará essa ferramenta para o armazenamento de dados oriundos do veículo. Pesquisar sobre a linguagem *plpgsql*, que é a linguagem procedural do *postgresql*.

Segundo os autores, na realização do trabalho, utilizaram a linguagem de programação Java para interpretar os sinais do sistema micro processado no veículo (velocidade, localização, documentos do condutor e do veículo) e a tecnologia GPRS para telemetria dessas monitorações.

Em resumo os autores realizaram a criação de um rastreador utilizando o estudo de seus componentes, realizaram a validação desse com o trafego de informações para uma base de dados e por fim realizaram o monitoramento dessas informações através de um sistema web.

3 DESENVOLVIMENTO

Para alcançar o objetivo proposto, foi realizado a divisão do projeto em duas partes: (a) desenvolvimento do *Hardware*; e (b) desenvolvimento do *Software*.

Os processos de desenvolvimento seguiram modelagens e requisitos criados que mostram o essencial para que o sistema seja validado, assim no desenvolvimento do hardware ocorreu a montagem e conexão do Arduino Due aos *Shields* GSM e GPS, realizando a conexão entre pinos e definição de protocolos de conexão. O objetivo desse desenvolvimento foi criar um protótipo que permita realizar a programação do *Software* e a adaptação do SO.

O desenvolvimento do *Software* ocorreu em sequência. Realizou-se a produção e adaptação do SO com base na criação dos requisitos funcionais e não funcionais, diagramas e no plano de testes, validando a utilização dos pinos, a utilização dos protocolos de comunicações e o protótipo em si.

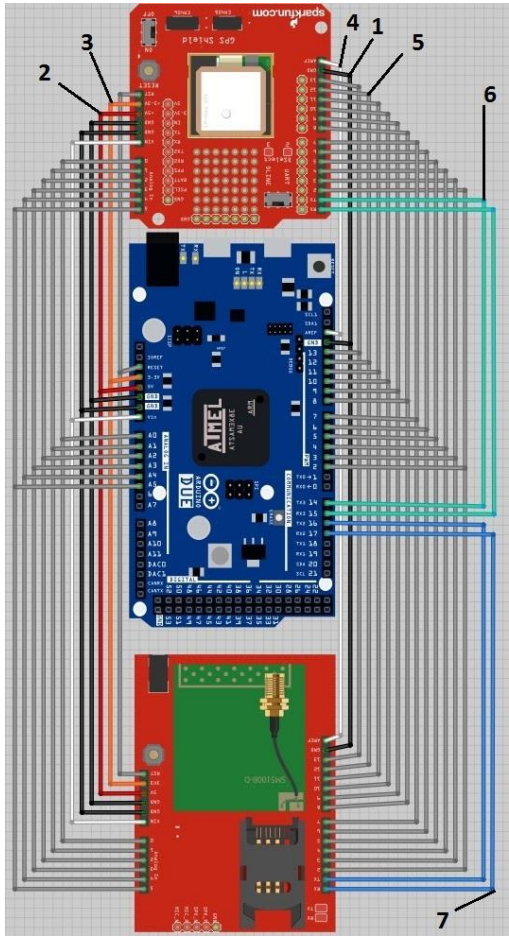
3.1 DESENVOLVIMENTO DO HARDWARE

Nesta etapa foi realizado o criação do protótipo em relação aos seus aspectos físicos, criando as conexões de alimentação e comunicação entre o Arduino Due e os *Shields*. O protótipo se baseou na modelagem no Quadro 4.

A modelagem realizada consiste basicamente no levantamento do hardware utilizado e no seu diagrama elétrico. É possível notar que a modelagem foi realizada com 3 placas eletrônicas que são: o Arduino Due; o GSM/GPRS *Shield*; e o GPS *Shield*. Para estes dois últimos, foram utilizados os modelos de exemplos dos *Shields* da *Sparkfun* (SPARKFUN, 2013).

3.1.1 Esquema eletrônico

O esquema eletrônico proposto neste modelo consiste na conexão entre o Arduino Due e os *Shield*, isso devido ao fato de eles serem circuitos impressos que servem para realizar prototipagem. No Quadro 4 é apresentada uma análise das conexões realizadas e o significado de cada fio.

Imagem	
Legenda	<ol style="list-style-type: none"> 1. <u>Fio preto</u>: GND (<i>Graduated Neutral Density Filter</i>); 2. <u>Fio vermelho</u>: Representa a tensão 5V e Pino de alimentação 5V; 3. <u>Fio laranja</u>: Representa a tensão 3,3V e Pino de alimentação 3,3V; 4. <u>Fio branco</u>: Representa o pino de entrada de tensão (não utilizado) e o pino de referência analógico; 5. <u>Fio cinza</u>: Representam as conexões entre os <i>Shield</i>, embora não sejam utilizado pelos mesmos; 6. <u>Fio verde</u>: São os fios RX e TX de conexão UART do <i>GPS Shield</i>, estes serão conectados a portas com manipulação UART em hardware do Arduino Due; 7. <u>Fio azul</u>: São os fios RX e TX de conexão UART do <i>GSM/GPRS Shield</i>, estes serão conectados a portas com manipulação UART em hardware do Arduino Due.

Quadro 4. Esquema Eletrônico.

O esquema de conexão mostrado no Quadro 4 permite tornar a visualização da conexão do Shields simplificada através de fios virtuais. Um *Shield* é acoplado a cima do Arduino e as conexões são realizadas através de barramentos. Neste esquema é possível notar que os padrões dos barramentos dos Shields são da versão 1.0 e, segundo o grupo Arduino (ARDUINO, 2013), o Due é compatível com todos os *Shields* que trabalham com 3.3V e com o padrão de pinos versão 1.0. Foi realizado a modificação da conexão dos pinos 0 e 1 (RX e TX UART) dos *Shields*, conectando eles nos pinos 14, 15 (RX e TX serial 3) e 16,17 (RX e TX serial 2) tornando possível realizar a comunicação serial direta por hardware sem precisar da emulação por softwares e a utilização de pinos digitais.

3.1.2 Realização e montagem do protótipo

Primeiramente foram realizada as aquisições dos componentes assim especificados, sendo que se manteve a ideia inicial de utilizar o Arduino Due e o *Shield* GSM/GPRS e alterou-se a ideia de utilizar o *Shield* GPS para apenas o módulo GPS. Essa alteração foi necessária devido à disposição do módulo ser mais facilitada.

Em seguida, foi realizada a acoplagem dos periféricos através das conexões estipuladas, com pequenas modificações nas conexões e algumas modificações no *Shield* GSM. As modificações no *Shield* envolveram reposicionar os MOSFETS conversores de nível referentes aos sinais RX, TX, RX_DBG e TX_DBG e a adição de cinco jumpers para que pudessem trabalhar com a tensão específica de 3.3V exigida pelo Arduino Due, modificações que podem ser vista na Figura 9.

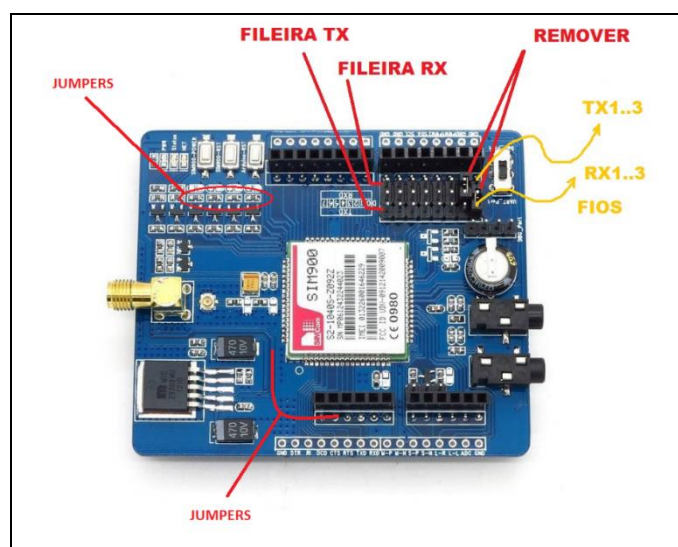


Figura 9. Alterações no GSM *Shield*

No caso da conexão seriais foi alterado os pino de conexão RX e TX no Arduino Due ficando na sequência:

- RX1 e TX1: Pinos seriais de comunicação com o módulo GPS;
- RX2 e TX2: Pinos seriais de comunicação com o *Shield* GSM;

Durante a montagem, verificou-se que seria necessário uma estrutura que guardasse os componentes com mais segurança, então utilizou-se uma caixa de madeira para tal, com 12 cm de altura, 22 cm de largura e 5 de comprimento. Conforme a Figura 10.



Figura 10. Modelo de rastreador.

No protótipo, foram adicionados 3 Leds que seguem o seguinte padrão:

- Led Verde: Status Sistema, se ligado sistema alimentado, se desligado sistema não alimentado;
- Led Azul: Status do Sistema Operacional, Se piscando a cada 1 segundo, o sistema Operacional está ativo e realizando escalonamento de processo;
- Led Vermelho: Status de Erro, acende se o sistema passar por um erro crítico e irreversível.

Na Figura 11 é possível notar a solução adotada para alocação dos dispositivos internamente, analisando também que ao ser alimentado é possível ver que os Leds indicativos de alimentação de cada periférico aparecem acessos. Percebe-se também os fios que foram utilizados para realizar conexão serial entre os periféricos, além de fios de alimentação e fios de conexão aos Leds.

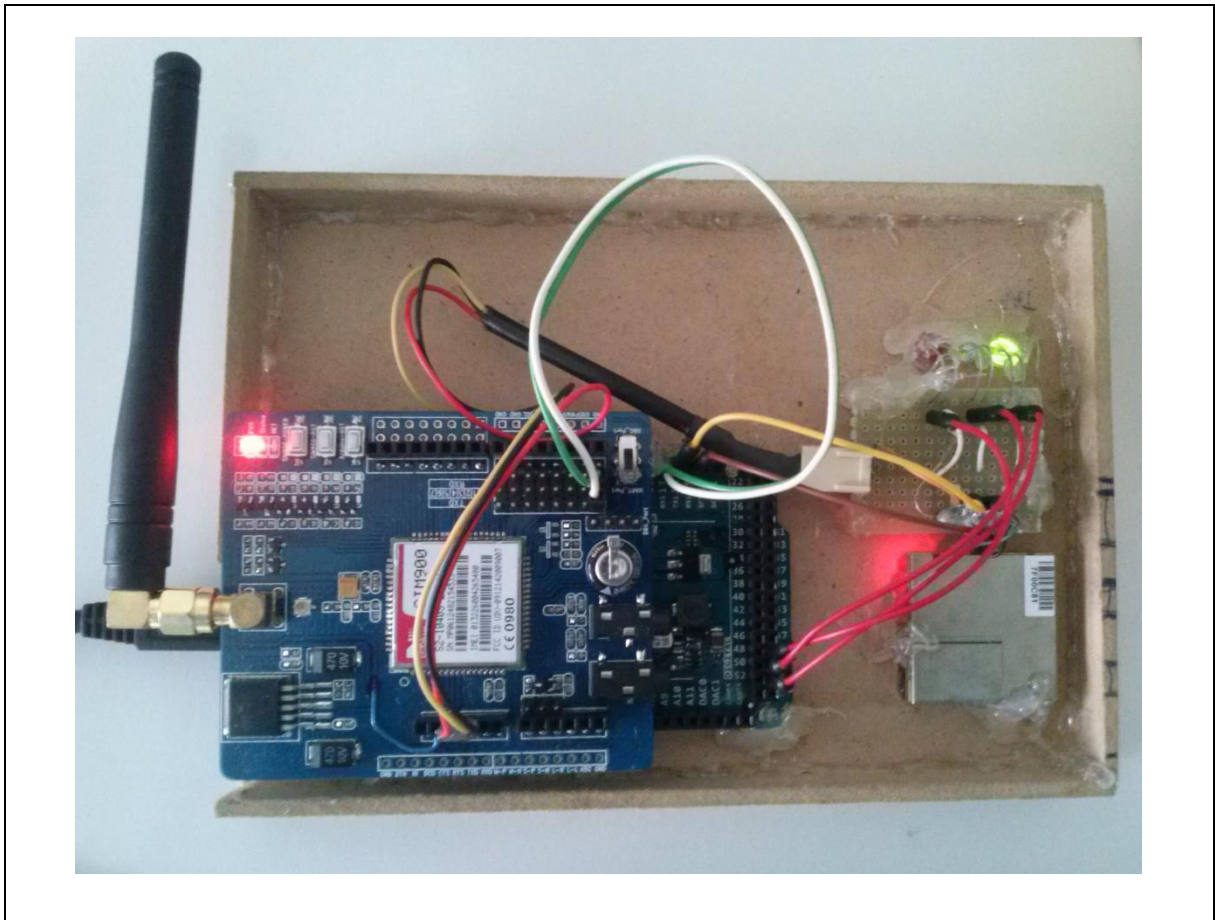


Figura 11. Modelo de rastreador (Visão Interna).

Após a finalização e total montagem do protótipo, iniciou-se o desenvolvimento e programação do sistema.

3.2 DESENVOLVIMENTO DO SOFTWARE

A partir do detalhamento e montagem do hardware e da definição do modelo de SO de tempo real escolhido (*ChibiOS*), modelou-se o Sistema Operacional realizando a adaptação do RTOS através da análise dos requisitos, modelagem e plano de teste; e após o desenvolvimento do mesmo junto com um modelo básico de aplicação de rastreamento.

3.2.1 Análise Requisitos

A análise dos requisitos constituiu-se em um fator fundamental para auxiliar na identificação do funcionamento do Sistema Operacional e seus requisitos são divididos entre funcionais e não funcionais. Os requisitos funcionais são:

1. RF01 – O SO deverá realizar o controle de multiprocessos e processos concorrentes;
2. RF02 – O SO deverá realizar a gerência do processador;
3. RF03 – O SO deverá realizar o controle de entrada e saída no hardware;
4. RF04 – O SO deverá realizar o controle e o acesso a memória;
5. RF05 – O sistema deverá fornecer funções que torne possível a utilização dos itens anteriores.

Nota-se primeiramente, que o SO deve realizar o controle de multiprocessos e processos concorrentes, ou seja, com o multiprocessamento se tem um melhor aproveitamento do uso da CPU e recursos de hardware. Para Oliveira e Andrade, (2006) um processo é definido com um “programa em execução” e estes programas são sequencias de instruções, assim o SO tem que identificar quando há processos executando e quando esses entram em fila de espera para serem executados, desta forma, realizando também a gerencia de processos concorrentes, necessária para evitar possíveis ameaças a estabilidade do sistema como postergação indefinida (*deadlock*).

O requisito funcional RF02 se refere a forma como será realizado o chaveamento de contexto, o *multithreading* (vários fluxos de execução) e o escalonamento. No caso do *ChibiOS* é possível utilizar dois métodos de escalonamento, que são escalonamento por prioridades e modelo de fatia de tempo (*Round-Robin*).

O requisito funcional RF03 se refere à interface entre os periféricos e o sistema, no caso o SO realizará a interação com os dispositivos de E/S e enviará comandos e aguardará resposta dos periféricos, assim utilizado no tratamento dos *Shield*.

A gerencia de memória está contemplada no requisito funcional RF04, que trata como a memória será utilizada pelos processos e como essa será alocada, a maneira implantada neste

projeto segue o modelo do ChibiOS com quatro formas de gerenciarmos, são elas: *Core Allocator*, *Memory Heaps*, *Memory Pools* e *Dynamic Threads* (CHIBIOS, 2013).

Por fim, o sistema deverá fornecer funções que torne possível a utilização dos itens anteriores, estes chamados de “chamadas do sistema”. São métodos que estão disponíveis no *kernel* onde realiza-se ações como criar *threads*, limpeza de memória e comunicação com periféricos.

Os requisitos não funcionais são:

1. RNF01 – O SO será realizado com linguagem de programação Assembly, C e C++;
2. RNF02 – O SO será executado através da implantação no protótipo de rastreador;
3. RNF03 – O sistema será validado com a execução do SO no protótipo de rastreador analisando seus resultados e seu comportamento.

O primeiro requisito não funcional (RNF01) se refere a linguagem de programação utilizada, segundo ChibiOS, (2013) a linguagem de desenvolvido utilizados por eles foram exatamente essas citadas. Os outros dois requisitos (RNF02 e RNF03) se referem a forma como será testado o Sistema Operacional, no caso, com a implantação do SO no rastreador é possível realizar a comunicação dele com uma base, está por sua vez pode receber pacotes de telemetria assim validando seu funcionamento.

3.2.2 Modelagem e Desenvolvimento

A modelagem e desenvolvimento do *firmware* segue a seguinte forma:

- Análise do hardware disponível;
- Adaptação do *Kernel*;
- Aplicação.

Com a modelagem do hardware e criação do protótipo já realizados, a adaptação do Kernel precisa ser melhor detalhada. O grupo ChibiOS já disponibiliza o código fonte do Sistema Operacional para ser utilizado, esse por sua vez, conforme a documentação, é

Verifica-se que o ChibiOS é modularizado para que se possa realizar a melhor adaptação possível do mesmo. Ele é dividido em aplicação, HAL (*hardware abstraction layer*)⁴, plataforma, *Kernel*, *Port*, Configurações de placa e hardware.

A aplicação contempla a lógica de funcionamento do rastreador, como ele se comportou, como ele realizou a captura e armazenamento das informações de GPS e o envio de pacotes através do *GSM Shield* foi realizado neste módulo. O código adicional desenvolvido para este projeto foi “embutido” junto com o código fonte do Sistema Operacional, já que optou-se por não utilizar um compilador e memória externa para o código e executar no SO. Esse método criou um único hexadecimal que foi executado pelo micro controlador.

Visualmente o código inicial foi igual o Quadro 5:

```
/*
 * Nome do Projeto
 * Modelo de Aplicação para rastreadores utilizando protótipo criado
 * com Arduino Due
 *
 * Copyright:
 * Lucas Selliach
 */

#include <ChibiOS_ARM.h>

#define pinLedON 53          //led1 - Sistema Operante
#define pinLedSO 51          //led2 - Status Sistema Operacional
#define pinLedTilt 49        //led3 - Erro no Sistema
```

Quadro 5. Código fonte inicial do modelo

Nota-se que o sistema operacional é importado com a chamada da biblioteca ChibiOS.h, nesse arquivo foi realizada toda a inicialização do Sistema Operacional compatível com o microcontrolador ARM-CortexM3 e também houve extensões para realizar o carregamento do HAL, do *Kernel* e do *Port*.

É possível perceber que com esse código inicial, realiza-se a validação dos requisitos funcionais, analisando que o RF01 e RF02 estão contidos no Quadro 6, onde é realizando a

⁴ Camada de abstração de hardware – A sua função é “apresentar uma parte do sistema operacional, dispositivos abstratos de hardware desprovidos de especificidades e idiosincrasias das quais o hardware real está repleto. Esses dispositivos são apresentados na forma de serviços independentes de máquina [...], os quais podem ser usados pelo restante do sistema operacional e pelos drivers” (TANENBAUM, 2010).

criação de duas *Threads*, uma para controlar o LED de status do sistema e outra para realizar o envio de informações por rede GSM.

```
chThdCreateStatic(waThread1, sizeof (waThread1), NORMALPRIO + 2,
(tfunc_t) Thread1, 0);
```

Quadro 6. Criação das *Threads* com prioridades diferenciadas.

A criação e utilização das threads só é possível por causa da utilização do *Kernel*, ele é responsável por disponibilizar as chamadas do sistema, além de realizar toda a gestão de memória, processos e entrada/saída. Segundo o Chibios, (2013), o *Kernel* do sistema também é modular e é composto por diversos sub sistemas. Ele é dividido em serviços de base (*microkernel*), Sincronizações, gerenciamento de memória, *Streams*⁵ / canais de I/O e depuração. Todos esses serão utilizados no SO padrão para o rastreadores.

O *microkernel* é composto de quatro partes:

- *System*: É a base do sistema, é aonde realiza a inicialização do SO;
- *Timers*: É a parte que fornece noção de tempo para o SO através do oscilador, também dispõem de API de tempo;
- *Scheduler*: É a parte responsável pela frequência utilizada do processador, ou seja, é serve como base para qualquer tarefa baseada no *clock*⁶ que utilize sincronismo;
- *Threads*: É a parte responsável pelo fluxo de execução.

Basicamente o serviço de *microkernel* se resume na Figura 13. Nota-se que a sincronização e as *threads* utilizam o “mecanismo de sincronização” (*scheduler*) para realizar o controle do tempo, este último por sua vez utiliza a inicialização do sistema, a camada de abstração do hardware e os tempos para seu correto trabalho.

⁵ Fluxo de mensagens.

⁶ “O *Clock* fornece a sensibilidade de tempo para o processador” (OLIVEIRA e ANDRADE, 2006).

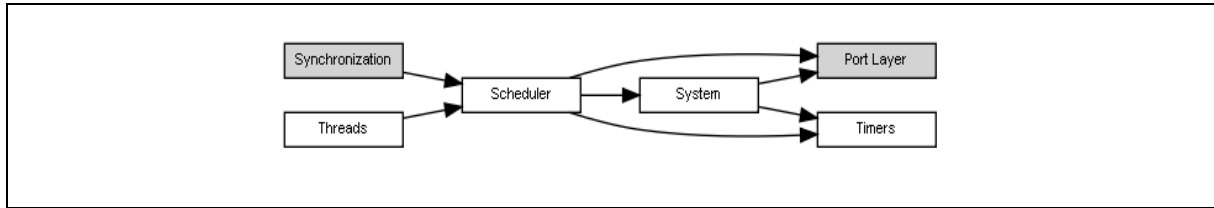


Figura 13. Diagrama *microkernel*

Fonte: *Chibios*, 2013

O serviço de sincronização possui o necessário para realizar o controle de multiprocessos e processos concorrentes. Possui funções como:

- Semáforos: Este trabalha com operações *DOWN* e *UP*;
- *Mutexes*: Estes são variáveis que trabalho com estados de impedido e desimpedido;
- Eventos: Este trabalha com ativação de um manipulador automático;
- Mensagens: Este trabalha com *send* e *receive*;
- Bandeiras: variáveis de condições que trabalham com o método de monitores.

Um exemplo de utilização é o Quadro 7 que é parte do código do apêndice B que utiliza o *Mutexes* para realizar comunicação entre as threads de leitura GPS e de envio por GSM.

```

MUTEX_DECL(dataMutex) ;

chMtxLock(&dataMutex) ;

dados.Tserial = serialSystem;
dados.TstatusSO = true;
dados.Tlatitude = latitude;
dados.Tlongitude = longitude;
dados.Taltitude = altitude;
dados.TgpsVelocidade = velocidade;
dados.Tcurso = curso;
dados.TgpsDateTime = data + " " + hora;
dados.TnumeroDeSatelites = satelites;
dados.TqualidadeDeSinalGPS = qualidadeSinal;
dados.TqualidadeDeSinalGSM = 0;
dados.Tobservacao = valorDescricao;

chMtxUnlock(&dataMutex) ;
  
```

Quadro 7. Utilização de tabela com sincronismo entre *Threads*

Neste trecho de exemplo é utilizada uma tabela de valores para realizar o tráfego de dados, assim mantendo o sincronismo entre as threads e permitindo ao Sistema Operacional funcionar sem o surgimento da postergação indefinida.

Embora nem todas as funções do *Kernel* tenham sido utilizadas em primeiro momento no projeto, é interessante possuí-las para futuros usos e necessidades. Segundo o grupo ChibiOS, (2013) todo o mecanismo de sincronização é montado em cima do *Scheduler*, Figura 14.

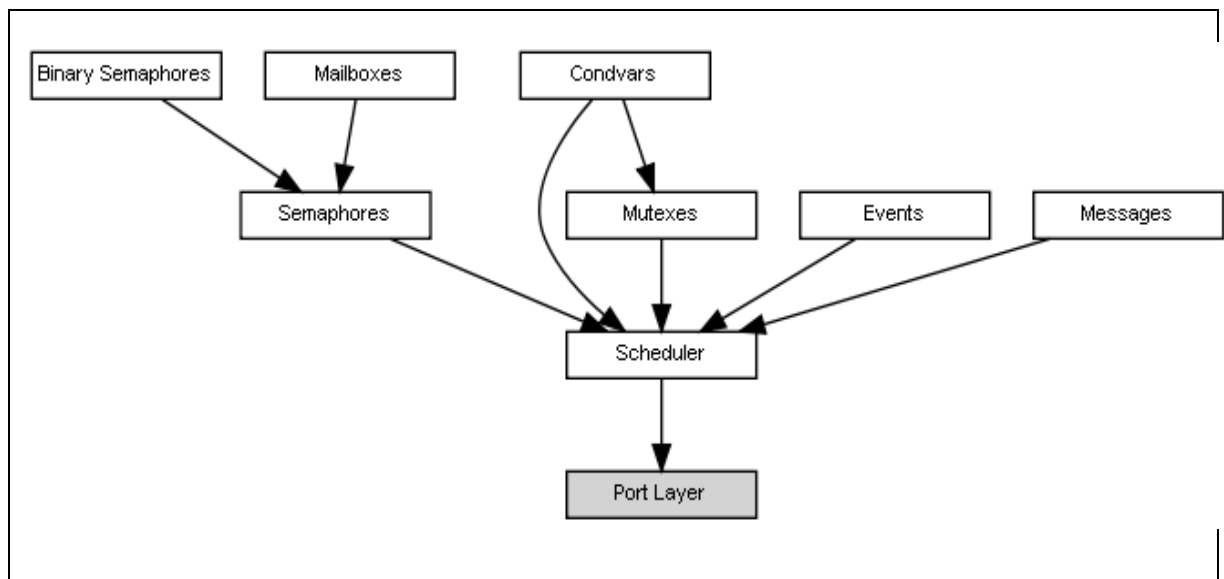


Figura 14. Diagrama sincronização

Fonte: Chibios, 2013

O RF04 está presente desde o início da inicialização do sistema operacional, porém fica evidente no Quadro 8, que mostra o trecho do código que realizar a criação da thread com alocação estática da memória no microcontrolador. Com alocação de 5001 *bytes* além da alocação da memória pelo SO, é somente ocupado 8% da memória do microcontrolador como pode ser visto na Figura 15.

```

static WORKING_AREA(waThread1, 5);

static WORKING_AREA(waThread2, 4096);
  
```

Quadro 8. Criação de *Thread*.

```

Sketch uses 43.336 bytes (8%) of program storage space. Maximum is 524.288 bytes.
Erase flash
Write 44732 bytes to flash

[          ] 0% (0/175 pages)
[=         ] 5% (10/175 pages)
[===        ] 11% (20/175 pages)
[=====   ] 17% (30/175 pages)
[=====   ] 22% (40/175 pages)
[=====   ] 28% (50/175 pages)
[=====   ] 34% (60/175 pages)
[=====   ] 40% (70/175 pages)
[=====   ] 45% (80/175 pages)
[=====   ] 51% (90/175 pages)
[=====   ] 57% (100/175 pages)
[=====   ] 62% (110/175 pages)
[=====   ] 68% (120/175 pages)
[=====   ] 74% (130/175 pages)
[=====   ] 80% (140/175 pages)
[=====   ] 85% (150/175 pages)
[=====   ] 91% (160/175 pages)
[=====   ] 97% (170/175 pages)
[=====   ] 100% (175/175 pages)
Verify 44732 bytes of flash

```

Figura 15. Uso da memória

Assim, o gerenciamento de memória realizada no *kernel*, possui as seguintes funções:

- Distribuidor de núcleo: É uma espécie de gerenciador de memória do núcleo, trabalha em conjunto com o MMU⁷ (*Memory Management Unit*);
- Gerenciador dinâmico: É o responsável pelo gerenciamento de pilha da memória, utilizado com partições variáveis, implementa a técnica de *First-Fit*⁸;
- Gerenciador fixo: É o responsável pelo gerenciamento de pilha de memória, utiliza partições fixas;

⁷ “A unidade de gerencia de memória é o componente do hardware responsável por prover mecanismos básicos que serão utilizados pelo SO para gerenciar a memória” (OLIVEIRA, CARISSIMI e TOSCANI, 2008).

⁸ Primeiro ajuste – Técnica “que utiliza a primeira lacuna que encontra com tamanha suficiente” (OLIVEIRA, CARISSIMI e TOSCANI, 2008).

A hierarquia das funções citadas anteriormente encontra-se disponível na Figura 16 e estes também foram utilizados no SO do projeto do rastreador exceto a locação de threads dinâmicas.

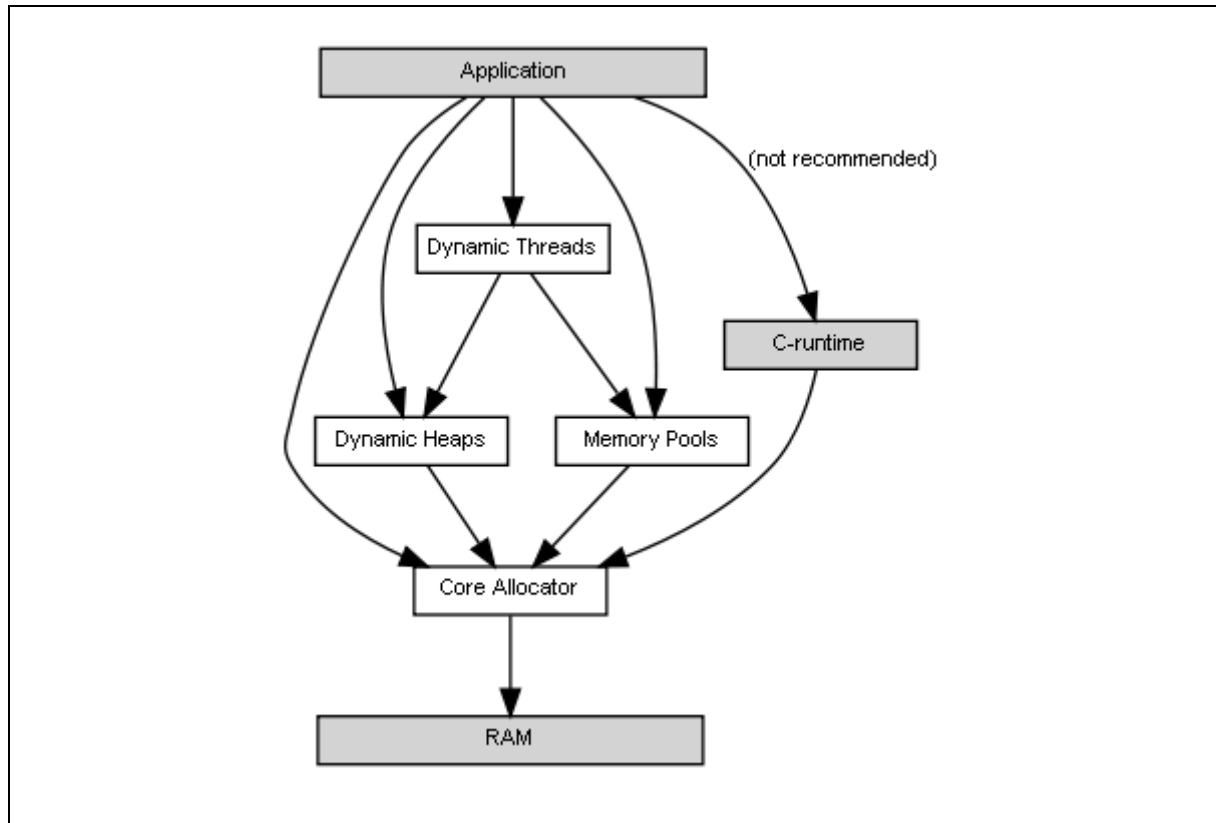


Figura 16. Diagrama de gerenciamento de memória

Fonte: Chibios, 2013

A validação do requisito RF03 é visível no Quadro 9, que realiza o controle de comunicação serial, esse trecho do código pode ser observado no apêndice B.

```

while (Serial1.available() > 0)
{
    if (gps.encode(Serial1.read()))
        GetGPSInfo();
}

```

Quadro 9. Serial com GPS.

A comunicação serial realizada entre o hardware de GPS e de GSM é de baixa interação com o SO, isso ocorreu porque não há *drivers* compatíveis sendo necessário realizar uma comunicação direta com os dispositivos. Esse método não é recomendando pelo grupo ChibiOS e, apesar de funcionar perfeitamente, torna funções de interrupção de *hardware* não funcionais.

É notável que todo o processo de utilização do SO resultou na validação do RF05, pois este está presente em todas as etapas, assim como também os requisitos não funcionais. Assim a aplicação terá uma sequência lógica representa na Figura 17.

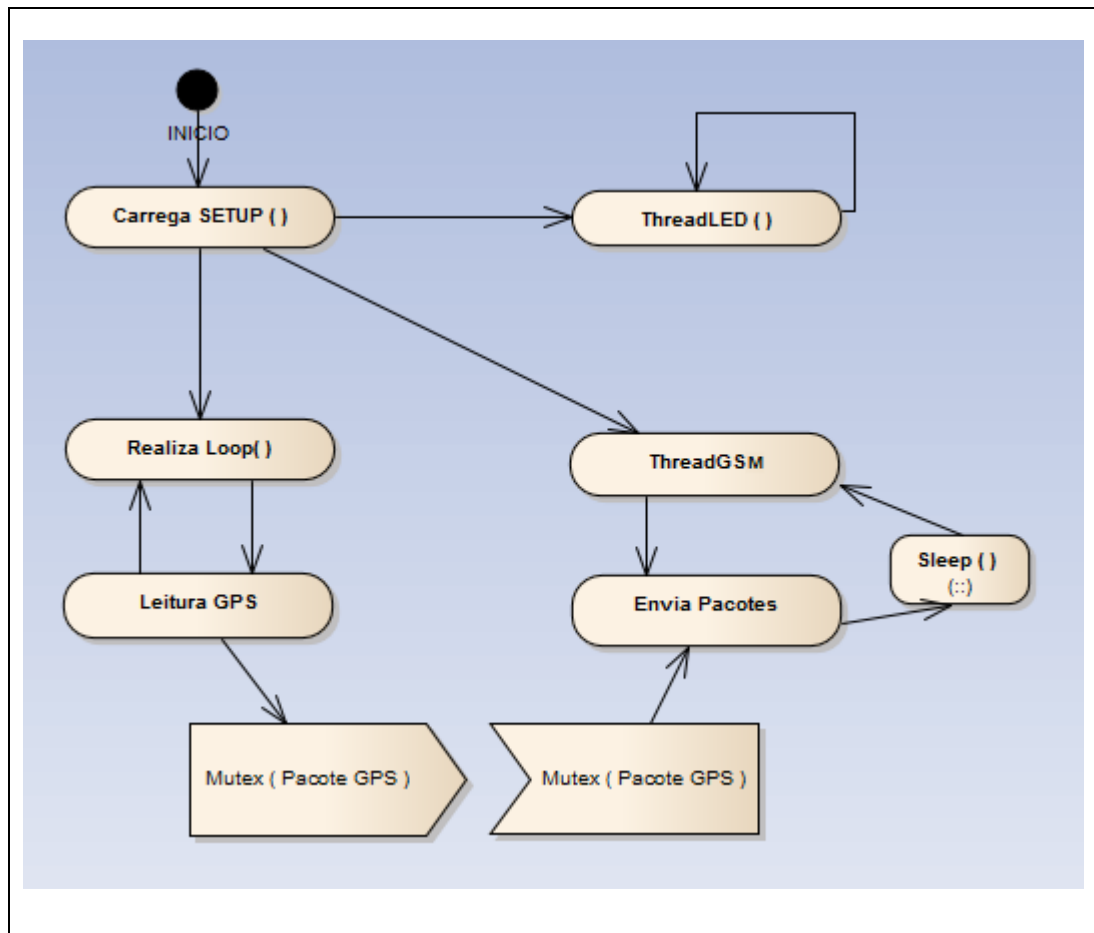


Figura 17. Fluxograma de aplicação.

O rastreador funcionou com uma base simples, onde ao ser inicializado ativou o SO, este por sua vez executou a aplicação, ela carregou as variáveis e as configuração dos pinos. O *Setup* foi responsável por realizar o carregamento do estado dos pinos de entrada e saída e os valores de alto e baixo deles, também responsável pela velocidade de comunicação dos pinos seriais e pela inicialização das *Thread* de envio GSM e estrado do SO. O *Loop* é o trabalho continuo que realiza a leitura do GPS e junto com ele será executado duas *Threads* em *Loop*, uma responsável pelo LED de informação do sistema e outra pela coleta dos dados e envio dos pacotes via rede GSM. Na *Thread* de envio de pacotes, foi estipulado um tempo para a *Thread* dormir, assim evitando o uso excessivo de banda.

No diagrama de sequência na Figura 18 é possível notar como é realizado o processo de comunicação com o GPS e o GSM *Shield* pela chamada da aplicação utilizando o Sistema Operacional.

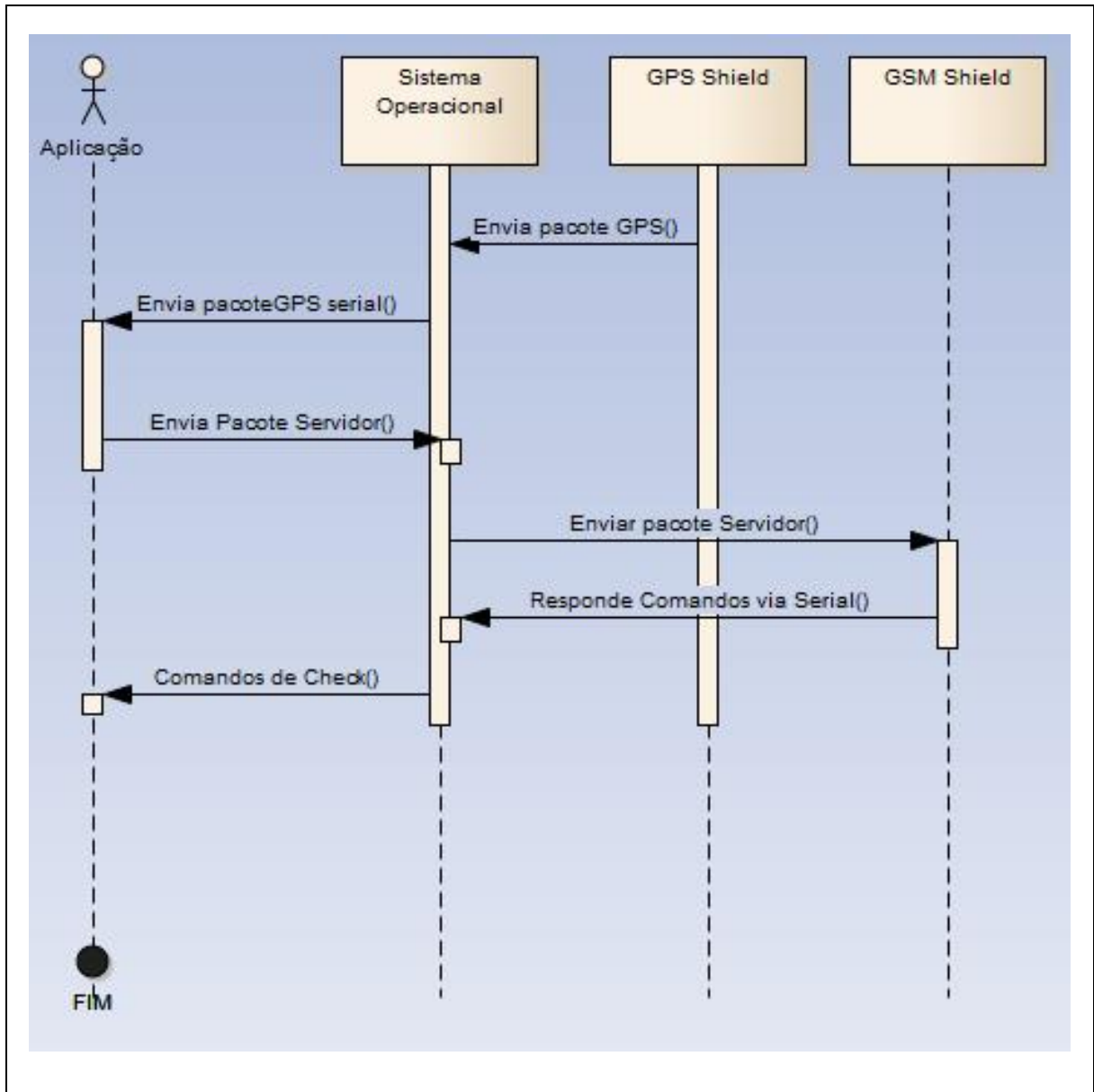


Figura 18. Diagrama de sequência do sistema.

O pacote de dados GPS são valores adquiridos pelo padrão NMEA (National Marine Electronics Association), esses valores vêm no formato apresentado na Figura 19, e é necessária a utilização de uma biblioteca disponível pela fabricante no módulo GPS, que realiza a conversão desses valores para variáveis que podem ser utilizadas e transformadas. A biblioteca utilizada é a `TinyGPS++.h`, que consegue realizar várias conversões de valores em formatos

decimais, data e hora, inteiros e *String*. Para a emissão do padrão NMEA foi realizando um configuração no módulo GPS.

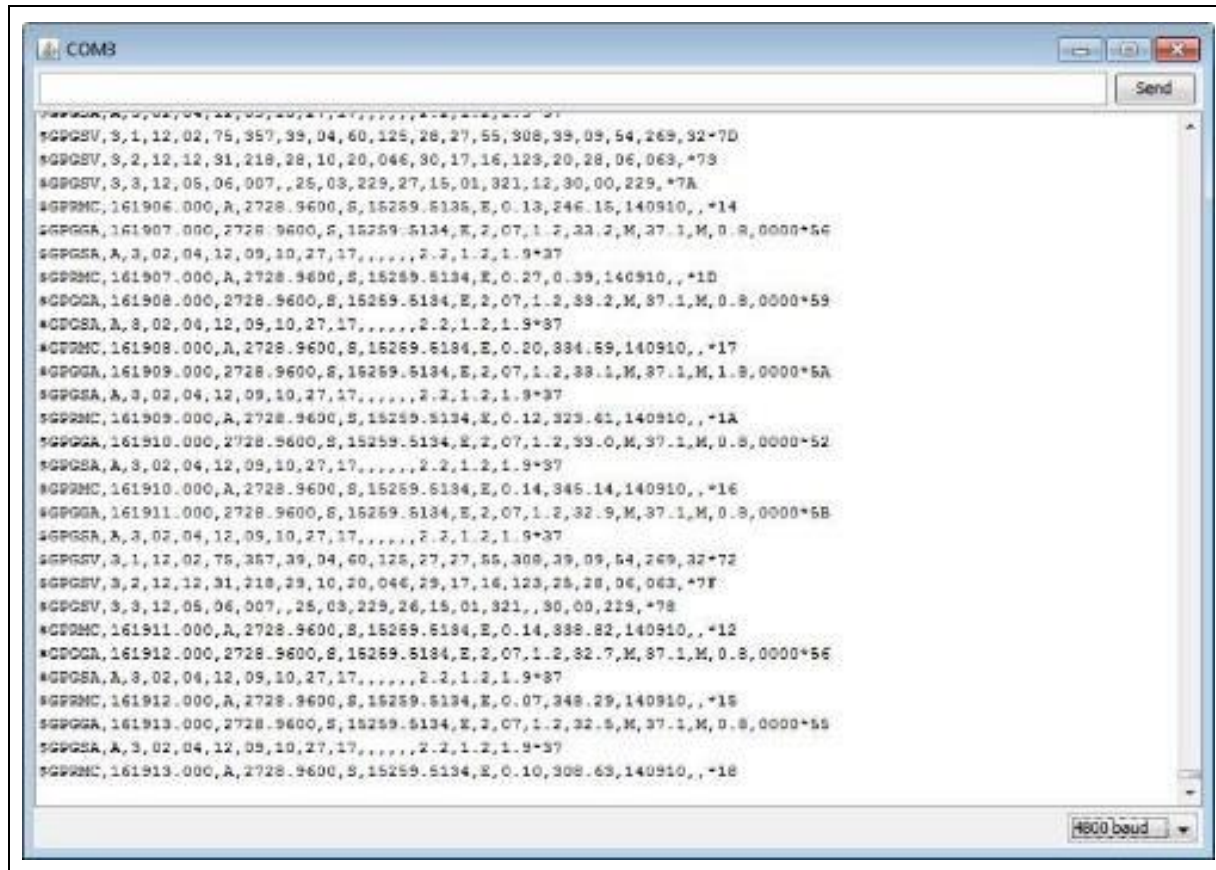


Figura 19. Padrão NMEA

Fonte: Tronixstuff, 2013

3.2.3 Plano de Testes

O plano de teste foi realizado em 3 partes, a primeira foi o teste de *Hardware*, em segundo o teste do SO e por último o teste do rastreador como um todo. No *Hardware* a fim de descobrir dificuldades que possam ocasionar seu funcionamento incorreto. O Quadro 10 apresenta quais foram os testes serem realizados e quais o resultados pretendidos e, por fim, os resultados obtidos.

Teste	Resultado Esperado	Resultado Obtido	Passou?
Com toda a montagem, colocar pinos digitais em alta.	Tensão de trabalho dos pinos em 3.3V.	Tensão medida nos pinos retornaram valores de 3.3V.	SIM
Enviar comandos diretos por porta UART para GPS shield.	Resposta a comandos.	Comandos checados com sucesso.	SIM
Enviar comandos diretos por porta UART para GSM shield.	Resposta a comandos.	Comandos checados com sucesso.	SIM

Quadro 10. Testes de *Hardware*.

Com os testes realizados em hardware e a garantia de funcionamento do mesmo, foi possível realizar a instalação do Sistema Operacional. Este teve que passar por testes, definidos no Quadro 11.

Teste	Resultado Esperado	Resultado Obtido	Passou?
Criar tarefa.	A tarefa executar suas ações definidas. Exemplo: LED pisca a cada 1 segundo.	Tarefas executada com sucesso.	SIM
Criação de rotina de <u>interrupção</u> .	Desvio para a tarefa numa interrupção.	Tarefa executada com sucesso mas irrelevante.	NÃO, a utilização da interrupção não será utilizada.

Quadro 11. Testes de Sistema Operacional.

Nos testes do Sistema Operacional, obteve-se resultados satisfatórios ao criar a tarefa, obtendo a correta execução da mesma; já no teste de interrupção notou-se que o mesmo não poderia ser utilizado pois seria necessário um drive para o GPS e um para o GSM, para que esses fizessem a requisição de interrupção.

Assim, com a validação dos testes do SO foi possível dar continuidade a realização do último teste no sistema, este baseou-se no comportamento do rastreador, ou seja, se este foi capaz de realizar as mesmas tarefas levantadas pelos rastreadores analisados no mercado. O protótipo de rastreador criado teve que realizar a leitura da posição via GPS e enviar via rede GSM/GPRS para um *webservice* capaz de realizar as leituras das informações de posicionamento. Esse teste consistiu na seguintes etapas:

- Etapa 1: Realizar a leitura de posição via GPS e analisar variáveis.
- Etapa 2: Realizar o envio de posições via rede GSM e analisar variáveis.
- Etapa 3: Realizar a verificação de posições chegadas pela rede ao webservice.

O resultado obtido para validar esse teste está dividido em duas partes, uma é um log gerado pelo rastreador e o outro seria a visualização dos dados por parte do webservice, resultando nos Quadros 12 e 13.

```
AT+SAPBR=3,1,"APN","zap.vivo.com.br"
OK
Call ReadAT+SAPBR=3,1,"USER","vivo"
OK
AT+SAPBR=3,1,"PWD","vivo"
Bem Vindo - Sistema Operacional modelo para rastreador. Versao: 1.2
OK
AT+HTTPIPINIT
OK
AT+HTTTPARA="URL","http://sellich.azurewebsites.net/WebService1.asmx/Tracker?Serial=1&StatusSO=true&LatitudeGPS=-27.10064125&LongitudeGPS=-48.61605453&AltitudeGPS=20.4000&VelocidadeGPS=0.33&DateTimeGPS=10/29/2013&NumeroDeSatelites=7&QualidadeSinalGPS=120&QualidadeSinalGSM=0&Observacao=ValoresDeGPSatualizados!"
AT+HTTTPARA="URL","http://sellich.azurewebsites.net/ServiceAT+HTTPIPACT
ION=0
OK
```

Quadro 12. LOG rastreador.


```

<?xml version="1.0" encoding="UTF-8"?>
<ArrayOfRastreadores xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://http://lucasselliach.azurewebsites.net">
  <Rastreadores>
    <ID>78</ID>
    <Serial>1</Serial>
    <StatusSO>true</StatusSO>
    <LatitudeGPS>-27.10064125</LatitudeGPS>
    <LongitudeGPS>-48.61605453</LongitudeGPS>
    <AltitudeGPS>20.4000</AltitudeGPS>
    <VelocidadeGPS>0.33</VelocidadeGPS>
    <DateTimeGPS>2013-10-29T00:00:00</DateTimeGPS>
    <DateTimeService>2013-10-29T12:41:36.067</DateTimeService>
    <NumeroDeSatelites>7</NumeroDeSatelites>
    <QualidadeSinalGPS>120</QualidadeSinalGPS>
    <QualidadeSinalGSM>0</QualidadeSinalGSM>
    <Observacao>ValoresDeGPSatualizados!</Observacao>
  </Rastreadores>
</ArrayOfRastreadores>

```

Quadro 13. LOG servidor

Com os resultado obtidos foi possível verificar que o rastreador possui o mesmo comportamento dos rastreadores disponíveis no mercado e que seu correto funcionamento resultou na aquisição e envio de informações em tempo de execução, assim tornando esse, um protótipo modelo funcional para ser utilizado para pesquisas futuras e melhorias a serem realizadas na área de GeoInformática.

4 CONCLUSÃO

Neste projeto foi proposto um modelo de SO para rastreador utilizando como base um protótipo desenvolvido com as mesmas características de um rastreador e a adaptação de um Sistema Operacional de tempo real.

O desenvolvimento do projeto acarretou pesquisas que contemplam desde as origens das características dos rastreadores até ao funcionamento e a modelagem de um protótipo. Logo de início, foram analisados os problemas que impediriam a realização da proposta inicial do projeto pelo fato de identificar a indisponibilidade por parte dos fabricantes em revelar a arquitetura do seu Hardware para o planejamento de um Sistema Operacional.

No início da fundamentação teórica, sentiu-se a necessidade de realizar o estudo sobre a história dos rastreadores, identificando a origem de funcionamento e as tecnologias utilizadas. Assim, na continuação dos estudos identificou-se Hardwares disponíveis para uso na modelagem de um protótipo, tornando possível a utilização de uma placa de prototipagem junto com periféricos de localização e comunicação. A pesquisa dos componentes de Hardware foi dificultada, pois a quantidade de produtos disponíveis tornou a análise de cada item extensa, o que levou o foco da atenção apenas para os microcontroladores disponíveis e para os protocolos de comunicações suportados. A partir destas análises realizou-se a criação do protótipo, atentando-se para dificuldades que já eram previstas, como os valores de tensão de trabalho entre a plataforma principal e os periféricos, que foram resolvidas.

No levantamento do Software, analisou-se Sistemas Operacionais com compatibilidades ao modelo de Hardware estipulado, levando como critérios de maior peso a documentação e o código fonte, este último sendo aberto para modificações.

A modelagem do projeto foi realizada com base no funcionamento e nos requisitos utilizados pela a maioria dos sistemas operacionais comerciais; foram definidos os requisitos funcionais e não funcionais do sistema, o fluxograma da aplicação e o diagrama de sequência. O sucesso da portabilidade se deu pelo fato da fabricante ter lançado durante a produção deste projeto, uma atualização do Sistema Operacional totalmente compatível com o modelo de microcontrolador utilizado, assim tornou mais simplificado a utilização de exemplos para utilizar o SO no protótipo proposto. A validação dos dados de localização gerados pelo protótipo foi feita através do envio dos pacotes de dados a um servidor, via rede GSM/GPRS.

Já a validação do protótipo foi realizada a partir de um plano de testes contemplando o atendimento, ou não, dos requisitos elencados. Como resultado, pode-se afirmar que o protótipo alcançou os objetivos esperados e tornou-se um modelo pronto para ser utilizado em futuros trabalhos de pesquisa, que possui o diferencial de ser um protótipo de hardware livre, código fonte aberto e objeto para uso em futuro estudos como o acoplamento de sensores específicos para atender demandas de diferentes áreas.

4.1 TRABALHOS FUTUROS

A realização desse projeto oportuniza aprofundar conhecimentos na área de GeoInformática, a partir do momento em que se tem disponível um modelo de rastreador com um modelo de SO base, sendo possível realizar:

- Melhorias do atual modelo, como a utilização dos protocolos de comunicações, criação de drivers dos periféricos e a melhora da aplicação;
- A criação de protocolos únicos de comunicação entre rastreadores e servidores;
- A criação de interface ricas para desenvolvimentos;
- A melhor adaptação do hardware disponível;
- A inovação em comunicação entre os rastreadores e entre rastreadores e sensores através de protocolos e padrões de comunicação;

REFERÊNCIAS

- ADRIANO, C.; HERZOG, F.; LOPES, F. **SISTEMAS DE RASTREAMENTOS EMBARCADOS**. Rio de Janeiro: Faculdade de Engenharia de Resende, 2009.
- ALBUQUERQUE, P. C. G.; SANTOS, C. C. D. GPS para iniciantes. **GeoSenso**, 2003. Disponível em: <<http://geosenso.com/arquivos/GPS%20para%20iniciantes%20-%20INPE.pdf>>.
- ALEGRE, C. P. L. N. A.; OLIVEIRA, B. S. D. **SISTEMA DE MONITORAÇÃO VEICULAR VIA GPRS**. Porto Alegre: [s.n.], 2008.
- ARAÚJO, F. Satélites Sputnik. **InfoEscola**, 29 fev. 2012. Disponível em: <<http://www.infoescola.com/astronomia/satelites-sputnik>>.
- ARDUINO, T. Arduino. **Arduino**, 08 maio 2013. Disponível em: <<http://arduino.cc/>>.
- ARM. RTOS - ARM. **www.arm.com**, 25 maio 2013. Disponível em: <<http://www.arm.com/community/software-enablement/rtos-real-time-operating-system.php>>.
- ATMEL. SAM3X - Microcontroladores baseados em ARM. **atmel**, 08 maio 2013. Disponível em: <<http://www.atmel.com/products/microcontrollers/ARM/SAM3X.aspx>>.
- CHIBIOS. CHIBIOS. **CHIBIOS**, 26 maio 2013. Disponível em: <<http://www.chibios.org/dokuwiki/doku.php>>.
- FREERTOS. FREERTOS. **freertos.org**, 26 maio 2013. Disponível em: <<http://www.freertos.org/>>.
- FUL-MAR. www.ful-mar.com.br/. **www.ful-mar.com.br/**, 20 abr. 2013. Disponível em: <<http://www.ful-mar.com.br/br/preguntas-frequentes-por-categoria.php?faqsc=4%E2%80%8E>>.
- LEÃO, L. **Cibernarrativas ou a arte de contar histórias no ciberespaço**. São Paulo: Ed. Annablume, 2004.
- MATTOS, A. N. D. **Telemetria e conceitos relacionados**. São Paulo: [s.n.], 2004.
- MAXTRACK. MAXTRACK. **MAXTRACK**, 03 maio 2013. Disponível em: <<http://www.maxtrack.com.br/site/?menu=Produtos&tipo=Equipamentos>>.
- MCNAMARA, J. **GPS for Dummies**. Canada: WileyPublishing, 2008.
- MCREBERTS, M. **Arduino básico**. São Paulo: Novatec, 2011.
- MONICO, J. F. G. **Posicionamento pelo Navstar GPS: Descrição, fundamentos e aplicações**. São Paulo: Ed.Unesp, 2000.
- OLIVEIRA, A. S. D.; ANDRADE, F. S. D. **Sistemas Embarcados**. São Paulo: Érica, 2006.
- OLIVEIRA, R. S. D.; CARISSIMI, A. D. S.; TOSCANI, S. S. **Sistema Operacionais**. Porto Alegre: Bookman: Instituto de Informática da UFRGS, 2008.

OLIVEIRA, S. D.; PITOMBO, C.; ALEGRE, L. N. **Sistema de monitoração veicular via GPRS**. Porto Alegre: PUCRS, 2008.

PRATES, G. NAVSTAR GPS: Sistema de Posicionamento Global, 2004. Disponível em: <http://w3.ualg.pt/~gprates/navstar_gps.pdf>.

QUANTA. QUANTA. **radartelematica**, 01 maio 2013. Disponível em: <<http://www.radartelematica.com.br/telematica/>>.

QUECLINK. QUECLINK. **QUECLINK**, 03 maio 2013. Disponível em: <<http://www.queclink.com/>>.

RIBEIRO, W. C. **Relações Internacionais**: Cenários para o século XXI. São Paulo: Ed.Scipione, 2004.

SKYPATROL. SKYPATROL. **SKYPATROL**, 03 maio 2013. Disponível em: <<http://www.skypatrol.com/>>.

SPARKFUN. Cellular Shield. **www.sparkfun.com**, 20 maio 2013. Disponível em: <<https://www.sparkfun.com/products/9607>>.

SPARKFUN. GPS Shield. **www.sparkfun.com**, 20 maio 2013. Disponível em: <<https://www.sparkfun.com/products/10709>>.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. São Paulo: Prantice Hall, 2010.

TRONIXSTUFF. **tronixstuff**, out. 2013. Disponível em: <<http://tronixstuff.com/2010/09/17/moving-forward-with-arduino-chapter-17-gps/>>.

UCONTROL. UCONTROL. **UCONTROL**, 17 maio 2013. Disponível em: <<http://www.ucontrol.com.ar/forosmf/explicaciones-y-consultas-tecnicas/pinout-conector-usb-tipo-b-para-circuito-impreso/>>.

APÊNDICE A. QUESTIONÁRIO SOBRE RASTREADORES

A utilização dos rastreadores é comum entre empresas de rastreamento e logísticas, essas possuem profissionais especializados na área que possam realizar manuseio e desenvolvimento em cima dos rastreadores. Esse apêndice trata da entrevistas realizada com os profissionais relatando suas experiências com os rastreadores.

A.1 ENTREVISTA 1

1. Qual a sua profissão ou ocupação?

R: Projetista Eletrônico, atualmente projetista autônomo.

2. Na sua profissão ou ocupação qual a sua área de atuação?

R: Equipamentos embarcados para telemetria e rastreamento, automação residencial e industrial, iluminação, entre outros projetos, todos incluindo microcontroladores.

3. Quantos anos de experiência possui na área de atuação?

R: Com programação já fazem 15 anos, com eletrônica são 10 e com telemetria são 6.

4. Quais as principais marcas de rastreadores utilizadas na sua área de atuação?

R: A família tetros da Quanta Tecnologia, os rastreadores da Maxtrack e os importados da Skypatrol e queclink.

5. Qual os benefícios encontrados nos rastreadores mais utilizados na sua área de atuação?

R: Cada modelo possui algumas particularidades que os tornam únicos. Essas particularidades vão desde o preço até as funcionalidades. Todos os fabricantes trabalham com uma linha de equipamentos tentando cobrir toda essa gama. Por exemplo, a quanta tem desde o tetros baby, um rastreador mais barato e mais simples, até o tetros plus, que é muito configurável, e com muitas portas de comunicação. Infelizmente o mercado nacional não consegue competir com o custo-benefício dos importados. A família de rastreadores da Queclink possui um custo-benefício assombroso com valores de rastreadores que são impraticáveis de se conseguir produzir no Brasil a um custo tão

baixo. Mas numa curva ascendente boa está a Maxtrack oferecendo produtos completos, porem simples de serem configurados, o que atende uma grande fatia do mercado. Mas no topo da configuração fica o Skypatrol, com um controle praticamente total do rastreador. Porem esse controle é muito complexo, o que exige uma equipe muito bem treinada para conseguir configura-lo. Mas ainda, o valor deles é inferior aos nacionais.

6. Qual as dificuldades encontradas nos rastreadores mais utilizados na sua área de atuação?

R: Particularmente acho que a falta de uma rede de comunicação inteligente(TIA/EIA-485) ou uma CAN-BUS faz muita falta nos rastreadores. Fora a família Tetros, todos eles dispõem somente de interfaces seriais TIA/EIA-232, o que limita muito o desenvolvimento. E mesmo a família tetros com a sua porta 485 é muito limitada no seu protocolo, principalmente no controle de trafego de dados e em erros nos documentos.

7. Dentre as dificuldades encontradas nos rastreadores mais utilizados na sua área de atuação, o que pode ser melhorado para sua utilização?

R: A adição de uma comunicação decente com periféricos, uma configuração melhor interna, porem simplificada, mirando o que o skypatrol faz, mas com algumas interfaces de programação dos rastreadores que facilitassem a vida de quem tem a tarefa de homologar e configurar esses rastreadores.

A.2 ENTREVISTA 2

1. Qual a sua profissão ou ocupação?

R: Programador que realiza programação voltada à banco de dados e homologação de rastreadores.

2. Na sua profissão ou ocupação qual a sua área de atuação?

R: Desenvolvimento e integração de dados da parte dos rastreadores.

3. Quantos anos de experiência possui na área de atuação?

R: 6 meses.

4. Quais as principais marcas de rastreadores utilizadas na sua área de atuação?

R: Quanta, Maxtrack e SkyPatrol.

5. Qual os benefícios encontrados nos rastreadores mais utilizados na sua área de atuação?

R: A alta capacidade na configuração do equipamento, podem ser configurados praticamente qualquer ação para inteligência embarcada.

6. Qual as dificuldades encontradas nos rastreadores mais utilizados na sua área de atuação?

R: O fato dos equipamentos possuírem um modo de comunicação totalmente diferente um do outro (modo de comunicação *OverTheAir*) dificulta bastante a integração destes equipamentos ao sistema.

7. Dentre as dificuldades encontradas nos rastreadores mais utilizados na sua área de atuação, o que pode ser melhorado para sua utilização?

R: A criação de um firmware que padronize a comunicação dos equipamentos, o recebimento dos comandos *OverTheAir*, de modo que facilite a integração destes equipamentos pelos diversos sistemas de telemetria existentes.

A.3 ENTREVISTA 3

8. Qual a sua profissão ou ocupação?

R: Diretor de Tecnologia da Empresa GeoSapiens Tecnologia e Informação Ltda..

9. Na sua profissão ou ocupação qual a sua área de atuação?

R: Desenvolvimento e integração de dados da parte dos rastreadores.

10. Quantos anos de experiência possui na área de atuação?

R: 6 meses.

11. Quais as principais marcas de rastreadores utilizadas na sua área de atuação?

R: Diversos rastreadores de origem nacional e internacional.

12. Qual os benefícios encontrados nos rastreadores mais utilizados na sua área de atuação?

R: Baixo custo.

13. Qual as dificuldades encontradas nos rastreadores mais utilizados na sua área de atuação?

R: A falta de padronização entre os diferentes rastreadores.

14. Dentre as dificuldades encontradas nos rastreadores mais utilizados na sua área de atuação, o que pode ser melhorado para sua utilização?

R: Gostaria de não depender do software de cada rastreador, ou seja, independente do hardware escolhido, gostaria de ter um único software que pudesse ser utilizado por todos.

APÊNDICE B. CÓDIGO FONTE DO RASTREADOR MODELO

```

/*
 * Nome do Projeto
 *   Modelo de SO para rastreadores utilizando prototipo criado com
 *   Arduino Due
 * Copyright:
 *   Lucas Selliach
 * Descrição:
 *
 * Configuração:
 *
 */
//===== Bibliotecas de Inicializacao =====
//=====

#include <ChibiOS_ARM.h>
#include <TinyGPS++.h>

//===== Declarações de variaveis =====
//=====

const String versaoSO="1.2";
const String serialSystem="1";
TinyGPSPlus gps;

MUTEX_DECL(dataMutex);

struct TabelaDados{
    String Tserial;
    boolean TstatusSO;
    float Tlatitude;
    float Tlongitude;
    float Taltitude;
    float TgpsVelocidade;
    String Tcurso;
    String TgpsDateTime;
    int TnumeroDeSatelites;
    int TqualidadeDeSinalGPS;
    int TqualidadeDeSinalGSM;
    String Tobservacao;
};

TabelaDados dados;

//===== Definições do Arduino DUE =====
//=====

#define pinLedON 53          //led1 - Sistema Operante
#define pinLedSO 51          //led2 - Status Sistema Operacional
#define pinLedTilT 49        //led3 - Erro no sistema

#define pinPowerGSMShield 9 //PIN - Alimentação do shield GSM

```

```
//=====
//===== Funções =====
//=====

void GetGPSInfo()
{
    float latitude = 0, longitude = 0, altitude = 0, velocidade = 0;
    int satellites = 0, qualidadeSinal = 0;
    String data = "", hora = "", curso = "", valorDescricao = "";
    boolean latitudeValid = false, longitudeValid = false, altitudeValid = false, velocidadeValid = false, cursoValid = false, satellitesValid = false, qualidadeSinalValid = false, dataValid = false, horaValid = false;

    if (gps.location.isValid())
    {
        latitudeValid = true;
        longitudeValid = true;
        latitude = gps.location.lat();
        longitude = gps.location.lng();
    }else{
        latitudeValid = false;
        longitudeValid = false;
    }

    if (gps.altitude.isValid())
    {
        altitudeValid = true;
        altitude = gps.altitude.meters();
    }else{
        altitudeValid = false;
    }

    if (gps.date.isValid())
    {
        dataValid = true;
        data = String(gps.date.value());
    }else{
        dataValid = false;
    }

    if (gps.time.isValid())
    {
        horaValid = true;
        hora = String(gps.time.value());
    }else{
        horaValid = false;
    }

    if (gps.satellites.isValid())
    {
        satellitesValid = true;
        satellites = gps.satellites.value();
    }else{
        satellitesValid = false;
    }

    if (gps.hdop.isValid())
    {
        qualidadeSinalValid = true;
        qualidadeSinal = gps.hdop.value();
    }
}
```

```

    }else{
        qualidadeSinalValid = false;
    }

    if (gps.speed.isValid())
    {
        velocidadeValid = true;
        velocidade = gps.speed.kmph();
    }else{
        velocidadeValid = false;
    }

    if (gps.speed.isValid())
    {
        cursoValid = true;
        curso = gps.course.isValid() ?
TinyGPSPlus::cardinal(gps.course.value()) : "***";
    }else{
        cursoValid = false;
    }

    if (latitudeValid && longitudeValid && alltitudeValid &&
velocidadeValid && cursoValid && satelitesValid && qualidadeSinalValid &&
dataValid && horaValid){
        valorDescricao = "ValoresDeGPSatualizados!";

        chMtxLock(&dataMutex);
        dados.Tserial = serialSystem;
        dados.TstatusSO = true;
        dados.Tlatitude = latitude;
        dados.Tlongitude = longitude;
        dados.Taltitude = alltitude;
        dados.TgpsVelocidade = velocidade;
        dados.Tcurso = curso;
        dados.TgpsDateTime = data + " " + hora;
        dados.TnumeroDeSatelites = satelites;
        dados.TqualidadeDeSinalGPS = qualidadeSinal;
        dados.TqualidadeDeSinalGSM = 0;
        dados.Tobservacao = valorDescricao;
        chMtxUnlock();
    }else{
        valorDescricao = "ValoreDeGPSdesatualizados.";
        chMtxLock(&dataMutex);
        dados.Tobservacao = valorDescricao;
        chMtxUnlock();
    }
}

void SIM900_PowerONOFF()
{
    short answer = 0;

    answer = sendATcommand("AT", "OK", 2000);

    if (answer == 0)
    {
        digitalWrite(pinPowerGSMShield,HIGH);
        delay(3000);
        digitalWrite(pinPowerGSMShield,LOW);
        while(answer == 0){
            answer = sendATcommand("AT", "OK", 2000);

```

```

    }
}

void SIM900_config()
{
    while (sendATcommand2("AT+CREG?", "+CREG: 0,1", "+CREG: 0,5", 2000)
== 0);

    Serial2.println("AT+SAPBR=3,1,\"APN\",\"zap.vivo.com.br\"");
    delay(2000);
    toSerial();

    Serial2.println("AT+SAPBR=3,1,\"USER\",\"vivo\"");
    delay(2000);
    toSerial();

    Serial2.println("AT+SAPBR=3,1,\"PWD\",\"vivo\"");
    delay(2000);
    toSerial();

    Serial2.println("AT+SAPBR=1,1");
    delay(2000);
    toSerial();
    delay(2000);
}

void toSerial()
{
    while(Serial2.available() != 0)
    {
        Serial.write(Serial2.read());
    }
}

short sendATcommand(char* ATcommand, char* expected_answer1, unsigned
int timeout){

    int x=0, answer=0;
    char response[100];
    unsigned long previous;
    memset(response, '\0', 100);
    delay(100);
    while( Serial2.available() > 0) Serial2.read();

    Serial2.println(ATcommand);
    x = 0;
    previous = millis();

    do{
        if(Serial2.available() != 0){
            response[x] = Serial2.read();
            x++;
            if (strstr(response, expected_answer1) != NULL)
            {
                answer = 1;
            }
        }
    }
    while((answer == 0) && ((millis() - previous) < timeout));
    return answer;
}

```

```

}

short sendATcommand2(char* ATcommand, char* expected_answer1, char*
expected_answer2, unsigned int timeout){

    short x=0, answer=0;
    char response[100];
    unsigned long previous;

    memset(response, '\0', 100);    // Initialize the string
    delay(100);

    while( Serial2.available() > 0) Serial2.read();
    Serial2.println(ATcommand);    // Send the AT command

    x = 0;
    previous = millis();

    do{
        if(Serial2.available() != 0){
            response[x] = Serial2.read();
            x++;
            if (strstr(response, expected_answer1) != NULL)
            {
                answer = 1;
            }
            if (strstr(response, expected_answer2) != NULL)
            {
                answer = 2;
            }
        }
    }while((answer == 0) && ((millis() - previous) < timeout));
    return answer;
}

//=====
//===== Threads ChibiOS =====
//=====

static WORKING_AREA(waThread1, 5);
static WORKING_AREA(waThread2, 4096);

//Thread do LED SO
void Thread1 () {
    while (1) {
        digitalWrite(pinLedSO, HIGH);
        chThdSleepMilliseconds(100);
        digitalWrite(pinLedSO, LOW);
        chThdSleepMilliseconds(900);
    }
}

//Thread Envio de pacotes por dispositivo GSM
void Thread2 () {
    while(1){

        chThdSleepMilliseconds(60000);

        float latitude = 0, longitude = 0, alltitude = 0, velocidade = 0;
        int satelites = 0, qualidadeSinalGPS = 0, qualidadeSinalGSM = 0;
    }
}

```

```

String serial = "", data = "", curso = "", valorDescricao = "",
PacoteEnvio = "";
boolean StatusSO = true;

chMtxLock(&dataMutex);
serial = dados.Tserial;
StatusSO = dados.TstatusSO;
latitude = dados.Tlatitude;
longitude = dados.Tlongitude;
alltitude = dados.Taltitude;
velocidade = dados.TgpsVelocidade;
curso = dados.Tcurso;
data = dados.TgpsDateTime;
satelites = dados.TnumeroDeSatelites;
qualidadeSinalGPS = dados.TqualidadeDeSinalGPS;
qualidadeSinalGSM = dados.TqualidadeDeSinalGSM;
valorDescricao = dados.Tobservacao;
chMtxUnlock();

String dia = data.substring(0,2);
String mes = data.substring(2,4);
String ano = data.substring(4,6);
String hora = data.substring(7,9);
String minuto = data.substring(9,11);
String segundo = data.substring(11,13);

String valorLat = String(latitude,8);
String valorLog = String(longitude,8);
String valorAlt = String(alltitude,4);

// initialize http service
Serial2.println("AT+HTTPIPINIT");
chThdSleepMilliseconds(2000);
toSerial();

Serial.println("AT+HTTTPARA=\"URL\", \"http://sellich.azurewebsites.net/We
bService1.aspx/Tracker?Serial=" + serial + "&StatusSO=" + "true" +
"&LatitudeGPS=" + valorLat + "&LongitudeGPS=" + valorLog + "&AltitudeGPS="
+ valorAlt + "&VelocidadeGPS=" + velocidade +
"&DateTimeGPS=" + mes + "/" + dia + "/20" + ano + "&NumeroDeSatelites=" +
satelites + "&QualidadeSinalGPS=" + qualidadeSinalGPS +
"&QualidadeSinalGSM=" + qualidadeSinalGSM + "&Observacao=" +
valorDescricao + "\"");

Serial2.println("AT+HTTTPARA=\"URL\", \"http://sellich.azurewebsites.net/
WebService1.aspx/Tracker?Serial=" + serial + "&StatusSO=" + "true" +
"&LatitudeGPS=" + valorLat + "&LongitudeGPS=" + valorLog + "&AltitudeGPS="
+ valorAlt + "&VelocidadeGPS=" + velocidade +
"&DateTimeGPS=" + mes + "/" + dia + "/20" + ano + "&NumeroDeSatelites=" +
satelites + "&QualidadeSinalGPS=" + qualidadeSinalGPS +
"&QualidadeSinalGSM=" + qualidadeSinalGSM + "&Observacao=" +
valorDescricao + "\"");

chThdSleepMilliseconds(2000);
toSerial();

// set http action type 0 = GET, 1 = POST, 2 = HEAD
Serial2.println("AT+HTTTPACTION=0");
chThdSleepMilliseconds(6000);
toSerial();

```

```

        // read server response
        Serial2.println("AT+HTTPREAD");
        chThdSleepMilliseconds(1000);
        toSerial();

        Serial2.println("");
        Serial2.println("AT+HTTPTERM");
        toSerial();
        chThdSleepMilliseconds(300);
        Serial2.println("");
    }
}

//=====
//===== Arduino DUE SETUP and LOOP =====
//=====

void setup() {
    pinMode(pinLedON, OUTPUT);
    pinMode(pinLedSO, OUTPUT);
    pinMode(pinLedTilt, OUTPUT);
    pinMode(pinPowerGSMShiield, OUTPUT);

    digitalWrite(pinLedON, HIGH);
    digitalWrite(pinLedSO, LOW);
    digitalWrite(pinLedTilt, LOW);

    Serial.begin(115200);
    Serial1.begin(4800);
    Serial2.begin(9600);

    Serial.println("");
    Serial.println("Inicializando Sistema Operacional...");
    SIM900_PowerONOFF();
    SIM900_config();
    Serial.println("Bem Vindo - Sistema Operacional modelo para
rastreador. Versao: " + versaoSO);
    Serial.println("");

    chBegin(mainThread);
}

void mainThread () {
    chThdCreateStatic(waThread1, sizeof (waThread1), NORMALPRIO,
(tfunc_t) Thread1, 0);
    chThdCreateStatic(waThread2, sizeof (waThread2), NORMALPRIO + 1,
(tfunc_t) Thread2, 0);

    while (true){
        while (Serial1.available() > 0)
        {
            if (gps.encode(Serial1.read()))
                GetGPSInfo();
        }
    }
}

void loop () {
}

```


ANEXO A. CÓDIGO FONTE DO CHIBIOS.C E CHIBIOS.H

```

/**
 * \file
 * \brief ChibiOS for Due and Teensy 3.0
 */
#include <unistd.h>
/** should use uinstd.h to define sbrk but Due causes a conflict
 * \param[in] incr Must call with zero here
 * \return start of main stack
 */
char* sbrk(int incr);
#include <Arduino.h>
#include <ChibiOS_ARM.h>
/** define loop */
extern void loop();
//-----
/** calibration factor for delayMS */
#define CAL_FACTOR (F_CPU/7000)
/** delay between led error flashes
 * \param[in] millis milliseconds to delay
 */
static void delayMS(uint32_t millis) {
    uint32_t i;
    uint32_t iterations = millis * CAL_FACTOR;
    for(i = 0; i < iterations; ++i) {
        asm volatile("nop\n\t");
    }
}
//-----
/** Blink error pattern
 *
 * \param[in] n number of short pulses
 */
static void errorBlink(int n) {
    noInterrupts();
    pinMode(13, OUTPUT);
    for (;;) {
        int i;
        for (i = 0; i < n; i++) {
            digitalWrite(13, 1);
            delayMS(300);
            digitalWrite(13, 0);
            delayMS(300);
        }
        delayMS(2000);
    }
}
//-----
// catch Teensy and Due exceptions
/** Hard fault - blink one short flash every two seconds */
void hard_fault_isr() {errorBlink(1);}
/** Hard fault - blink one short flash every two seconds */
void HardFault_Handler() {errorBlink(1);}

```

```

/** Bus fault - blink two short flashes every two seconds */
void bus_fault_isr() {errorBlink(2);}
/** Bus fault - blink two short flashes every two seconds */
void BusFault_Handler() {errorBlink(2);}

/** Usage fault - blink three short flashes every two seconds */
void usage_fault_isr() {errorBlink(3);}
/** Usage fault - blink three short flashes every two seconds */
void UsageFault_Handler() {errorBlink(3);}

/** Dummy init - already done in startup */
void hal_llc_init(void) {
}
/** Dummy init - already done in startup */
void boardInit(void) {
}
//-----
/** fill heap on Teensy */
void startup_early_hook() {
    uint32_t* end = &_estack - 100;
    uint32_t* p = &ebss;
    while (p < end) *p++ = MEMORY_FILL_PATTERN;
}
//-----
/** continuation of main thread */
static void (*mainFcn)() = 0;
/**
 * Start ChibiOS/RT - does not return
 * \param[in] mainThread Function to be called before repeated calls
 *                to loop().
 */
void chBegin(void (*mainThread)()) {
    extern sysTickEnabled;
    sysTickEnabled = 1;
    mainFcn = mainThread;
    uint32_t msp, psp, reg;
    // disable interrupts
    asm volatile ("cpsid i");
    // set Main Stack pointer to top of SRAM
    msp = (uint32_t)(&_estack);
    asm volatile ("msr      MSP, %0" : : "r" (msp));
    // Process Stack initialization for main thread
    // allow HANDLER_STACK_SIZE entries for handler stack */
    psp = (uint32_t)(&_estack - HANDLER_STACK_SIZE);
    asm volatile ("msr      PSP, %0" : : "r" (psp));
    reg = 2;
    asm volatile ("msr      CONTROL, %0" : : "r" (reg));
    asm volatile ("isb");
    {
        uint32_t* p = (uint32_t*)sbrk(0);
        // fill memory - loop works since compiler doesn't use stack
        while (p < &_estack) *p++ = MEMORY_FILL_PATTERN;
    }
    halInit();
    chSysInit();
    interrupts();
    if (mainFcn) mainFcn();
    while(1) {loop();}
}

```

```

//-----
/**
 * Determine unused bytes in the handler stack area
 *
 * \return number of unused bytes
 */
size_t chUnusedHandlerStack() {
    uint32_t* used = &_estack - HANDLER_STACK_SIZE;
    while (used < &_estack) {
        if (*used != MEMORY_FILL_PATTERN) break;
        used++;
    }
    return sizeof(uint32_t)*(used - &_estack + HANDLER_STACK_SIZE);
}
//-----
/**
 * Determine unused bytes in the heap/loop stack area
 *
 * \return number of unused bytes
 */
size_t chUnusedHeapMain() {
    uint32_t* bgn;
    uint32_t* end;
    uint32_t* brk = (uint32_t*) sbrk(0);
    if (*brk == MEMORY_FILL_PATTERN) {
        bgn = brk - 1;
        while (bgn >= &_ebss) {
            if (*bgn != MEMORY_FILL_PATTERN) break;
            bgn--;
        }
        end = brk + 1;
        while (end < (&_estack - HANDLER_STACK_SIZE)) {
            if (*end != MEMORY_FILL_PATTERN) break;
            end++;
        }
        return sizeof(uint32_t)*(end - bgn - 1);
    }
    return 0;
}
//-----
/**
 * Determine unused stack for a thread
 *
 * \param[in] wsp pointer to working space for thread
 * \param[in] size working space size
 *
 * \return number of unused stack locations
 */
size_t chUnusedStack(void *wsp, size_t size) {
    size_t n = 0;
#ifdef CH_DBG_FILL_THREADS
    uint8_t *startp = (uint8_t *)wsp + sizeof(Thread);
    uint8_t *endp = (uint8_t *)wsp + size;
    while (startp < endp)
        if (*startp++ == CH_STACK_FILL_VALUE) ++n;
#endif
    return n;
}

```

```

/**
 * \file
 * \brief ChibiOS for Due and Teensy 3.0
 */

#ifndef ChibiOS_ARM_h
#define ChibiOS_ARM_h
#include <utility/ch.h>
#include <utility/hal.h>
#ifdef __cplusplus
extern "C"{
#endif // __cplusplus
//-----
/** ChibiOS_ARM version YYYYMMDD */
#define CHIBIOS_ARM_VERSION 20130710
//-----
#ifndef __arm__
#error ARM Teensy 3.0 or Due required
#endif // __arm__

/** number of 32-bit entries in main stack */
#define HANDLER_STACK_SIZE 100
/** fill pattern for heap, process stack, and main stack */
#define MEMORY_FILL_PATTERN 0X55555555UL
/** end of bss section */
extern unsigned long _ebss;
/** end of heap/stack area */
extern unsigned long _estack;
void chBegin(void (*mainThread)());
/** \return size of Handler stack in bytes */
inline size_t chHandlerStackSize() {return 4*HANDLER_STACK_SIZE;}
/** \return size of heap/loop stack in bytes */
inline size_t chHeapMainSize() {return 4*(&_estack - &_ebss -
HANDLER_STACK_SIZE);}
size_t chUnusedHandlerStack();
size_t chUnusedHeapMain();
size_t chUnusedStack(void *wsp, size_t size);

#ifdef __cplusplus
} // extern "C"
#endif // __cplusplus
#endif // ChibiOS_ARM_h

```