INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

XtranX Passenger - Navegação inercial

Vítor Manuel Lopes Nico Borrego - Nº 36017

(Licenciado em Engenharia Electrotécnica e de Computadores)

TRABALHO DE PROJECTO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

(Relatório final)

Presidente do Júri:

Mestre Vítor Jesus Sousa de Almeida

Vogal-Arguente:

Doutor José Augusto Afonso

Vogal-Orientador:

Doutor João Carlos Amaro Ferreira

Outubro de 2015

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

XtranX Passenger - Navegação inercial

Vítor Manuel Lopes Nico Borrego - Nº 36017

(Licenciado em Engenharia Electrotécnica e de Computadores)

TRABALHO DE PROJECTO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

(Relatório final)

Presidente do Júri:

Mestre Vítor Jesus Sousa de Almeida

Vogal-Arguente:

Doutor José Augusto Afonso

Vogal-Orientador:

Doutor João Carlos Amaro Ferreira

Outubro de 2015

Agradecimentos

O aluno que desenvolveu o trabalho de projeto gostaria de agradecer ao orientador, o Doutor João Ferreira, por toda disponibilidade e apoio prestado de forma a se concluir com sucesso o projeto.

Gostaria também de agradecer ao Mário Isidoro e Gabriel Saragoça, colaboradores na empresa Tecmic, por todo o apoio prestado de forma a se compreender e a se implementar a solução desenvolvida.

Resumo

Criar um estimador de posições geo-referenciadas, para um equipamento da empresa Tecmic. Quando não existirem posições geo-referenciadas válidas obtidas a partir de GPS, o estimador deve ser capaz de indicar uma posição aproximada com base na última posição conhecida obtida via GPS, e com os dados obtidos a partir de sensores existentes no equipamento, que são um magnetómetro e odómetro. As estimativas devem ser disponibilizadas num ficheiro que reside no próprio equipamento da Tecmic.

Palavras-chave: Tecmic, GPS, sensores, sistema embebido, geo-referenciação, bússola electrónica, odómetro, magnetómetro

Abstract

Create an estimator of geo-referenced positions, to an equipment created by Tecmic company. When it's not possible to get geo-referenced positions from GPS, the estimator must be able to supply an approximate position based on the last known position obtained by GPS, and with the data supplied by a set of sensors located inside the equipment, which are an odometer and a magnetometer. The estimates should be available in a file inside the Tecmic equipment.

Keywords: Tecmic, GPS, sensors, embedded system, geo-reference, e-compass, odometer, magnetometer

Índice de conteúdos

Lista de Tabelas

Lista de Figuras

Lista de excertos de código

Lista de Acrónimos

ADC - *Analog to Digital Converter*
AJAX - *Asynchronous JavaScript And XML*
API - *Application Programming Interface*
ASIC - *Application Specific Integrated Circuits*
BPS - *Bauds per second*
DR - *Dead Reckoning*
GNU - *GNU's Not Unix*
GPS - *Global Positioning System*
GLIBC - *GNU C Library*
GSL - *GNU Scientific Library*
HDOP - *Horizontal Dilution of Precision*
I2C - *Inter-integrated circuit*
IP - *Internet Protocol*
IIO - *Industrial Input Output*
ISEL - Instituto Superior de Engenharia de Lisboa
JSON - *JavaScript Object Notation*
NMEA - *National Marine Electronics Association*
RISC - *Reduced instruction set computing*
SPI - *Serial Peripheral Interface*
SSH - *Secure Shell*
USART - *Universal Synchronous Asynchronous Receiver Transmitter*
UTC - *Coordinated Universal Time*
USB - *Universal Serial Bus*
XML - *eXtended Markup Language*

1 Introdução

O presente capítulo apresenta a motivação para o trabalho de projeto, objetivos, metodologia e ambiente de desenvolvimento, riscos e dificuldades, e no final do capítulo é apresentada a organização do relatório.

1.1 Motivação

A Tecmic [1], uma empresa portuguesa fundada em 1988, tem vários computadores de bordo instalados em veículos de empresas clientes, de transportes de passageiros. Os computadores de bordo encontram-se lá instalados para permitir que a Tecmic auxilie as empresas a gerir o seu negócio de transporte de passageiros. Cada um dos computadores de bordo adquire informação sobre o veículo onde está instalado, utilizando dados disponibilizados por um conjunto de sensores, que adquirem informação sobre o veículo. Um desses sensores tem como objetivo adquirir a posição geo-referenciada do veículo, a partir de sinal GPS (*Global Positioning System*) [2].

Atualmente, caso os computadores de bordo percam sinal GPS, quando entram por exemplo num túnel, não têm a capacidade de indicar a sua posição geo-referenciada aproximada, com o auxílio de informação proveniente de outros sensores. Para ajudar a solucionar esse problema, a Tecmic contactou o ISEL (Instituto Superior de Engenharia de Lisboa), de forma a se criar uma solução para o problema indicado, no âmbito de um trabalho de projeto de Mestrado.

O trabalho de projecto tem com objetivo implementar uma funcionalidade de navegação inercial, para computadores de bordo instalados em veículos, controlados pelo sistema *XtranX Passenger* que foi criado pela empresa Tecmic.

Após a implementação da funcionalidade, é esperado que o computador de bordo seja capaz de obter pontos de geo-referenciação somente com base num último ponto válido obtido via GPS, e dados obtidos a partir de sensores (odómetro, acelerómetro e magnetómetro).

É esperado pela empresa Tecmic, que a funcionalidade a ser desenvolvida fique operacional, de forma a que a mesma seja integrada nos vários computadores de bordo, instalados em clientes da Tecmic. Assim o presente trabalho encontra-se ligado a um problema real de uma empresa com possibilidade de poder a vir a ser integrado no sistema comercial da empresa.

1.1.1 Tecmic

A Tecmic é uma empresa multinacional portuguesa, criada em 1988. Ela têm a capacidade de desenhar e construir *hardware* e *software* de forma independente, de forma a criar serviços que geram valor acrescentado, a clientes que operam nas áreas de atuação da Tecmic [3].

A Tecmic atua em várias áreas, que se encontram indicadas na Tabela 1.1.

Área de atuação
Sistemas de gestão da logística e transporte de mercadorias
Gestão de frotas terrestres e marítimas
Sistemas de gestão de transportes públicos
Sistemas de comando e controlo de Forças de Segurança e Emergência
Sistemas de gestão de redes de assistência
Gestão remota de equipamentos, quiosques e *vending machines*
Gestão integrada de resíduos
Sistemas de gestão de acessos e assiduidade
ASIC´s (*Application Specific Integrated Circuits*) [4] - circuitos integrados de aplicação específica

Tabela 1.1: Áreas de atuação da Tecmic

1.1.2 *XtranX Passenger*

O *XtranX Passenger* é um sistema desenvolvido pela Tecmic de ajuda à operação e informação a passageiros, direcionado para empresas de transportes de passageiros. É um sistema que se enquadra na área de atuação de sistemas de gestão de transportes públicos, mencionado anteriormente [5].

O sistema é composto por diversos módulos, os quais se encontram indicados na Tabela 1.2.

Módulo	Objetivo
InfoPublic	Fornecer informação precisa e em tempo real aos passageiros.
Eco-driver	Controle da eficiência energética da condução com vista à redução do consumo de combustível e aumento do conforto e segurança dos passageiros.
Counter	Contagem de passageiros.
BusDVR	Obter imagens de alta qualidade em movimento relativo.
InfoDesigner	Geração dinâmica de informação ao público para empresas de transporte público de passageiros.
Infotainer	Gerar informação dinâmica a bordo relativa ao serviço, fornecendo entretenimento e publicidade baseada na localização.

Tabela 1.2: Módulos do sistema *XtranX Passenger*

Para que cada um dos módulos indicados anteriormente seja capaz de atingir os seus objetivos, um computador de bordo encontra-se instalado nos veículos de clientes.

1.1.3 Computador de bordo

Um computador de bordo, ilustrado na Figura 1.1, encontra-se instalado em cada veículo. Ele tem a responsabilidade de obter dados do veículo, e controlar outros dispositivos instalados no veículo. Os dados obtidos são utilizados pelos módulos do sistema *XtranX Passenger*.

Figura 1.1: Computador de bordo, extraído de [6]

O computador de bordo adquire informação diversa sobre o veículo, como por exemplo:

- última posição conhecida;
- número de metros percorridos pelo veículo;
- consumo de combustível efetuado pelo veículo.

Para obter as informações indicadas anteriormente, e mais outras informações, o computador de bordo utiliza um conjunto de sensores para esse fim. Os dados dos sensores são adquiridos periodicamente por vários módulos de *kernel* Linux [7].

1.2 Objetivos

O trabalho de projecto tem com objetivo principal a implementação de uma funcionalidade de navegação inercial, para computadores de bordo da Tecmic instalados em veículos de clientes da mesma. O sistema *XtranX Passenger* controla os computadores de bordo.

A funcionalidade deve permitir que o computador de bordo seja capaz de obter pontos de georeferenciação, com base no último ponto válido obtido via GPS, e dados obtidos a partir de sensores existentes dentro do computador de bordo.

A Tecmic espera que a funcionalidade a ser desenvolvida fique operacional, de forma a ela ser integrada nos vários computadores de bordo, instalados nos seus clientes.

1.3 Metodologia e ambiente de desenvolvimento

Utilizou-se uma metodologia iterativa e incremental, de forma a se criarem componentes capazes de atingir os objetivos do projeto.

A Tecmic forneceu uma máquina virtual e um computador embarcado, de forma a se ter um ambiente de desenvolvimento adequado para o desenvolvimento dos componentes necessários para se solucionar o problema. No projeto, dado que foi necessário desenvolver um módulo *kernel* Linux, foi seguido o procedimento indicado em [8], para a versão de *kernel* utilizada que é a versão 3.13.5 [9]. O módulo foi desenvolvido em linguagem C, e foi utilizado um conjunto de ferramentas GNU (*GNU's Not Unix*) para esse efeito.

Para além do módulo *kernel*, existiu também a necessidade de desenvolver componentes adicionais capazes de comunicar com o módulo *kernel*, de forma a se configurar e extrair dados dos sensores com os quais o módulo comunica. Esses componentes foram também desenvolvidos utilizando o conjunto de ferramentas GNU.

Os componentes desenvolvidos devem ser capazes de ser executados no sistema operativo do computador embarcado. Para testar foi feito o *cross-build* do módulo *kernel* e dos componentes adicionais. De seguida eles foram instalados, configurados e executados no computador embarcado.

1.4 Riscos e dificuldades

No decorrer do trabalho de projecto, existiu o risco de não se conseguir obter pontos geo-referenciados válidos, a partir da navegação inercial, quando comparados com pontos geo-referenciados obtidos via GPS.

As principais dificuldades previstas foram as seguintes:

- implementar o módulo *kernel* seguindo as metodologias recomendadas;
- integrar os dados obtidos pelo módulo com o restante *software* já existente no computador de bordo;
- obter pontos de geo-referenciação válidos, quando comparados com pontos obtidos por GPS.

1.5 Organização do relatório

No capítulo atual é descrita a motivação, objetivos, metodologia e ambiente de desenvolvimento, riscos e dificuldades, e a organização do relatório. No segundo capítulo é descrito o que é um sistema de navegação, descreve-se o sistema *XtranX Passenger* da Tecmic, os componentes existentes no sistema *XtranX Passenger*, e as tecnologias já utilizadas no sistema *XtranX Passenger*, e as tecnologias adicionais necessárias para criar o sistema de navegação inercial. No terceiro capítulo são descritos os componentes criados para o sistema de navegação inercial. No quarto capítulo indicam-se os testes realizados sobre o sistema de navegação inercial. No quinto capítulo apresentam-se as conclusões. No último capítulo são indicadas as referências utilizadas para a criação do presente relatório e do trabalho de projeto.

2 Estado da arte

O presente capítulo descreve o que é um sistema de navegação, descreve o sistema *XtranX Passenger*, criado pela Tecmic, e os componentes que fazem parte do sistema *XtranX Passenger*. Também descreve as tecnologias utilizadas pelo *XtranX Passenger*, e as tecnologias utilizadas para o desenvolvimento do sistema de navegação inercial.

2.1 Sistema de navegação

O principal objetivo de um sistema de navegação consiste em determinar a posição atual de um determinado objeto, utilizando vários tipos de instrumentos ou dispositivos para atingir esse fim [10]. Exemplos de dispositivos são:

- bússola
- sextante
- cronómetro
- GPS
- mapas
- céu
- pontos de referência geográficos

Existem vários tipos de navegação, a navegação electrónica, a navegação visual, a navegação astronómica e a navegação estimada. Na navegação electrónica são utilizados equipamentos electrónicos como receptores GPS ou sistemas de navegação inerciais normalmente utilizados em aviões, navios, e submarinos [11]. Na navegação visual são utilizadas referências visuais facilmente identificáveis no terreno, e que sejam fáceis de seguir [12]. A navegação astronómica consiste em identificar a posição com base nas posições dos corpos celestes, como por exemplo o Sol, a Lua e as estrelas, que variam temporalmente [13]. Um exemplo bem conhecido é o da Estrela Polar [14]. A navegação estimada permite calcular a posição atual com base numa posição anterior. O cálculo pode ser feito com base na direção, velocidade e diferença de tempo, mas pode também ser calculada com a direção e a distância percorrida [15] [16]. Em inglês chama-se à navegação estimada de DR (*Dead Reckoning*).

A navegação estimada ou DR está sujeita a erros que são acumulados ao longo do tempo devido aos erros fornecidos pelos instrumentos utilizados para determinar a direção, velocidade, diferença de tempo e distância. Outra razão pela qual o erro acumula tem a ver com a velocidade com a qual as amostras são obtidas dos instrumentos utilizados. Se a velocidade de aquisição das amostras for

a adequada não é possível ter toda a informação necessária para criar a estimativa com precisão. A falta de precisão cria erros que acumulados ao longo do tempo geram grandes erros. Devido a isso para o DR é necessário periodicamente corrigir o erro, pois se passar muito tempo sem correção as estimativas podem ficar inutilizáveis. O DR pode ser utilizado para alturas em que receptores GPS tenham problemas em indicar a posição.

Um sistema de navegação inercial utiliza sensores e DR para determinar continuamente a posição de um objeto em movimento. Os sensores tipicamente utilizados são acelerómetros e giroscópios. O giroscópio mede a velocidade angular e consegue indicar a direção, e o acelerómetro mede a aceleração. Ao se realizar integração sobre os valores da aceleração tendo a velocidade inicial é possível se obter a velocidade atual. E realizando integração de novo mas sobre a velocidade é possível se obter a distância percorrida. Um computador é utilizado para recolher os dados dos sensores e indicar a posição, velocidade e direção. Hoje em dia grande parte dos sistemas de navegação inercial são do tipo navegação electrónica. Os sistemas de navegação inercial também podem utilizar outros tipos de sensores para além de acelerómetros e giroscópios, desde que com base na informação por eles fornecida se consiga indicar a posição, velocidade e direção [17] [18].

2.2 Arquitetura do sistema *XtranX Passenger*

O sistema *XtranX Passenger* encontra-se ilustrado na Figura 2.1, onde com borda azul se encontram os componentes desenvolvidos e já utilizados pela Tecmic, e com borda encarnada encontram-se os locais onde os componentes criados para o sistema de navegação inercial são executados.

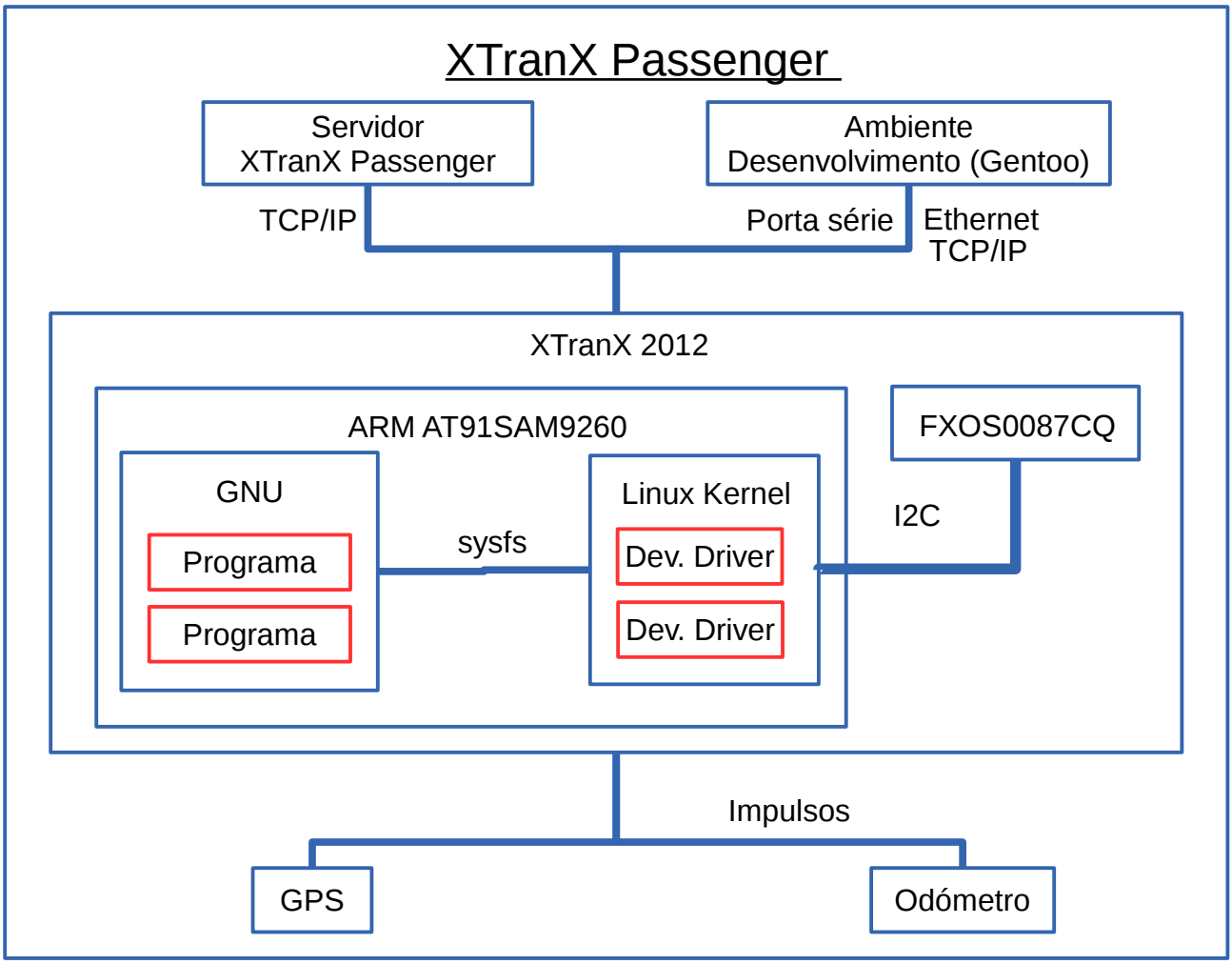

Figura 2.1: Arquitetura do sistema *XtranX Passenger*

2.3 Servidor *XtranX Passenger*

O servidor *XtranX Passenger*, desenvolvido pela Tecmic, tem como objetivo controlar os computadores embarcados, e receber toda a informação disponibilizada por eles.

2.4 Ambiente de desenvolvimento

O ambiente de desenvolvimento fornecido pela Tecmic está orientado para o sistema alvo dos componentes a desenvolver, neste caso o computador embarcado. O ambiente de desenvolvimento é composto por uma máquina virtual Gentoo Linux.

A máquina virtual contém instalado o sistema operativo Gentoo Linux com ferramentas de desenvolvimento GNU e o *kernel* Linux, que permitem realizar a compilação e ligação de componentes

para o computador embarcado.

2.5 *XtranX 2012*

O computador embarcado *XtranX 2012* é um sistema embebido desenvolvido pela Tecmic. Nele encontram-se um microcontrolador ARM *AT91SAM9260* [19], vários sensores especializados para as tarefas do computador de bordo, e o dispositivo *FXOS8700CQ* [20] que internamente contêm os sensores acelerómetro e magnetómetro. O computador de bordo utiliza uma versão *Ad-Hoc* de Linux com o *kernel* versão 3.13.5.

Para se interagir com o computador de bordo, a partir de uma *shell bash*, deve-se efetuar uma ligação por porta série, que comunica a 115200 BPS (*Bauds per second*) e configurar a porta para comunicar com 8N1 (oito bits, sem bit de paridade e um bit de paragem). Ele também tem disponível uma porta de rede *Ethernet*, que após ser executado um comando interno para se obter um endereço IP (*Internet Protocol*) e ativar um serviço de SSH, torna-se possível ligar ao computador de bordo com o auxílio de um *switch* e cabos de rede apropriados.

A partir do momento em que o serviço de SSH se encontre ativo, pode-se transferir para o computador de bordo qualquer componente que se deseje executar no contexto da versão *Ad-Hoc* de Linux lá existente.

2.5.1 ARM *AT91SAM9260*

O ARM *AT91SAM9260*, ilustrado na Figura 2.2, é um microcontrolador, que contém internamente um conjunto de periféricos que lhe permitem comunicar com outros dispositivos que também comuniquem a partir de USB (*Universal Synchronous Asynchronous Receiver Transmitter*), USART (*Universal Synchronous Asynchronous Receiver Transmitter*), Ethernet, I2C (*Inter-integrated circuit*), entre outros. Internamente tem um processador RISC (*Reduced instruction set computing*) ARM926EJ-S baseado na arquitetura ARM v5TEJ [21].

Os componentes desenvolvidos pela Tecmic, e todos os componentes desenvolvidos no âmbito deste trabalho de projeto devem ser capazes de ser executados na arquitetura ARM v5TEJ.

Figura 2.2: ARM *AT91SAM9260*, imagem extraída de [22]

2.5.2 I2C

O I2C é um *bus* de dados série, *half-duplex* (somente uma linha é utilizada para enviar e receber dados), inventado pela Philips que permite ter vários dispositivos a atuar como *master* e ter vários dispositivos a atuar como *slaves*. Normalmente é utilizado para ligar periféricos que têm velocidades baixas de comunicações de dados a microcontroladores. O *bus* é composto por duas linhas, a linha SCL e a linha SDA. A linha SCL transporta uma onda de relógio de forma a que se possa fazer sincronismo com os dados enviados na linha SDA. Cada periférico ligado ao *bus* tem um endereço próprio de 7 bits, que serve para indicar o destinatário das mensagens enviadas. O I2C permite que um microcontrolador seja capaz de comunicar com periféricos com duas das suas portas de entrada e saída [23]. A Figura 2.3 mostra um conjunto de periféricos *master* e *slaves* ligados ao *bus*.

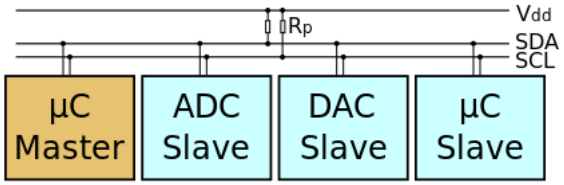

Figura 2.3: Exemplo de *bus* I2C, imagem extraída de [23]

2.5.3 *FXOS8700CQ*

O planeta Terra no seu interior contém metal liquido, em permanente movimento, que cria um campo magnético em redor do planeta. O campo magnético gerado toma valores de 25 a 65 micro Teslas à superfície da Terra [24]. Um tipo de sensores denominados de magnetómetros são capazes de medir campos magnéticos. Como descrito em [20], o *FXOS8700CQ*, ilustrado na Figura 2.4, é um dispositivo que disponibiliza dados em formato digital, obtidos pelos sensores acelerómetro e magnetómetro. O dispositivo conforme a configuração disponibilizada tem a capacidade de obter dados somente do acelerómetro ou magnetómetro, ou de obter os dados de ambos os sensores. Os dados do magnetómetro são disponibilizados em unidades micro Teslas (μT), e os do acelerómetro são disponibilizados em metros por segundo ao quadrado (m/s^2). Os magnetómetros podem receber interferências nas

suas medidas de campo magnético provenientes de linhas de energia e de equipamentos eléctricos e electrónicos.

Os dados obtidos pelo dispositivo podem ser disponibilizados via SPI (*Serial Peripheral Interface*) ou I2C. Os dados obtidos pelos sensores são convertidos por intermédio de um ADC (*Analog to Digital Converter*), com resolução de 14 bits para o acelerómetro e de 16 bits para o magnetómetro. Para se permitir o controlo do dispositivo e obtenção de dados, o dispositivo disponibiliza um conjunto de registos para esse efeito. Encontra-se um resumo dos registos utilizados no âmbito do trabalho na Tabela 2.1 . A Figura 2.5 mostra o diagrama de pinos do *FXOS8700CQ*.

Um endereço de dispositivo I2C pode ser selecionado para o *FXOS8700CQ*. Essa seleção é feita a partir da definição de valores lógicos nas entradas SA0 (pino 7) e SA1 (pino 10) do *FXOS8700CQ*. A Tabela 2.2 mostra os valores lógicos permitidos para se selecionar o endereço desejado. O endereço selecionado atualmente no computador embarcado para o dispositivo *FXOS8700CQ* é o endereço 0x1F.

O dispositivo tem vários modos de funcionamento, donde se destaca o modo de funcionamento híbrido, que é ativado utilizando-se o registo M_CTRL_REG1. O registo M_CTRL_REG1 permite que o dispositivo disponibilize dados de ambos os sensores presentes. Também é necessário por o dispositivo em modo ativo usando para tal o registo CTRL_REG1.

Para se obter um vector tridimensional de campo magnético, são utilizados os registos seguintes:

- M_OUT_X_MSB
- M_OUT_X_LSB
- M_OUT_Y_MSB
- M_OUT_Y_LSB
- M_OUT_Z_MSB
- M_OUT_Z_LSB

Os valores dos registos são convertidos de forma a se obter um vector tridimensional com componentes X,Y e Z. Conforme indicado em [24], a componente X corresponde ao ponto cardeal Norte, a componente Y corresponde ao ponto cardeal Este, e a componente Z aponta para o centro da Terra. O sistema de coordenadas encontra-se ilustrado na Figura 2.6.

Os registos abaixo indicados são utilizados para se obter um vector tridimensional de aceleração:

- OUT_X_MSB
- OUT_X_LSB

- OUT_Y_MSB
- OUT_Y_LSB
- OUT_Z_MSB
- OUT_Z_LSB

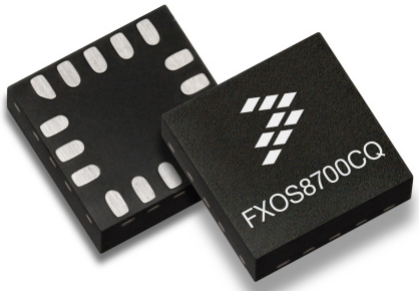

Figura 2.4: Dispositivo *FXOS8700CQ*, imagem extraída de [25]

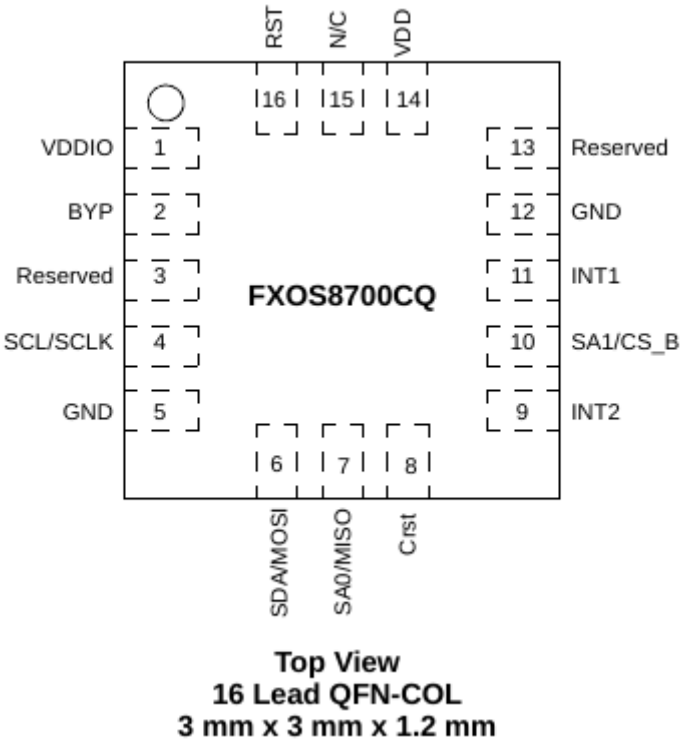

Figura 2.5: Diagrama de pinos do dispositivo *FXOS8700CQ*, imagem extraída de [26]

Registo	Descrição
CTRL_REG1	Registo de controlo do dispositivo
M_CTRL_REG1	Registo de controlo do magnetómetro
M_OUT_X_MSB	Bits mais significativos componente X do vector campo magnético
M_OUT_X_LSB	Bits menos significativos componente X do vector campo magnético
M_OUT_Y_MSB	Bits mais significativos componente Y do vector campo magnético
M_OUT_Y_LSB	Bits menos significativos componente Y do vector campo magnético
M_OUT_Z_MSB	Bits mais significativos componente Z do vector campo magnético
M_OUT_Z_LSB	Bits menos significativos componente Z do vector campo magnético
OUT_X_MSB	Bits mais significativos componente X do vector aceleração
OUT_X_LSB	Bits menos significativos componente X do vector aceleração
OUT_Y_MSB	Bits mais significativos componente Y do vector aceleração
OUT_Y_LSB	Bits menos significativos componente Y do vector aceleração
OUT_Z_MSB	Bits mais significativos componente Z do vector aceleração
OUT_Z_LSB	Bits menos significativos componente Z do vector aceleração

Tabela 2.1: Registos do dispositivo *FXOS8700CQ*, adaptado de [26]

Valor pino SA0	Valor pino SA1	Endereço
0	0	0x1E
0	1	0x1D
1	0	0x1C
1	1	0x1F

Tabela 2.2: Endereços I2C para o dispositivo *FXOS8700CQ*, adaptado de [26]

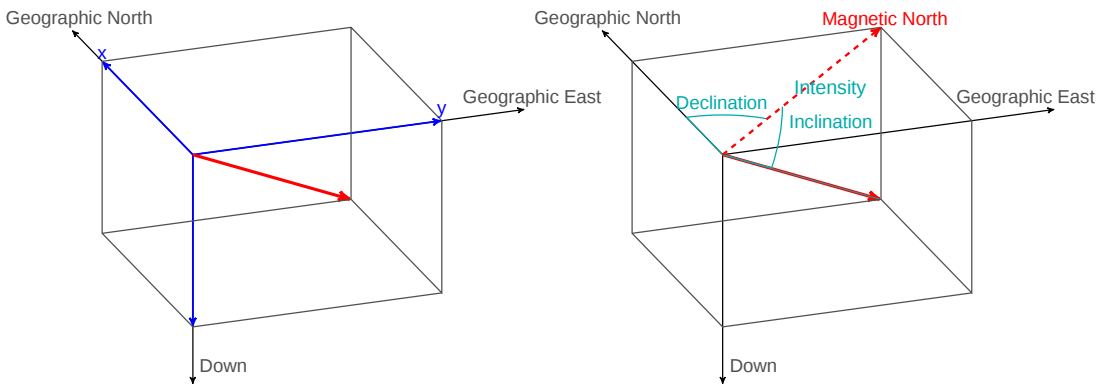

Figura 2.6: Sistema de coordenadas utilizado para representar o campo magnético na Terra, extraído de [27]

2.6 *sysfs*

O *sysfs* é um sistema de ficheiros disponibilizado pelo *kernel* Linux, que permite comunicar com módulos em execução no *kernel*. Esses módulos por sua vez podem ter a responsabilidade de funcionar como *device drivers* de dispositivos existentes no *hardware* do sistema. Normalmente o sistema de ficheiros encontra-se montado nas pasta */sys* de um sistema Linux. O *sysfs* serve como substituto a invocações à função *ioctl()* [28]. A função *ioctl()* tem como objetivo controlar dispositivos conhecidos pelo *kernel* do sistema, onde seja invocada a função [29].

2.7 GPS

Como indicado em [2], o GPS é um sistema de posicionamento global que tem a capacidade de fornecer localização no globo terrestre em determinado momento, necessitando que o equipamento que calcula a posição, o equipamento receptor, tenha em linha de vista pelo menos quatro satélites GPS.

A posição no globo terrestre é indicada a partir um sistema de coordenadas composto por dois componentes, a latitude e longitude. A latitude no globo terrestre corresponde ao ângulo entre o plano equatorial e a linha que passa por um ponto localizado no globo e o centro da Terra. A longitude é uma linha paralela ao Equador. Quanto mais a linha de longitude se afasta do Equador menor é o seu comprimento. A utilização da latitude e longitude permite indicar a posição de qualquer local no globo terrestre. As posições podem ser descritas em graus decimais, ou em graus, minutos e segundos. Vários modelos existem para representar o globo terrestre. O sistema GPS utiliza o modelo WGS84 para representar o globo terrestre, e indicar as posições em latitude e longitude.

A distância de um grau de latitude e longitude vária conforme o grau de latitude em questão. A distância em metros de um grau de latitude pode ser obtido com base na formula (2.1), e para se obter a distância em metros para a longitude a formula (2.2) pode ser usada. A Figura 2.7 mostra o sistema de coordenadas com latitude e longitude. O λ corresponde ao ângulo de longitude e o φ corresponde ao ângulo da latitude.

$$distanciaGrauLatitude = 111132.92 - 559.82\cos(2\varphi) + 1.175\cos(4\varphi) - 0.0023\cos(6\varphi)[km] \quad (2.1)$$

$$distanciaGrauLongitude = 111412.84\cos(\varphi) - 93 - 5\cos(3\varphi) - 0.118\cos(5\varphi)[km] \quad (2.2)$$

Com base nas formulas indicadas anteriormente (2.1) (2.2) criou-se a Tabela 2.3 que indica o número de metros para latitude e longitude para um determinado grau de latitude. Portugal encontra-se na latitude 38º Norte, pelo que segundo a Tabela 2.3 cada 10μ graus de latitude correspondem a 1,109965

metros, e cada 10μ graus de longitude correspondem a 0,876996 metros.

Um dispositivo receptor GPS encontra-se instalado no computador embarcado que tem a capacidade de comunicar com o sistema GPS. O dispositivo é inicializado com a ajuda de um conjunto de comandos indicados na Tabela 2.4, e após essa inicialização são obtidos dados, devolvidos pelo dispositivo receptor GPS que seguem o protocolo NMEA (*National Marine Electronics Association*) [30]. Os dados são obtidos lendo o ficheiro */dev/ttyS5* existente no computador embarcado. O dispositivo receptor GPS normalmente leva cerca de 30 segundos a obter uma posição a partir de GPS, e a partir dessa altura devolve posições válidas. As posições podem ser obtidas a partir dos muitos tipos de mensagens NMEA devolvidas, indicadas na Tabela 2.5. No âmbito do trabalho a mensagem utilizada para se obter posições é a mensagem $GPGGA [31].

A mensagem $GPGGA é composta pelos campos indicados na Tabela 2.6. A mensagem começa com o símbolo $ e acaba com o símbolo correspondente a mudança de linha, que é representado pelo valor 0x13 em hexadecimal. Todos os valores para cada campo existente são separados por virgula. Exemplos de mensagens $GPGGA encontram-se na Tabela 2.7 .

Para se obterem pontos de geo-referência, é necessário tratar as mensagens $GPGGA, verificando se a mensagem contém todos os dados necessários para o ponto de geo-referência ser considerado válido e com qualidade. Um bom indicador de validade é a existência de um valor de HDOP (*Horizontal Dilution of Precision*) inferior a 5 [32], e a qualidade da posição ter um valor maior ou igual a 1. Os valores devolvidos de latitude e longitude são em graus e minutos. Normalmente o valor do grau para a latitude corresponde aos dois primeiros dígitos, e para a longitude corresponde aos três primeiros. Assim sendo os minutos são os dígitos restantes, compostos pelos 2 dígitos à esquerda do separador decimal, neste caso o ponto (.) e os restantes dígitos após o separador.

Por exemplo, se fosse tratada a mensagem $GPGGA seguinte:

- $GPGGA,162254.00,3723.02837,N,12159.39853,W,1,03,2.36,525.6,M,-25.6,M,,*65

Teríamos para a latitude o valor 37º 23,02837’ N (Norte) e para a longitude 121º 59.39853’ W (Oeste). Caso fosse desejável converter para graus decimais, bastaria dividir os minutos da latitude e longitude por 60, e somar o resultado aos graus correspondentes. Caso o ponto cardeal seja S (Sul) ou W (Oeste), deve-se multiplicar por -1. Assim sendo, para este caso o valor em graus decimais seria 37,3838061667 de latitude e -121,9899755 de longitude.

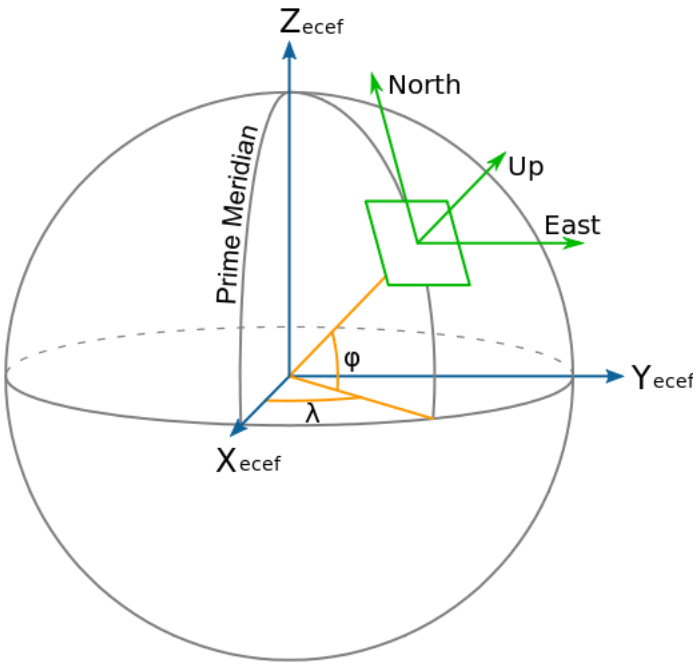

Figura 2.7: Sistema de coordenadas latitude e longitude, extraído de [33]

Ângulo φ de latitude	**Distância de 1 grau de latitude**	**Distância de 10μ graus de latitude**	**Distância de 1 grau de longitude**	**Distância de 10μ graus de longitude**
2	110575.625009	1.105756	111256.826710	1.112568
6	110586.404965	1.105864	110714.161897	1.107142
10	110607.760229	1.106078	109631.482895	1.096315
14	110639.285090	1.106393	108014.077870	1.080141
18	110680.379912	1.106804	105869.846407	1.058698
22	110730.261739	1.107303	103209.260481	1.032093
26	110787.978236	1.107880	100045.312681	1.000453
30	110852.424800	1.108524	96393.451939	0.963935
34	110922.364614	1.109224	92271.507074	0.922715
38	110996.451360	1.109965	87699.598534	0.876996
42	111073.254263	1.110733	82700.038802	0.827000
46	111151.285058	1.111513	77297.222007	0.772972
50	111229.026434	1.112290	71517.503343	0.715175
54	111304.961438	1.113050	65389.068985	0.653891
58	111377.603283	1.113776	58941.797231	0.589418
62	111445.524959	1.114455	52207.111651	0.522071
66	111507.388014	1.115074	45217.827064	0.452178
70	111561.969887	1.115620	38007.989171	0.380080
74	111608.189131	1.116082	30612.708690	0.306127
78	111645.127957	1.116451	23067.990826	0.230680
82	111672.051518	1.116721	15410.560897	0.154106
86	111688.423454	1.116884	7677.686931	0.076777

Tabela 2.3: Distâncias em metros para latitude e longitude por grau de latitude

Ficheiro	**Valor**
/sys/class/gpio/export	50
/sys/class/gpio/export	25
/sys/class/gpio/gpio25/direction	high
/sys/class/gpio/gpio50/direction	high
/sys/class/gpio/gpio25/value	0

Tabela 2.4: Ficheiros de inicialização do receptor GPS

Mensagem	**Descrição**
$GPBOD	*Bearing, origin to destination*
$GPBWC	*Bearing and distance to waypoint, great circle*
$GPGGA	*Global Positioning System Fix Data*
$GPGLL	*Geographic position, latitude / longitude*
$GPGSA	*GPS DOP and active satellites*
$GPGSV	*GPS Satellites in view*
$GPHDT	*Heading, True*
$GPR00	*List of waypoints in currently active route*
$GPRMA	*Recommended minimum specific Loran-C data*
$GPRMB	*Recommended minimum navigation info*
$GPRMC	*Recommended minimum specific GPS/Transit data*
$GPRTE	*Routes*
$GPTRF	*Transit Fix Data*
$GPSTN	*Multiple Data ID*
$GPVBW	*Dual Ground / Water Speed*
$GPVTG	*Track made good and ground speed*
$GPWPL	*Waypoint location*
$GPXTE	*Cross-track error, Measured*
$GPZDA	*Date & Time*

Tabela 2.5: Mensagens NMEA, adaptado de [31]

Campo	Exemplo
Identificador campo	$GPGGA
Hora UTC	170834
Latitude	4124.8963
Ponto cardeal latitude	N
Longitude	08151.6838
Ponto cardeal longitude	W
Qualidade da posição	1
Número de satélites	05
HDOP	1.5
Altitude	280.2
Unidades altitude	M
Altura do geóide acima do elipsóide WGS84	-34.0
Unidades da altura do geóide acima do elipsóide WGS84	M
Hora desde última atualização DGPS	
Identificador da estação de referência DGPS	
Valor de verificação mensagem	* 75

Tabela 2.6: Campos $GPGGA NMEA, adaptado de [31]

Exemplo
$GPGGA,162254.00,3723.02837,N,12159.39853,W,1,03,2.36,525.6,M,-25.6,M,,*65
$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75
$GPGGA,002454,3553.5295,N,13938.6570,E,1,05,2.2,18.3,M,39.0,M,,*7F
$GPGGA,023042,3907.3837,N,12102.4684,W,1,04,2.3,507.3,M,-24.1,M,,*75
$GPGGA,152926,6027.8259,N,02225.6713,E,8,09,2.0,44.7,M,20.6,M,,*79

Tabela 2.7: Exemplos de mensagens $GPGGA, extraídos de [34]

2.8 Odómetro

Grande parte dos veículos de transporte e mercadorias têm instalados dispositivos que permitem contar o número de metros percorridos pelo veículo. Ao dispositivo que tem por objetivo contar o número de metros percorridos dá-se o nome de odómetro. O computador embarcado tem forma de utilizar valores de odómetro disponibilizados por CANBus [35], ou a partir de um contador de impulsos, ao qual é dado um impulso por cada N metros percorridos.

A Tecmic deseja que seja utilizado o odómetro disponibilizado pelo contador de impulsos, devido a ser um odómetro de aplicação universal a todos os seus clientes, quando comparado ao conjunto

mais restrito de clientes que têm veículos onde os valores de odómetro são fornecidos por CANBus.

Para se ativar o contador de impulsos é necessário carregar o módulo de *kernel taco.ko*, criado pela Tecmic. Após o carregamento do módulo é necessário definir valores nos ficheiros de configuração indicados na Tabela 2.8, para que se tenha acesso às leituras do contador de impulsos. Com base nos valores existentes nos ficheiros indicados na Tabela 2.9 é possível calcular o número de metros percorridos num segundo. A forma de cálculo para se obter a distância percorrida em metros é indicada na equação (2.3).

$$distanciaPercorrida = \frac{valorRaw * 1000}{valorCalibscale}[m] \tag{2.3}$$

Ficheiro	**Valor a ser escrito**
/sys/class/gpio/export	79
/sys/class/gpio/gpio79/direction	high
/sys/class/gpio/gpio79/value	1

Tabela 2.8: Ficheiros de configuração do odómetro

Ficheiro	**Significado do valor**
/sys/devices/platform/taco/iio:device0/in_rot0_raw	número de impulsos por segundo
/sys/devices/platform/taco/iio:device0/in_rot0_calibscale	número de impulsos que equivalem a 1000 metros percorridos.

Tabela 2.9: Ficheiros de valores do odómetro

2.9 glibc

A glibc (GNU C Library) é uma biblioteca de funções padrão para a linguagem C, criada pelo projeto GNU, muito utilizada em sistemas que utilizam o *kernel* Linux. Ela contém funções muito conhecidas como por exemplo *printf()*, *scanf()*, *malloc()*. A versão de glibc utilizada no computador embarcado é a 2.20. A biblioteca pode ser compilada para várias arquiteturas de computadores como as indicadas na Tabela 2.10.

Arquitetura
aarch64
alpha
arm
hppa
ia64
m68k
microblaze
mips
nios2
powerpc
s390
sh
sparc
tilegx
tilepro
x86/x86_64

Tabela 2.10: Arquiteturas suportadas pela glibc, adaptado de [36]

2.10 GSL

A GSL (*GNU Scientific Library*) é uma biblioteca com um grande conjunto de funções para cálculo matemático. As funções existentes cobrem algumas das seguintes áreas:

- Números complexos
- Raízes de polinómios
- Vectores e matrizes
- Permutações
- Ordenação
- Álgebra Linear
- Transformadas rápidas de Fourier
- Quadratura
- Números aleatórios
- Distribuições aleatórias

- Estatística
- Histogramas

2.11 ExtJS

O ExtJS é uma *framework* de JavaScript que serve para criar aplicações *web*. A *framework* inclui um conjunto de controlos para construir aplicações *web*. Muitos do controlos disponibilizados são capazes de comunicar com um servido *web* a partir de AJAX (*Asynchronous JavaScript And XML*).

Muitos controlos de utilizador estão disponíveis, como por exemplo:

- Caixa de texto (*Text field*)
- Janela (*Window*)
- Barra de ferramentas (*Toolbar*)
- Grelha (*Grid*)
- Plugin para *Google Maps*

3 Sistema de navegação inercial

O presente capítulo descreve a implementação dos componentes criados para o sistema de navegação inercial. Para que o sistema fosse criado, vários objetivos deviam ser atingidos. Para isso dividiu-se o trabalho em diversas fases, em que para cada fase existe um conjunto de objetivos a ser cumprido. As fases e objetivos associados encontram-se indicados na Tabela 3.1. O sistema de navegação inercial complementa o *XtranX Passenger* para que ele tenha a capacidade de utilizar navegação estimada ou DR para além de navegação electrónica com base no receptor GPS.

Fase	**Objetivo**
Desenvolvimento do módulo *kernel*	Implementação do módulo de *kernel* Linux para adquirir dados dos sensores
Desenvolvimento do algoritmo de navegação inercial	Gerar pontos geo-referenciados válidos com base nos dados dos sensores
Testes	Testar e verificar as previsões de geo-referenciação, tendo por base um último ponto válido obtido por GPS, e vários dados obtidos a partir do sensor magnetómetro

Tabela 3.1: Fases do projecto

Para o sistema de navegação inercial ser capaz de criar as previsões de geo-referenciação, foram criados no âmbito do trabalho de projeto os seguintes componentes:

- Ambiente de desenvolvimento Ubuntu Linux;
- Módulo *kernel Hello World*;
- Módulo *kernel Acc-driver*;
- Programa *calibHardIron*;
- Programa *checkValues*;
- Programa *estimator*;
- Aplicação *web getPoints*.

Os módulos e programas foram criados na linguagem C, tendo como alvo a arquitetura ARM, que é a arquitetura utilizada pelo microcontrolador descrito na subsecção 2.5.1. Todos os módulos e programas após a compilação foram verificados com o comando *file* antes de serem copiados para o computador embarcado, para se ter a garantia de que a arquitetura alvo (ARM) era a correta. A aplicação *web* foi desenvolvida em JavaScript com a biblioteca ExtJS. Os módulos e programas podem

ser compilados no ambiente de desenvolvimento Ubuntu Linux ou Gentoo Linux, bastando alterar no ficheiro *Makefile* o compilador para ARM existente em cada um dos ambientes. No Gentoo Linux o compilador é o *armv5tel-softfloat-linux-gnueabi-gcc* e no Ubuntu Linux com Emdebian é o *arm-linux-gnueabi-gcc*.

Para se instalarem os componentes desenvolvidos, o computador de bordo foi ligado conforme as instruções fornecidas pela Tecmic. Para testar se o ambiente de desenvolvimento criava binários executáveis pelo computador embarcado, criou-se um módulo *kernel* muito simples denominado de *Hello World*, que ao ser carregado e descarregado escreve uma mensagem acessível a partir do comando *dmesg*.

A Figura 3.1 mostra onde ficam colocados os componentes desenvolvidos, com borda verde, que compõem o sistema de navegação inercial, no sistema *XtranX Passenger*. O componente *getPoints* corre num servidor *Web* independente, que não têm qualquer ligação com o *XtranX Passenger*.

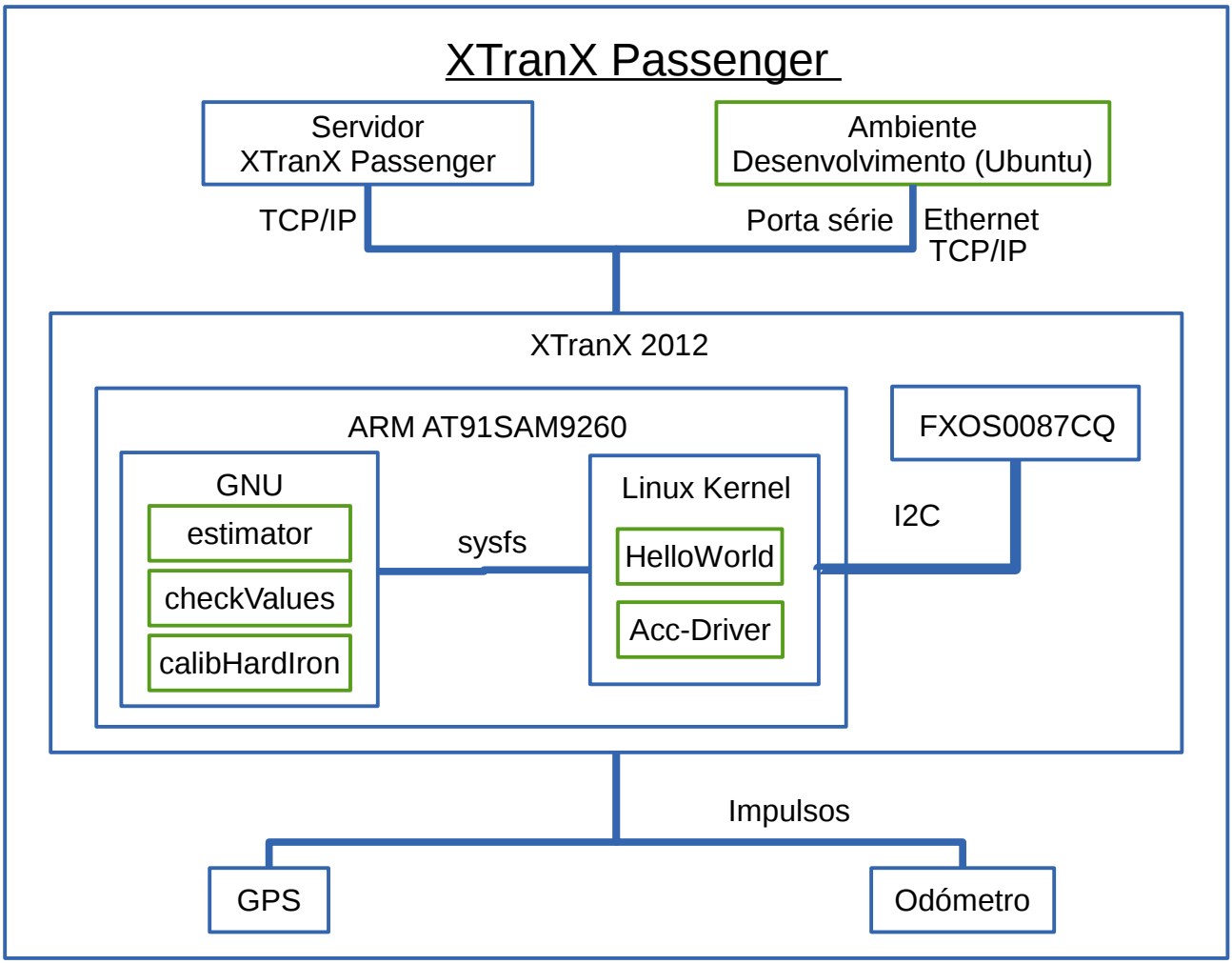

Figura 3.1: Arquitetura do sistema *XtranX Passenger* com os componentes do sistema de navegação inercial

3.1 Ambiente de desenvolvimento Ubuntu Linux

Foi criada uma máquina virtual em Ubuntu Linux, baseada na máquina virtual fornecida pela Tecmic com Gentoo Linux, que corre com o auxílio do Vagrant e do VirtualBox. Decidiu-se criar a máquina virtual em Ubuntu Linux para se obter um conhecimento mais profundo sobre o conjunto de ferramentas de desenvolvimento disponibilizados para ambientes ARM, em sistemas que se baseiam em Debian Linux.

Para se criar a máquina virtual seguiram-se os passos seguintes:

- Instalar o VirtualBox 4.2.18 [37];
- Instalar o Vagrant 1.2.3 [38];

- Obter uma Vagrant *box* lucid32 [39] [40], que corresponde a uma máquina virtual Ubuntu Linux (Ubuntu 10.04.4 LTS (Lucid Lynx));
- Inicializar a máquina virtual com *vagrant up*;
- Após a inicialização executar o *vagrant ssh* para aceder à máquina virtual por SSH (Secure Shell);
- Instalar os pacotes indicados na Tabela 3.2. Os pacotes correspondem a pacotes disponibilizados pelo projeto Emdebian [41], que disponibiliza um conjunto de ferramentas para desenvolvimento de sistemas embebidos;
- Obter o *kernel* Linux 3.13.5 e instalar dentro da máquina virtual;
- Aplicar o *patch* da Tecmic sobre o *kernel* Linux 3.13.5;
- Compilar o *kernel* tendo como arquitetura alvo a arquitetura ARM.

Pacote
linux-libc-dev-armel-cross
libc6-armel-cross
libc6-dev-armel-cross
binutils-arm-linux-gnueabi
gcc-4.4-arm-linux-gnueabi
g++-4.4-arm-linux-gnueabi
pdebuild-cross
dpkg-cross
qemu

Tabela 3.2: Pacotes Emdebian

3.2 Módulo *kernel Hello World*

O componente *Hello World* é um módulo *kernel* que foi desenvolvido com o objetivo de se verificar uma boa compilação e ligação de um módulo *kernel*, e que funcione dentro do computador de bordo. O *Hello World* é copiado após a compilação para o computador de bordo com o comando *scp*. Após efetuada a cópia, acede-se ao computador de bordo por SSH. Ao ser executado o comando *insmod*, a função *helloworld_init()* é invocada, é adicionada uma mensagem ao *log* de mensagens do *kernel*, e o módulo é carregado para o *kernel*. Ao ser executado o comando *rmmod*, a função *helloworld_exit()* é invocada, que também adiciona uma mensagem ao *log* de mensagens do *kernel*, e o módulo é descarregado do *kernel* . De seguida executa-se o comando *dmesg* para se verificar se aparecem as mensagens emitidas na inserção e remoção do módulo *Hello World*.

O módulo *Hello World* exigiu que se estuda-se a forma como construir módulos *kernel* para a arquitetura alvo, que é neste caso a arquitetura ARM.

O projeto do módulo de *kernel* é composto pelos ficheiros indicados na Tabela 3.3. As funções implementadas encontram-se descritas na Tabela 3.4.

Ficheiro
Makefile
copyToEmdebian.sh
helloworld.c

Tabela 3.3: Ficheiros de projeto do módulo *kernel Hello World*

Função	**Objetivo**
helloworld_init()	Mostrar mensagem quando o módulo é carregado no *kernel*
helloworld_exit()	Mostrar mensagem quando o módulo é descarregado do *kernel*

Tabela 3.4: Funções implementadas no módulo *kernel Hello World*

3.3 Módulo *kernel Acc-driver*

O componente *Acc-driver* tem como objetivo permitir a comunicação com o dispositivo *FXOS8700CQ*. Após o carregamento do módulo, um conjunto de ficheiros é criado dentro da pasta */sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/*. Esses ficheiros permitem a comunicação em modo de leitura e escrita com o dispositivo, e correspondem a um determinado tipo de registo disponibilizado pelo dispositivo. Os registos e seus objetivos encontram-se descritos no *datasheet* [26] do dispositivo. Conforme indicado em [28], os ficheiros existentes dentro da pasta */sys* permitem implementar funcionalidades que antes estavam reservadas a partir de chamadas feitas pela função *ioctl()*, a ficheiros existentes sobre a pasta */dev*.

Para o módulo *Acc-driver* precisou-se de estudar a forma como interagir com módulos *kernel* a partir da pasta */sys*. Além disso também se estudou a documentação do dispositivo *FXOS8700CQ*. Para se comunicar com o dispositivo *FXOS8700CQ* a partir do *bus* I2C utilizaram-se funções próprias do *kernel* Linux para esse efeito. Ouve dificuldade em compreender inicialmente as funções I2C do *kernel* pelo que se contactou a Tecmic para ajudar a compreender melhor como interagir com dispositivos ligados ao *bus* I2C. Para se criarem os ficheiros *sysfs* também foram utilizadas funções disponibilizadas pelo *kernel* Linux.

Durante o estudo foi-se implementando o código, compilando e instalando no sistema embarcado, de forma a se verificar se a interação desejada com o dispositivo a partir do módulo estava a ser bem sucedida. Quando se começaram a obter valores da parte do sensor de magnetómetro, utilizaram-se ímanes para verificar se o sensor magnetómetro reagia apropriadamente ao campo magnético apresentado pelo íman. O programa *checkValues* era desenvolvido em paralelo, para ser possível configurar o dispositivo por intermédio do módulo e obter valores dos sensores lá existentes.

O projeto do módulo de *kernel Acc-driver* é composto pelos ficheiros indicados na Tabela 3.5. As funções implementadas encontram-se descritas na Tabela 3.6.

A Figura 3.2 mostra as ligações feitas entre o módulo, o dispositivo *FXOS8700CQ* e os ficheiros criados para aceder aos registos do dispositivo.

O módulo ao ser carregado inicializa o *sysfs* com os ficheiros correspondentes aos registos do *FXOS8700CQ* aos quais se desejam aceder. Depois é desativado o pino AT91_PIN_PC7 do microcontrolador *AT91SAM9260* que se encontra ligado ao pino RST do *FXOS8700CQ*, de forma a se ativar o dispositivo *FXOS8700CQ*. É feita uma pesquisa no *bus* I2C pelo identificador do dispositivo *FXOS8700CQ*. Após estes passos o módulo fica a aguardar por acessos de leitura e escrita aos ficheiros expostos no *sysfs* até o módulo ser descarregado. Se é feita uma leitura num ficheiro *sysfs*, é lido o valor atual do registo correspondente no *FXOS8700CQ* enviando um comando para o *bus* I2C e devolvendo o valor lido de seguida no *bus* I2C para quem efetuou a leitura num formato hexadecimal. Caso seja realizada uma escrita, é enviado o valor escrito e um comando para o *bus* I2C para escrever o valor no registo correspondente. As etapas anteriores encontram-se ilustradas pela Figura 3.3.

Ficheiro
FXOS8700CQ_VB_core.c
FXOS8700CQ_VB_core.h
Makefile
checkValues.sh
copyToEmdebian.sh

Tabela 3.5: Ficheiros de projeto do módulo *kernel Acc-driver*

Função	**Objetivo**
handler_mctrlreg1()	Escreve o valor do registo M_CTRL_REG1 no ficheiro lido em *sysfs*
handler_m_out_x_msb()	Escreve o valor do registo M_OUT_X_MSB no ficheiro lido em *sysfs*
handler_m_out_x_lsb()	Escreve o valor do registo M_OUT_X_LSB no ficheiro lido em *sysfs*
handler_m_out_y_msb()	Escreve o valor do registo M_OUT_Y_MSB no ficheiro lido em *sysfs*
handler_m_out_y_lsb()	Escreve o valor do registo M_OUT_Y_LSB no ficheiro lido em *sysfs*
handler_m_out_z_msb()	Escreve o valor do registo M_OUT_Z_MSB no ficheiro lido em *sysfs*
handler_m_out_z_lsb()	Escreve o valor do registo M_OUT_Z_LSB no ficheiro lido em *sysfs*
handler_ctrl_reg1()	Escreve o valor do registo CTRL_REG1 no ficheiro lido em *sysfs*
handler_ctrl_reg2()	Escreve o valor do registo CTRL_REG2 no ficheiro lido em *sysfs*
handler_ctrl_reg3()	Escreve o valor do registo CTRL_REG3 no ficheiro lido em *sysfs*
handler_ctrl_reg4()	Escreve o valor do registo CTRL_REG4 no ficheiro lido em *sysfs*
handler_ctrl_reg5()	Escreve o valor do registo CTRL_REG5 no ficheiro lido em *sysfs*
handler_out_x_msb	Escreve o valor do registo OUT_X_MSB no ficheiro lido em *sysfs*
handler_out_x_lsb()	Escreve o valor do registo OUT_X_LSB no ficheiro lido em *sysfs*
handler_out_y_msb()	Escreve o valor do registo OUT_Y_MSB no ficheiro lido em *sysfs*
handler_out_y_lsb()	Escreve o valor do registo OUT_Y_LSB no ficheiro lido em *sysfs*
handler_out_z_msb()	Escreve o valor do registo OUT_Z_MSB no ficheiro lido em *sysfs*
handler_out_z_lsb()	Escreve o valor do registo OUT_Z_LSB no ficheiro lido em *sysfs*
handler_read()	Escreve valor de um registo escolhido num ficheiro lido em *sysfs*
default_show()	Devolve valores quando uma leitura é feita sobre um ficheiro no *sysfs*
default_store()	Guarda valores quando uma escrita é feita sobre um ficheiro no *sysfs*
fxos8700cq_i2c_write()	Escreve para o dispositivo no bus I2C
fxos8700cq_i2c_read()	Lê resposta do dispositivo no bus I2C
free_sysfs()	Liberta os atributos do módulo no *sysfs*
init_sysfs()	Inicializa os atributos do módulo no *sysfs*
fxos8700cq_remove()	Liberta os recursos para comunicar com o dispositivo
fxos8700cq_i2c_probe()	Procura o dispositivo no *bus* I2C
fxos8700cq_i2c_remove()	Remove o módulo de *kernel*
fxos8700cq_i2c_shutdown()	Desliga o dispositivo

Tabela 3.6: Funções implementadas no módulo *kernel Acc-driver*

Ficheiro	**Tipo de acesso**
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/sysmod	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/temp	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/whoami	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/acc_x_lsb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/acc_x_msb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/acc_y_lsb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/acc_y_msb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/acc_z_lsb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/acc_z_msb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/ctrlreg1	Leitura e Escrita
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/ctrlreg2	Leitura e Escrita
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/ctrlreg3	Leitura e Escrita
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/ctrlreg4	Leitura e Escrita
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/ctrlreg5	Leitura e Escrita
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/mctrlreg1	Leitura e Escrita
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/mx_lsb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/mx_msb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/my_lsb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/my_msb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/mz_lsb	Leitura
/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/mz_msb	Leitura

Tabela 3.7: Ficheiros no *sysfs* do dispositivo, criados pelo *Acc-driver*

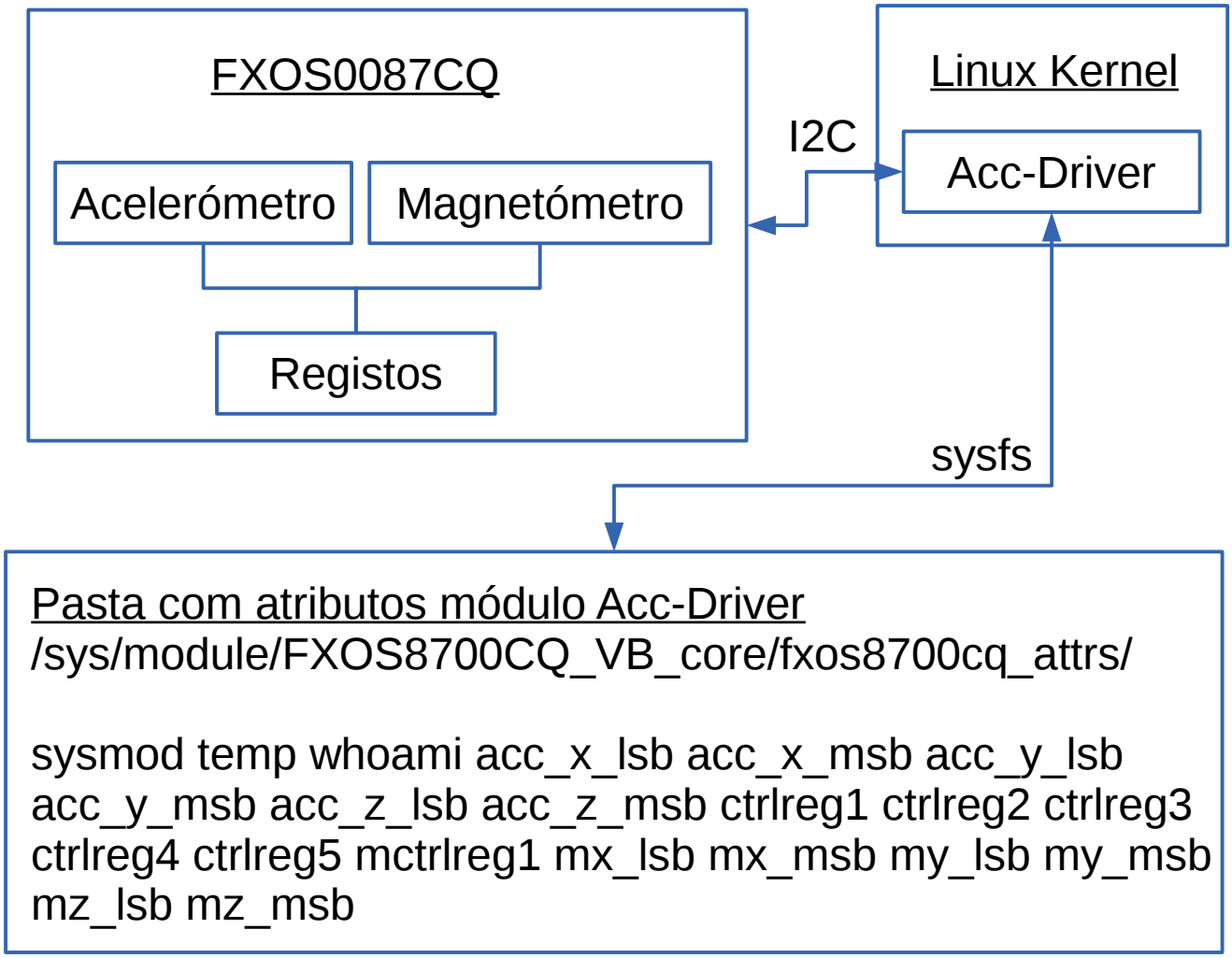

Figura 3.2: Diagrama do módulo *kernel Acc-driver*

Figura 3.3: Fluxograma do módulo *kernel Acc-driver*

3.4 Programa *checkValues*

O programa *checkValues* inicializa o dispositivo *FXOS8700CQ* e obtém amostras dos sensores acelerómetro e magnetómetro de três em três segundos, com base nos valores existentes nos ficheiros

indicados na Tabela 3.7. Para além disso aplica sobre as amostras obtidas do sensor magnetómetro um vector de calibração, obtido a partir do programa *calibHardIron*, que é passado como argumento de entrada do mesmo. Os vectores de magnetómetro obtidos são gravados no ficheiro */tmp/mxmymz.samples*. Para inicializar o dispositivo o programa *checkValues* necessita do módulo *kernel Acc-driver* carregado, pois ele disponibiliza os ficheiros indicados na Tabela 3.7. O fluxograma do programa *checkValues* encontra-se ilustrado na Figura 3.4.

O dispositivo *FXOS8700CQ* é inicializado escrevendo para ficheiros na pasta **/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/**. Escreve-se para o ficheiro *ctrlreg1* o valor correspondente a *ACTIVE_MODE* que é 1. De seguida escreve-se para o ficheiro *mctrlreg1* o valor correspondente a *HYBRID_MODE* que é 3.

Para se obterem os dados dos sensores, os valores existentes nos ficheiros dentro da pasta **/sys/module/FXOS8700CQ_VB_core/fxos8700cq_attrs/** são lidos. Os ficheiros lidos são os seguintes:

- acc_x_lsb
- acc_x_msb
- acc_y_lsb
- acc_y_msb
- acc_z_lsb
- acc_z_msb
- mx_lsb
- mx_msb
- my_lsb
- my_msb
- mz_lsb
- mz_msb

A cada valor do magnetómetro que contém os *bits* mais significativos (msb) é aplicada uma deslocação de 8 *bits* para a esquerda (*shift left*) que corresponde a multiplicar o valor por 256. Depois a esse valor é somado o valor dos *bits* menos significativos (lsb). O resultado para cada componente é multiplicado por 0,1. De seguida aplica-se o vector de calibração sobre cada componente (3.1) (3.2)

(3.3). A fórmula (3.4) mostra como se calcula a magnitude do campo magnético com base nas componentes do vector M. A multiplicação por 0,1 deve-se ao facto de cada *bit* corresponder a 0,1 [μT], conforme indicado em [26].

$$MX = ((mx\_msb \cdot 256) + mx\_lsb) \cdot 0.1 - VX \tag{3.1}$$

$$MY = ((my\_msb \cdot 256) + my\_lsb) \cdot 0.1 - VY \tag{3.2}$$

$$MZ = ((mz\_msb \cdot 256) + mz\_lsb) \cdot 0.1 - VZ \tag{3.3}$$

$$|M| = \sqrt{MX^2 + MY^2 + MZ^2} \tag{3.4}$$

Para cada valor do acelerómetro o cálculo é parecido. Efetua-se um deslocamento de 6 *bits* para a esquerda para os *bits* mais significativos, um deslocamento para a direita (*shift right*) de 2 *bits* para os *bits* menos significativos, conforme se indica nas fórmulas (3.5) (3.6) (3.7).

$$ACCX = (accx\_msb \cdot 64) + (accx\_lsb/4) \tag{3.5}$$

$$ACCY = (accy\_msb \cdot 64) + (accy\_lsb/4) \tag{3.6}$$

$$ACCZ = (accz\_msb \cdot 64) + (accz\_lsb/4) \tag{3.7}$$

De seguida apresenta-se um exemplo de cálculo (3.8) (3.9) (3.10) para valores oriundos dos ficheiros mx_lsb, mx_msb ,my_lsb, my_msb, mz_lsb e mz_msb. O vector de calibração tem as suas componentes a 0. Os valores em representação hexadecimal são convertidos para valores em representação

decimal.

$$
\begin{aligned}
VX &= 0 \\
VY &= 0 \\
VZ &= 0 \\
mx\_lsb &= 0x90 \\
mx\_msb &= 0x02 \\
MX &= ((mx\_msb \cdot 256) + mx\_lsb) \cdot 0.1 - VX \\
&= (mx\_msb \cdot 256 + mx\_lsb) \cdot 0,1 - 0 \\
&= (0x02 \cdot 256 + 0x90) \cdot 0,1 \\
&= ((0 \cdot 16 + 2) \cdot 256 + (9 \cdot 16 + 0)) \cdot 0,1 \\
&= (512 + 144) \cdot 0,1 \\
&= (656) \cdot 0,1 \\
&= 65,6[\mu T]
\end{aligned} \tag{3.8}
$$

$$
\begin{aligned}
my\_lsb &= 0xBA \\
my\_msb &= 0xFE \\
MY &= ((my\_msb \cdot 256) + my\_lsb) \cdot 0.1 - VY \\
&= (my\_msb \cdot 256 + my\_lsb) \cdot 0,1 - 0 \\
&= (0xFE \cdot 256 + 0xBA) \cdot 0,1 \\
&= ((15 \cdot 16 + 14) \cdot 256 + (11 \cdot 16 + 10)) \cdot 0,1 \\
&= (65024 + 186) \cdot 0,1 \\
&= (65210) \cdot 0,1 \\
&= 6521,0[\mu T]
\end{aligned} \tag{3.9}
$$

$$
\begin{aligned}
mz\_lsb &= 0xBF \\
mz\_msb &= 0xFF \\
MZ &= ((mz\_msb \cdot 256) + mz\_lsb) \cdot 0.1 - VZ \\
&= (mz\_msb \cdot 256 + mz\_lsb) \cdot 0,1 - 0 \\
&= (0xFF \cdot 256 + 0xBF) \cdot 0,1 \\
&= ((15 \cdot 16 + 15) \cdot 256 + (11 \cdot 16 + 15)) \cdot 0,1 \\
&= (65280 + 191) \cdot 0,1 \\
&= (65471) \cdot 0,1 \\
&= 6547,1[\mu T]
\end{aligned} \tag{3.10}
$$

Para se calcular o ângulo do magnetómetro, é necessário calcular o arco tangente da divisão de MY por MX, e converter esse valor para graus, conforme indicado na fórmula (3.11). Após efectuado esse cálculo levando em consideração o valor do cálculo, e os valores de MX e MY, é identificado o valor do ângulo. O processo de identificação do ângulo é indicado na Figura 3.6.

$$atanAngulo = \arctan(MY/MX) \cdot \frac{180}{\pi} \tag{3.11}$$

De seguida apresenta-se um exemplo de cálculo do ângulo(3.12) utilizando os valores de (3.8) (3.9). Como MX e MY são superiores a 0, segundo o fluxograma 3.6 o ângulo é igual a 89,423 graus.

$$\begin{aligned} MX &= 65,6 \\ MY &= 6521,0 \\ atanAngulo &= \arctan(MY/MX) \cdot \frac{180}{\pi} \\ &= \arctan(6521,0/65,6) \cdot \frac{180}{\pi} \\ &= 1,560737 \cdot \frac{180}{\pi} \\ &= 89,423[^\circ] \end{aligned} \tag{3.12}$$

O diagrama existente na Figura 3.5 mostra os componentes com os quais o programa *checkValues* comunica para que seja capaz de obter amostras dos sensores acelerómetro e magnetómetro.

Ficheiro
Makefile
buildCheckValues.sh
checkValues.c
checkValues.h

Tabela 3.8: Ficheiros de projeto do programa *checkValues*

Função	**Objetivo**
signal_handler()	Intercepta sinais enviados ao programa como por exemplo CTRL+C ou um sinal SIGTERM enviado pelo comando kill
read_int_value()	Lê um valor de um ficheiro presente na Tabela 3.7
write_int_value()	Escreve um valor de um ficheiro presente na Tabela 3.7
initialize_sensor()	Inicializa o dispositivo *FXOS8700CQ* que contém os sensores
init_signals_handler()	Inicializa a intercepção de sinais
get_values_sensor()	Devolve valores dos sensores de magnetómetro e acelerómetro
get_angle()	Devolve um ângulo em graus com base nos valores do magnetómetro
main()	Ponto de entrada principal do programa

Tabela 3.9: Funções implementadas no programa *checkValues*

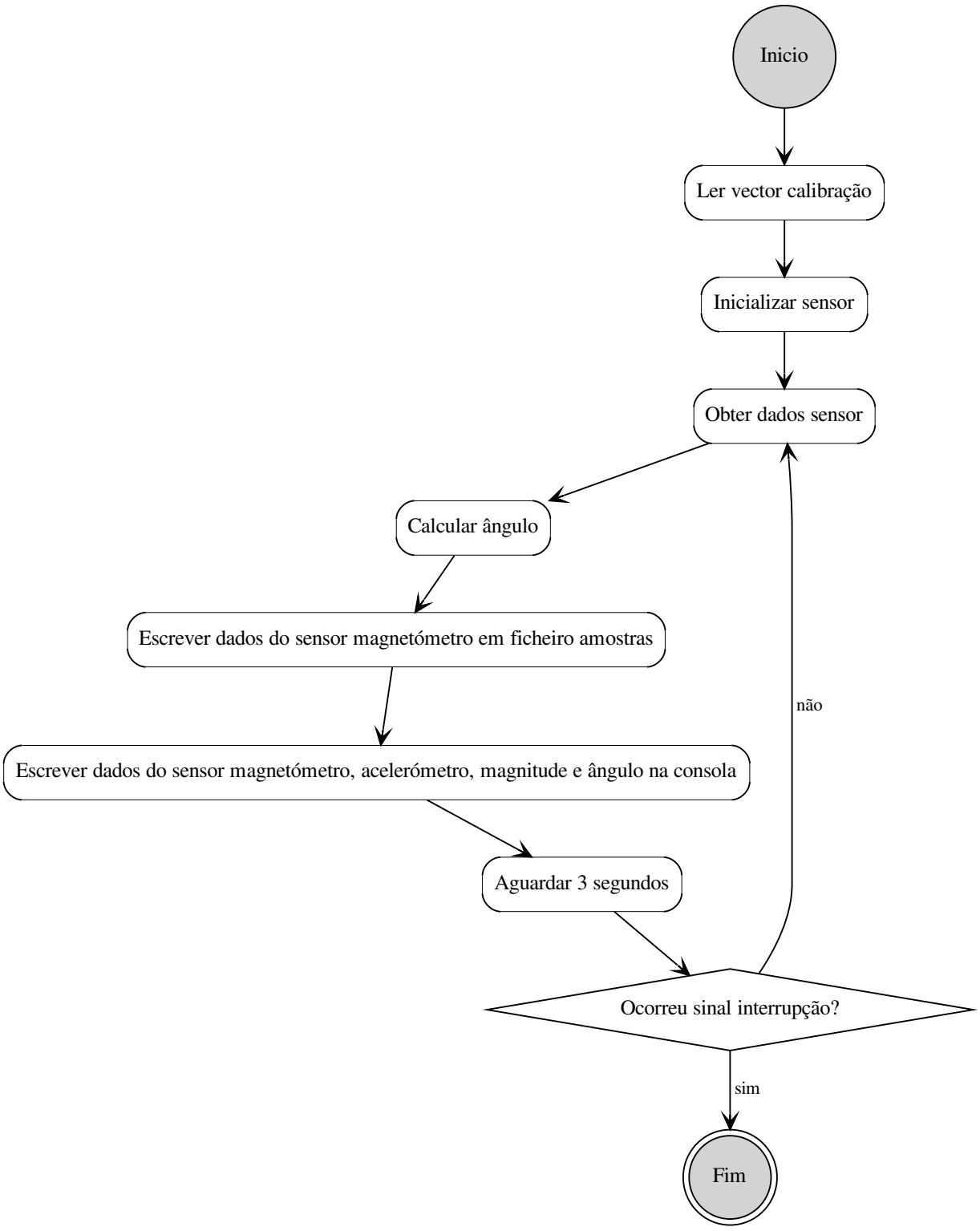

Figura 3.4: Fluxograma do programa *checkValues*

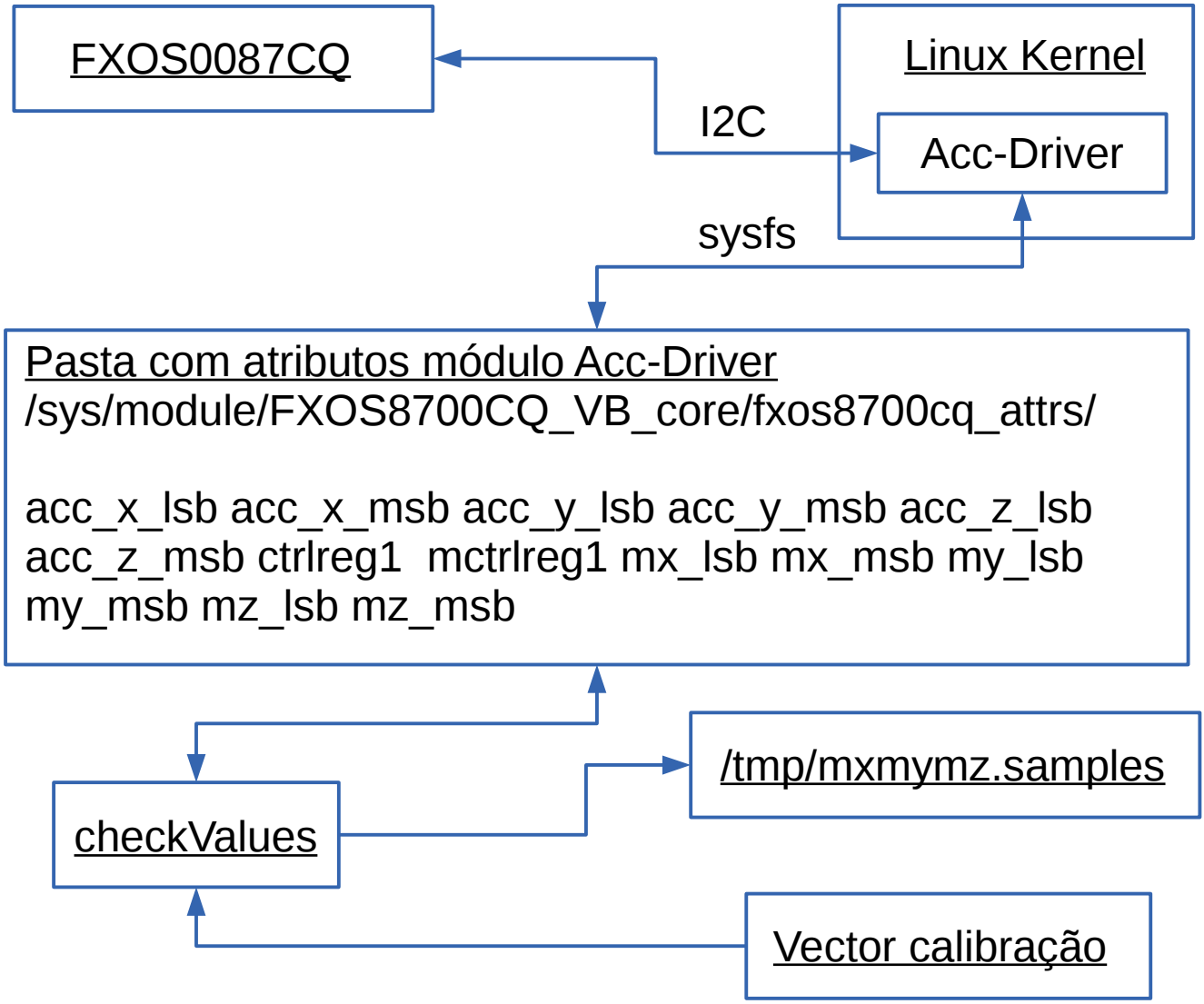

Figura 3.5: Diagrama do programa *checkValues*

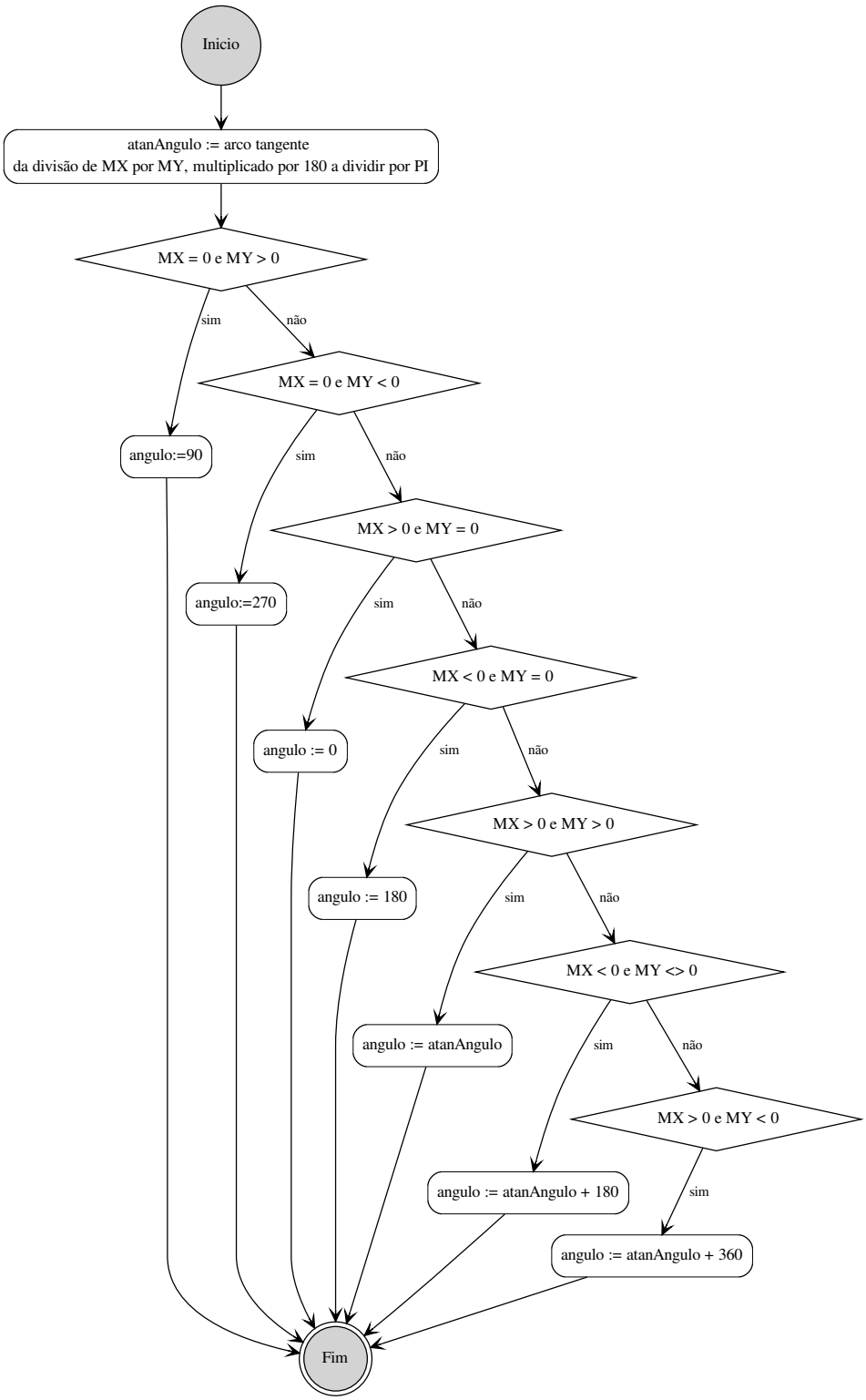

Figura 3.6: Cálculo do ângulo do magnetómetro

3.5 Programa *calibHardIron*

O programa *calibHardIron* calcula valores de calibração para colmatar as interferências de *hard-iron* causadas por criação de campos magnéticos, devidos ao próprio funcionamento da placa de circuito

impresso do computador de bordo. Necessita da biblioteca GSL [42] para a utilização de funções de cálculo matricial, necessárias para o cálculo dos valores de calibração. A forma de se chegar aos valores de calibração encontra-se descrita no documento AN4246 [43] da Freescale. Foi estudado o documento AN4246, para se desenvolver o programa, e avaliar se valia a pena o esforço de implementar funções de cálculo matricial próprias, ou em alternativa, utilizar bibliotecas já existentes como o GSL. Devido ao risco inerente à implementação *Ad-Hoc*, de criar *bugs*, e devido à existência de bibliotecas maduras para esse efeito, decidiu-se utilizar a GSL. Caso fosse tomado o caminho de uma implementação *Ad-Hoc* a parte mais difícil seria a de implementar e testar a inversa de uma matriz, com bases em vários algoritmos existentes [44] para esse efeito, como por exemplo o algoritmo de *Eliminação Gauss-Jordan* [45]. As unidades utilizadas pelo vector de calibração são μT.

Para se realizar o cálculo com o *calibHardIron* deve-se indicar um ficheiro de texto com amostras de valores de vectores do magnetómetro, obtido a partir do programa *checkValues*. O ficheiro gerado pelo *checkValues* encontra-se em */tmp/mxmymz.samples*. O *checkValues* deve ter como argumentos as componentes de um vector de calibração a zeros (./checkValues 0 0 0).

Para se obterem as amostras de valores de vectores do magnetómetro, deve-se executar o programa *checkValues* tendo como argumentos as componentes de um vector de calibração a zeros (./checkValues 0 0 0). De seguida deve-se rodar o computador embarcado em todas as direcções possíveis. De seguida para-se o programa *checkValues*. O ficheiro */tmp/mxmymz.samples* é gerado pelo *checkValues* que contém as componentes do vectores do magnetómetro, MX, MY e MZ. De notar que se deve ter um mínimo de 6 amostras obtidas pelo *checkValues*. Um exemplo de um ficheiro */tmp/mxmymz.samples* encontra-se na Listagem 3.1.

O cálculo do vector de calibração é feito com base nas fórmulas indicadas em (3.13) (3.14) (3.15) (3.16) que se encontram descritas com mais detalhe em [43]. O fluxograma ilustrado pela Figura 3.7 mostra os passos dados para se calcular o vector de calibração composto pelas componentes V_x,V_y e V_z.

$$\beta = (X^T X)^{-1} X^T Y \tag{3.13}$$

$$V_x = \frac{1}{2}\beta[0] \tag{3.14}$$

$$V_y = \frac{1}{2}\beta[1] \tag{3.15}$$

$$V_z = \frac{1}{2}\beta[2] \tag{3.16}$$

A Tabela 3.10 mostra os ficheiros de projeto para o *calibHardIron*, e a Tabela 3.11 mostra as funções criadas.

Com os dados presentes na Listagem 3.1 mostra-se de seguida um exemplo de cálculo do vector de calibração. A matriz 3.17 contém as amostras presentes na Listagem 3.1. A matriz Y é construída com base na matriz 3.17 calculando a magnitude de um vector (3.18) sobre cada linha da matriz 3.17. A matriz Y resultante encontra-se em (3.19). A matriz X (3.20) é uma cópia da matriz de amostras com uma coluna extra preenchida com o valor 1. Com as matrizes X e Y é possível calcular a matriz β (3.21) usando as operações matriciais indicadas na fórmula (3.13). Os componentes calculados para o vector de calibração encontram-se em (3.22) (3.23) (3.24) (3.25).

$$Amostras = \begin{pmatrix} 59,50 & 6489,10 & 6521,60 \\ 59,50 & 6489,90 & 6520,90 \\ 79,20 & 6533,40 & 6520,40 \\ 76,70 & 6531,60 & 6524,40 \\ 15,40 & 6505,10 & 6524,50 \\ 14,00 & 6503,20 & 6520,50 \\ 31,90 & 6546,60 & 6523,40 \\ 32,10 & 6549,40 & 6519,70 \\ 71,00 & 6502,80 & 6523,80 \\ 73,10 & 6498,10 & 6523,70 \end{pmatrix} \tag{3.17}$$

$$magnitudeVector = \sqrt{MX^2 + MY^2 + MZ^2} \tag{3.18}$$

$$Y = \begin{pmatrix} 84643225,620 \\ 84644479,070 \\ 85207204,360 \\ 85235476,810 \\ 84885663,420 \\ 84808726,490 \\ 85413736,730 \\ 85402158,860 \\ 84851415,280 \\ 84789308,910 \end{pmatrix} \tag{3.19}$$

$$X = \begin{pmatrix} 59,50 & 6489,10 & 6521,60 & 1,00 \\ 59,50 & 6489,90 & 6520,90 & 1,00 \\ 79,20 & 6533,40 & 6520,40 & 1,00 \\ 76,70 & 6531,60 & 6524,40 & 1,00 \\ 15,40 & 6505,10 & 6524,50 & 1,00 \\ 14,00 & 6503,20 & 6520,50 & 1,00 \\ 31,90 & 6546,60 & 6523,40 & 1,00 \\ 32,10 & 6549,40 & 6519,70 & 1,00 \\ 71,00 & 6502,80 & 6523,80 & 1,00 \\ 73,10 & 6498,10 & 6523,70 & 1,00 \end{pmatrix} \tag{3.20}$$

$$\beta = (X^T X)^{-1} X^T Y = \begin{pmatrix} 91,836 \\ 13038,060 \\ 13014,091 \\ -84840156,877 \end{pmatrix} \tag{3.21}$$

$$V = \frac{1}{2} \begin{pmatrix} \beta[0] \\ \beta[1] \\ \beta[2] \end{pmatrix} \tag{3.22}$$

$$V_x = V[0] = 45,918 \tag{3.23}$$

$$V_y = V[1] = 6519,030 \tag{3.24}$$

$$V_z = V[2] = 6507,045 \tag{3.25}$$

O diagrama existente na Figura 3.8 mostra os componentes com os quais o programa *calibHardIron* comunica para que seja capaz de gerar o vector de calibração.

Ficheiro
Makefile
buildCalibHardIron.sh
calibHardIron.c
calibHardIron.h
samples1.txt

Tabela 3.10: Ficheiros de projeto do programa *calibHardIron*

Função	**Objetivo**
showMatrix()	Mostra uma matriz na consola
calculateMatrixY()	Calcula a matriz Y da fórmula (3.13)
calculateMatrixX()	Calcula a matriz X da fórmula (3.13)
showCalibratedSamples()	Mostra as amostras com o vector de calibração calculado com a ajuda das fórmulas (3.14) (3.15) (3.16)
multiply()	Multiplica duas matrizes
main()	Ponto de entrada principal do programa

Tabela 3.11: Funções implementadas no programa *calibHardIron*

Excerto de código 3.1: Amostras de vectores magnetómetro para o *calibHardIron*

```
59.500000  6489.100000  6521.600000
59.500000  6489.900000  6520.900000
79.200000  6533.400000  6520.400000
76.700000  6531.600000  6524.400000
15.400000  6505.100000  6524.500000
14.000000  6503.200000  6520.500000
31.900000  6546.600000  6523.400000
32.100000  6549.400000  6519.700000
71.000000  6502.800000  6523.800000
73.100000  6498.100000  6523.700000
```

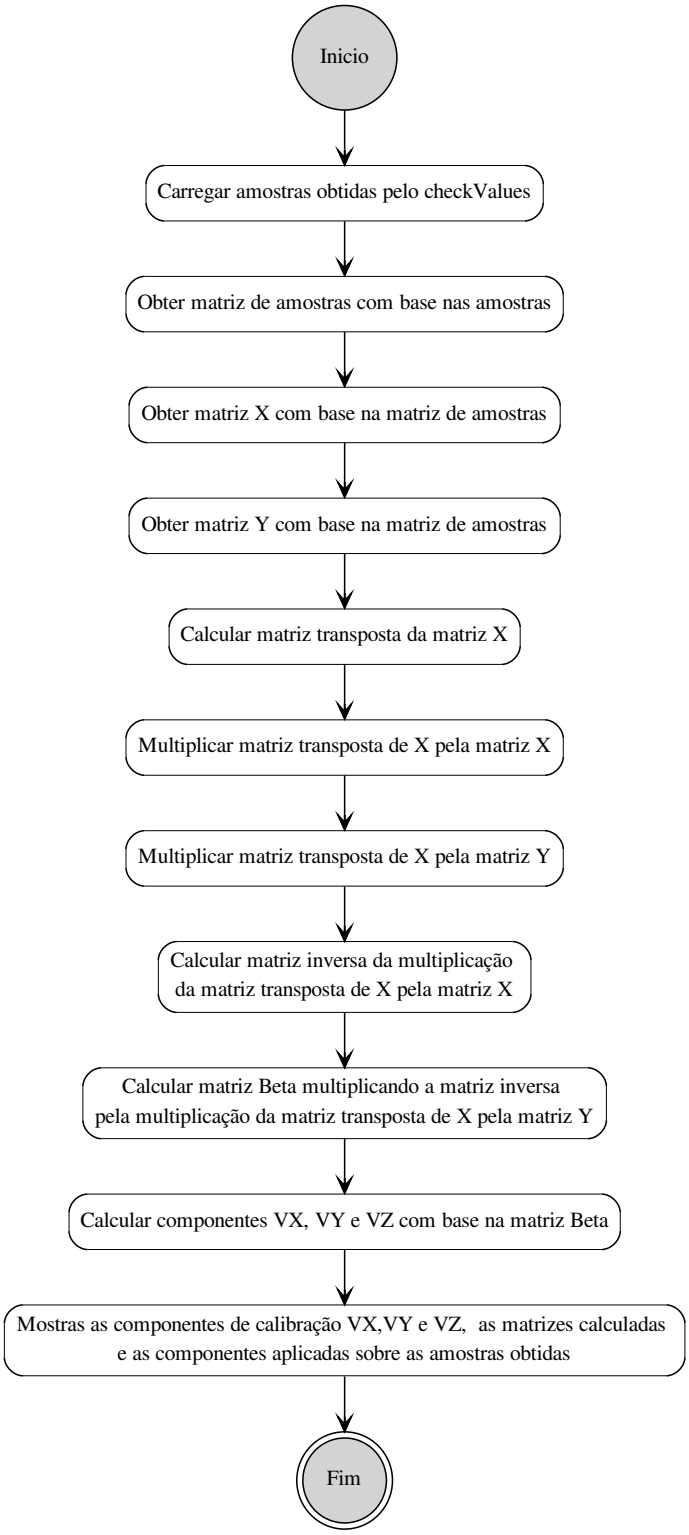

Figura 3.7: Fluxograma do programa *calibHardIron*

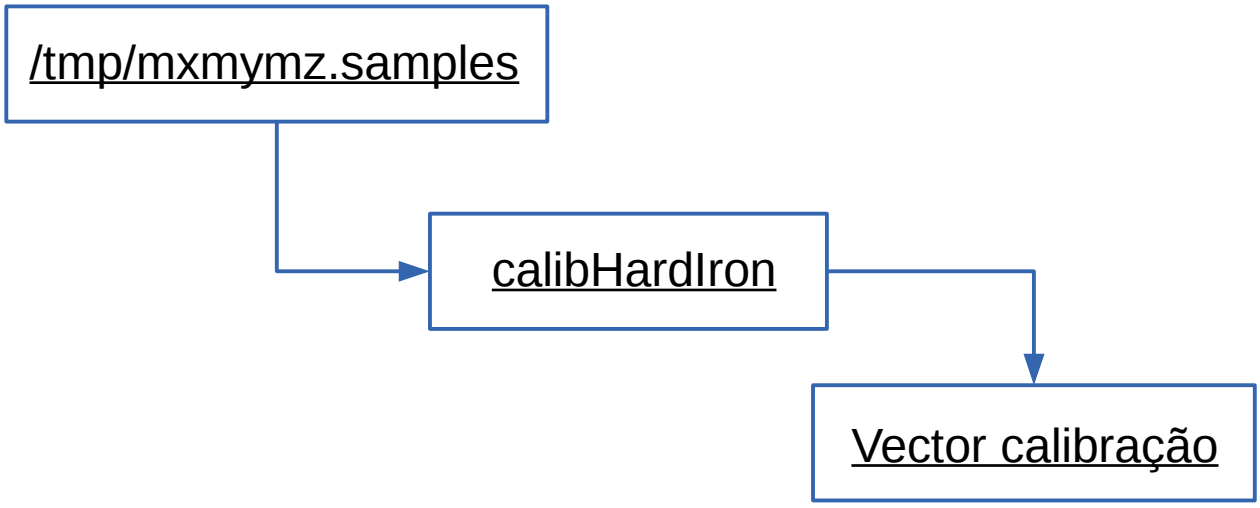

Figura 3.8: Diagrama do programa *calibHardIron*

3.6 Programa *estimator*

O programa *estimator* tem como objetivo estimar uma posição geo-referenciada com base no último ponto válido de GPS, em conjunto com dados do odómetro e o ângulo obtido com base nos dados do sensor magnetómetro. Ou seja, tem como objetivo realizar navegação estima ou DR.

A posição atual GPS é obtida a partir da leitura de mensagens NMEA $GPGGA do ficheiro */dev/ttyS5*. A posição é considerada válida quando o HDOP da posição se encontra entre 0 e 5.

O odómetro atualiza a cada segundo o valor de metros percorridos no ficheiro */sys/devices/platform/-taco/iio:device0/in_rot0_raw*. O valor é convertido com a ajuda do valor existente em */sys/devices/-platform/taco/iio:device0/in_rot0_calibscale* para metros. A fórmula de conversão encontra-se em (2.3).

O ângulo em graus do magnetómetro é obtido com base nos valores presentes nos ficheiros indicados na Tabela 3.7. Tal como para o programa *checkValues*, o programa *estimator* também precisa que se forneça o vector de calibração para o magnetómetro como argumento.

Para cada segundo que passa são guardadas num *array* circular, amostras com os valores da posição GPS atual, o ângulo do magnetómetro em graus, os metros percorridos e os valores do acelerómetro em metros por segundo ao quadrado. O *array* circular permite guardar várias amostras ao longo de vários segundos, para que o *estimator* seja capaz de estimar a posição geo-referenciada. A Tabela 3.12 mostra um exemplo do *array* circular com dados.

TS	Ângulo	Odómetro	Latitude	Longitude	HDOP
18	10	19,180	40,096349	-7,471279	1
19	20	17,257	40,096490	-7,471207	1
20	20	18,658	40,096646	-7,471134	5
21	10	19,681	40,096810	-7,471065	5
22	10	21,372	40,096989	-7,470990	5

Tabela 3.12: Exemplo de dados no *array* circular

Para se estimar a posição geo-referenciada, deve-se continuamente, desde a última posição GPS obtida considerada válida, ler a distância percorrida em metros pelo odómetro, e obter o ângulo em graus obtido com base nos valores do sensor magnetómetro. Com o valor da distância percorrida em metros e o ângulo deve-se criar um vector que representa a direção (ângulo) e a distância em metros percorrida. Esse processo servirá para se criarem vectores, que aplicados juntamente com a última posição GPS conhecida, devem ajudar a se conhecer a posição atual do computador embarcado. Os vectores calculados são convertidos para graus decimais de forma a ser possível se representar o ponto estimado num sistema de coordenadas, como ilustrado pela Figura 2.7. O vector da distância percorrida em metros com determinada direcção é calculado como indicado nas fórmulas (3.26) e (3.27).

$$metrosPercor_x = metrosOdometro \cdot cos(\frac{anguloMagnetoGraus \cdot \pi}{180}) \tag{3.26}$$

$$metrosPercor_y = metrosOdometro \cdot sin(\frac{anguloMagnetoGraus \cdot \pi}{180}) \tag{3.27}$$

Após se obter o vector da distância percorrida em metros com determinada direcção é necessário converter as componentes, pois a distância de um grau de latitude e longitude vária conforme o grau de latitude em questão. Portugal encontra-se na latitude 38º Norte, pelo que segundo a Tabela 2.3 cada 10μ graus de latitude correspondem a 1,109965 metros, e cada 10μ graus de longitude correspondem a 0,876996 metros. Ou seja, para se obter um vector em graus decimais com base no vector da distância percorrida é necessário utilizar as fórmulas (3.28) e (3.29), que utilizam os valores ajustados para Portugal.

$$\Delta lat = metrosPercor_x \cdot \frac{10\mu}{1,109965} \tag{3.28}$$

$$\Delta lng = metrosPercor_y \cdot \frac{10\mu}{0,876996} \tag{3.29}$$

As estimativas são guardadas num ficheiro no computador embarcado que se encontra em */tmp/dados.json*. Os dados são guardados em formato JSON degradado. Idealmente o ficheiro deveria guardar os dados em forma de *array* JSON, mas como se escrevem os dados periodicamente torna-se complicado guardarem-se os dados dessa forma. Assim sendo, escolheu-se guardar os dados de uma forma degrada, que depois é reconstruída pela aplicação *web getPoints* para a forma correcta em JSON, para de seguida se mostrarem na aplicação *web getPoints*. A Listagem 3.2 mostra um exemplos dos dados guardados em JSON degradado. Para se ter um JSON correcto bastaria acrescentar o parêntese reto ([) ao inicio dos dados, retirar a vírgula (,) no final e acrescentar o parêntese reto (]) ao final.

Excerto de código 3.2: Estimativas em JSON

```
{"ts":263, "dist":0.000000, "angle":27.271808,
"xmeters":0.000000, "ymeters":0.000000,
"estPos":{"lat":40.136151,"lng":-7.513828},
"prevGPSPos":{"lat":9999.000000,"lng":9999.000000},
"currGPSPos":{"lat":40.136150,"lng":-7.513832},
"acc":{"x":16316.000000,"y":16348.000000,"z":4124.000000} ,
"currGPSValid":1 ,"currGPSts":262 , "prevGPSts":261 ,"mode":1 } ,
{"ts":264, "dist":0.000000, "angle":24.930700,
"xmeters":0.000000, "ymeters":0.000000,
"estPos":{"lat":40.136150,"lng":-7.513832},
"prevGPSPos":{"lat":40.136150,"lng":-7.513832},
"currGPSPos":{"lat":40.136150,"lng":-7.513833},
"acc":{"x":16348.000000,"y":16340.000000,"z":4128.000000} ,
"currGPSValid":1 ,"currGPSts":263 , "prevGPSts":262 ,"mode":0 } ,
{"ts":265, "dist":0.000000, "angle":27.390394,
"xmeters":0.000000, "ymeters":0.000000,
"estPos":{"lat":40.136150,"lng":-7.513833},
"prevGPSPos":{"lat":40.136150,"lng":-7.513833},
"currGPSPos":{"lat":40.136148,"lng":-7.513834},
"acc":{"x":16328.000000,"y":16328.000000,"z":4084.000000} ,
"currGPSValid":1 ,"currGPSts":264 , "prevGPSts":263 ,"mode":0 } ,
```

Com base nos dados presentes na Tabela 3.12 apresenta-se um exemplo de cálculo de uma estimativa. Para o instante de tempo TS=20 o cálculo de estimativa é feito da seguinte maneira:

- Obtém-se os dados para TS-1 ou seja a linha com TS=19, denominada amostraAtual.
- Obtém-se os dados para TS-2 ou seja a linha com TS=18, denominada amostraAnterior.

De seguida calcula-se a distância percorrida em metros dividida em componentes *x* e *y* com base nas

fórmulas (3.26) e (3.27).

$$\begin{aligned} metrosPercor_x &= amostraAtual['odometro'] \cdot \cos(amostraAtual['angulo'] \cdot \pi/180) \\ &= 17,257 \cdot \cos(20 \cdot \pi/180) \\ &= 16,216[metros] \\ metrosPercor_y &= amostraAtual['odometro'] \cdot \sin(amostraAtual['angulo'] \cdot \pi/180) \\ &= 17,257 \cdot \sin(20 \cdot \pi/180) \\ &= 5,902[metros] \end{aligned}$$

As componentes x e y da distância são convertidas para graus decimais conforme as fórmulas (3.28) e (3.29).

$$\begin{aligned} \Delta lat &= metrosPercor_x \cdot 0,00001/1,109965 \\ &= 16,216 \cdot 0,00001/1,109965 \\ &= 146,1[\mu^\circ] \\ \Delta lng &= metrosPercor_y \cdot 0,00001/0,876996 \\ &= 5,902 \cdot 0,00001/0,876996 \\ &= 67,298[\mu^\circ] \end{aligned}$$

Como o HDOP da amostraAnterior e amostraAtual são iguais a 1 considera-se que os pontos GPS de ambas as amostras são válidos. Assim sendo a estimativa tem latitude igual a 40,0964951º e longitude igual a -7,471121702º conforme se mostra nas equações (3.30) (3.31). A estimativa é guardada em memória para cálculos posteriores.

$$\begin{aligned} estLatitude &= amostraAnterior['latitude'] + \Delta lat \\ &= 40,096349 + 146,1\mu \\ &= 40,0964951[^\circ] \end{aligned} \tag{3.30}$$

$$\begin{aligned} estLongitude &= amostraAnterior['longitude'] + \Delta lng \\ &= -7,471279 + 67,298\mu \\ &= -7,471121702[^\circ] \end{aligned} \tag{3.31}$$

Para o instante de tempo TS=21 o cálculo de estimativa é feito da seguinte maneira:

- Obtém-se os dados para TS-1 ou seja a linha com TS=20, denominada amostraAtual.
- Obtém-se os dados para TS-2 ou seja a linha com TS=19, denominada amostraAnterior.

Como indicado anteriormente calcula-se a distância percorrida em metros dividida em componentes

x e y com base nas fórmulas (3.26) e (3.27).

$$\begin{aligned} metrosPercor_x &= amostraAtual['odometro'] \cdot \cos(amostraAtual['angulo'] \cdot \pi/180) \\ &= 18,658 \cdot \cos(20 \cdot \pi/180) \\ &= 17,532[metros] \\ metrosPercor_y &= amostraAtual['odometro'] \cdot \sin(amostraAtual['angulo'] \cdot \pi/180) \\ &= 18,658 \cdot \sin(20 \cdot \pi/180) \\ &= 6,381[metros] \end{aligned}$$

As componentes da distância são convertidas para graus decimais conforme as fórmulas (3.28) e (3.29).

$$\begin{aligned} \Delta lat &= metrosPercor_x \cdot 0,00001/1,109965 \\ &= 17,532 \cdot 0,00001/1,109965 \\ &= 157,95[\mu^\circ] \\ \Delta lng &= metrosPercor_y \cdot 0,00001/0,876996 \\ &= 6,381 \cdot 0,00001/0,876996 \\ &= 72,76[\mu^\circ] \end{aligned}$$

Como o HDOP da amostraAnterior é 1 e o da amostraAtual é 5 considera-se que o ponto GPS da amostraAtual não é válido. Como não se tem as duas amostra válidas deve-se utilizar a estimativa anterior com latitude igual a 40,0964951º e longitude igual a -7,471121702º. Com base na estimativa anterior a nova estimativa tem latitude igual a 40,0966531º e longitude igual a -7,471048942º conforme se mostra nas equações (3.32) (3.33).

$$\begin{aligned} estLatitude &= estLatitude + \Delta lat \\ &= 40,0964951 + 157,95\mu \\ &= 40,0966531[^\circ] \end{aligned} \tag{3.32}$$

$$\begin{aligned} estLongitude &= estLongitude + \Delta lng \\ &= -7,471121702 + 72,76\mu \\ &= -7,471048942[^\circ] \end{aligned} \tag{3.33}$$

Para esta estimativa seria criado um JSON degradado descrito na Listagem 3.3.

Excerto de código 3.3: Estimativa em JSON criado com base em exemplo de cálculo

```
{"ts":21 , "dist":18.658, "angle":20.0, "xmeters":17.532,
"ymeters":6.381,
"estPos": {"lat":40.0966531,"lng":-7.471048942} ,
"prevGPSPos": {"lat":40.096490, "lng": -7,471207} ,
"currGPSPos": {"lat":40.096646, "lng": -7.471134 } ,
```

```
"currGPSts":20, "prevGPSts":19, "mode":1  , "currGPSValid":0
"acc":{"x":16348.000000,"y":16340.000000  ,"z":4128.000000}
}  ,
```

Dado	**Descrição**
ts	Data e hora da estimativa em número de segundos decorridos desde o ano 1970
dist	Distância em metros do odómetro
angle	Ângulo magnetómetro em graus
xmeters	Componente X do vector que representa a distância percorrida com determinada orientação
ymeters	Componente Y do vector que representa a distância percorrida com determinada orientação
currGPSPos	Posição GPS atual em graus decimais
prevGPSPos	Posição GPS anterior em graus decimais
acc	Valores do sensor acelerómetro
prevGPSts	Data e hora da estimativa da amostra anterior GPS em número de segundos decorridos desde o ano 1970
currGPSts	Data e hora da estimativa da amostra atual GPS em número de segundos decorridos desde o ano 1970
estPos	Posição estimada em graus decimais
mode	Indica se a posição anterior e atual de GPS são válidas. Válidas se igual a 0, inválidas de igual a 1

Tabela 3.13: Campos por objecto JSON de estimativa

A Figura 3.9 mostra o fluxograma do algoritmo implementado. Os ficheiro de projeto do *estimator* encontram-se na Tabela 3.14, e as funções implementadas encontram-se na Tabela 3.15.

O diagrama existente na Figura 3.10 mostra os componentes com os quais o programa *estimator* comunica para que seja capaz de criar as estimativas.

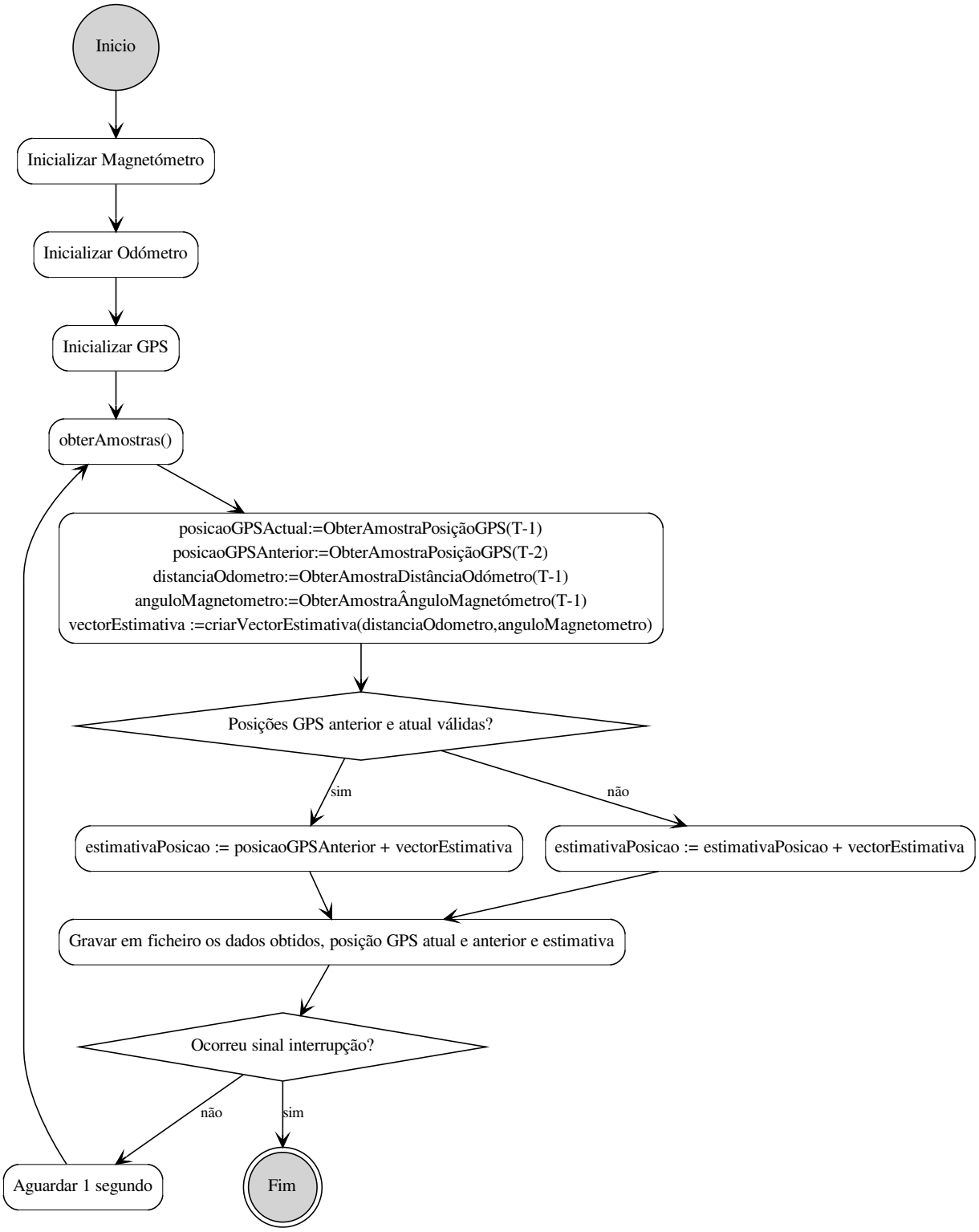

Figura 3.9: Algoritmo de estimação

Ficheiro
Makefile
buildEstimator.sh
gpggaSamples.txt
estimator.c
estimator.h

Tabela 3.14: Ficheiros de projeto do programa *estimator*

Função	**Objetivo**
signal_handler()	Intercepta sinais enviados ao programa como por exemplo CTRL+C ou um sinal SIGTERM enviado pelo comando kill
read_int_value()	Lê um valor de um ficheiro presente na Tabela3.7
write_int_value()	Escreve um valor de um ficheiro presente na Tabela3.7
initialize_sensor()	Inicializa o dispositivo *FXOS8700CQ* que contém os sensores
init_signals_handler()	Inicializa a intercepção de sinais
get_values_sensor()	Devolve valores dos sensores de magnetómetro e acelerómetro
get_angle()	Devolve um ângulo em graus com base nos valores do magnetómetro
init_gps()	Inicializa o dispositivo receptor GPS
init_tacho()	Inicializa o odómetro
write_string_value()	Escreve uma cadeia de caracteres para um ficheiro
get_values_from_GPS()	Extrai valores de uma mensagem NMEA $GPGGA
replace_comma_by_space()	Substitui virgulas por espaços numa cadeia de caracteres
load_GPGGA_samples()	Carrega amostra de mensagens NMEA $GPGGA de um ficheiro
get_latitude_longitude()	Devolve a latitude e longitude existentes numa mensagem NMEA $GPGGA
gps_thread()	*Thread* que periodicamente lê valores do receptor GPS
tacho_thread()	*Thread* que periodicamente lê valores do odómetro
init_estimator()	Inicializa o estimador
estimator_thread()	*Thread* que periodicamente cria estimativas de posição
init_sample()	Inicializa o *array* circular que guarda as amostras de posição GPS. odómetro, ângulo e acelerómetro
fill_sample_angle()	Preenche ângulo no *array* circular para determinado período no tempo
fill_sample_odometer()	Preenche distância percorrida no *array* circular para determinado período no tempo
fill_sample_gps()	Preenche posição GPS no *array* circular para determinado período no tempo
fill_sample_acc()	Obtém valores de acelerómetro do *array* circular para determinado período no tempo
get_sample()	Obtém amostra do *array* circular para determinado período no tempo
show_samples()	Mostra as amostras existentes no *array* na consola
main()	Ponto de entrada principal do programa

Tabela 3.15: Funções implementadas no programa *estimator*

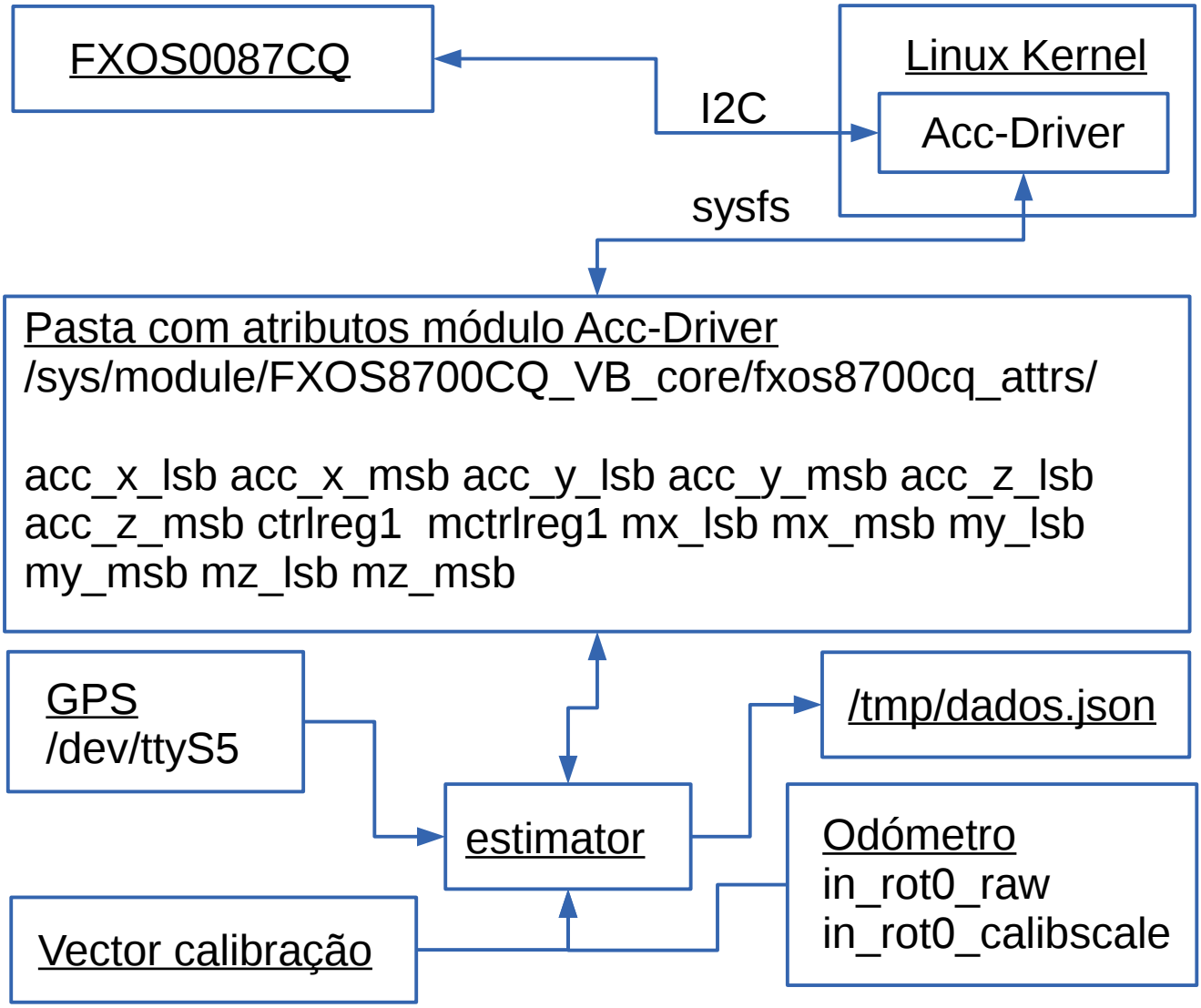

Figura 3.10: Diagrama do programa *estimator*

3.7 Aplicação *web getPoints*

A aplicação *web getPoints* permite mostrar e obter pontos de geo-referenciação, em graus decimais no *Google Maps*, e permite também mostrar as estimativas geradas pelo programa *estimator* gravadas no ficheiro */tmp/dados.json*. A Figura 4.27 ilustra a janela de carregamento de pontos em formato JSON(*JavaScript Object Notation*) degradado. A aplicação é útil para gerar pontos de geo-referenciação de teste para quando ainda não existem dados reais gerados com computadores embarcados nos clientes Tecmic, e para comparar os dados reais com as estimativas para quando existirem dados reais provenientes dos computadores embarcados.

O excerto de código 3.4 corresponde a um exemplo JSON de pontos possíveis de serem carrega-

dos, e o excerto de código 3.5 contém um exemplo de JSON degradado.

A Tabela 3.16 mostra os ficheiros de projeto para a aplicação *web* getPoints, e a Tabela 3.17 mostra as funções implementadas. A Figura 3.11 mostra as entidades com as quais a aplicação comunica.

Excerto de código 3.4: Pontos JSON

```
[
{ "lat":40.13609, "lng":-7.51417, "icon":"info.png",
  "desc":"Est - Ponto 1" },
{ "lat":40.13665, "lng":-7.51383, "icon":"info.png",
  "desc":"Est - ponto 2" },
{ "lat":40.13674, "lng":-7.51322, "icon":"info.png",
  "desc":"Est - ponto 3" },
{ "lat":40.13671, "lng":-7.51218,  "icon":"info.png",
  "desc":"Est - ponto 4" }
]
```

Excerto de código 3.5: Estimativas em JSON degradado

```
[
{"ts":1, "dist":0, "angle":0 , "xmeters":0, "ymeters":0,
"prevGPSPos":{"lat":0,"lng":0},
"estPos":{"lat":40.13609,"lng":-7.51417},
"currGPSPos":{"lat":40.13607046388973,"lng":-7.5141978263854 98}},
{"ts":2, "dist":0, "angle":0 , "xmeters":0, "ymeters":0,
"prevGPSPos":{"lat":0,"lng":0},
"estPos":{"lat":40.13665,"lng":-7.51383 },
"currGPSPos":{"lat":40.13662822235713,"lng":-7.5138115882873535}},
{"ts":3, "dist":0, "angle":0 , "xmeters":0, "ymeters":0,
"prevGPSPos":{"lat":0,"lng":0},
"estPos":{"lat":40.13674,"lng":-7.51322 },
"currGPSPos":{"lat":40.136718447561364 ,"lng":-7.513200044631958}},
{"ts":4, "dist":0, "angle":0 , "xmeters":0, "ymeters":0,
"prevGPSPos":{"lat":0,"lng":0},
"estPos":{"lat":40.13671,"lng":-7.51218 },
"currGPSPos":{"lat":40.13669384069938 ,"lng":-7.51215934753418}},
{"ts":5, "dist":0, "angle":0 , "xmeters":0, "ymeters":0,
"prevGPSPos":{"lat":0,"lng":0},
"estPos":{"lat":40.13639,"lng":-7.51128  },
"currGPSPos":{"lat":40.13637395068243  ,"lng":-7.511258125305176}},
{"ts":6, "dist":0, "angle":0 , "xmeters":0, "ymeters":0,
"prevGPSPos":{"lat":0,"lng":0},
```

```
"estPos":{"lat":40.13618,"lng":-7.51015   },
"currGPSPos":{"lat": 40.136160689834526,"lng": -7.510131597518921}},
{"ts":7, "dist":0, "angle":0  , "xmeters":0, "ymeters":0,
"prevGPSPos":{"lat":0,"lng":0},
"estPos":{"lat":40.13617,"lng":-7.50927    },
"currGPSPos":{"lat":40.136152487480864,"lng":-7.509251832962036 }},
```


Ficheiro
deploy.sh
getPoints.html
app/tese.js
extjs/*
images/info.png
images/sunny.png
js/jquery-1.7.2.min.js
estimativaPontosAldeiaJoanesAltran.json
pontosAldeiaJoanesAltran.json

Tabela 3.16: Ficheiros de projeto da aplicação *web getPoints*

Função	**Objetivo**
definePONTOSModel()	Define o modelo de suporte aos pontos
getGoogleMap()	Devolve objeto para utilizar o *Google Maps*
clearMarkers()	Limpa todos os marcadores e linhas do *Google Maps*
addMarker()	Adiciona marcador no *Google Maps*
loadDataStore()	Carrega grelha com dados
getData()	Cria *array* para suportar dados da grelha
getDataStore()	Cria a *datastore* da grelha
getColumns()	Cria colunas da grelha
createGrid()	Cria a grelha
createGridWindow()	Cria janela contentora da grelha
getGMapItems()	Define opções para o *Google Maps*
createGMapWindow()	Cria janela com o *Google Maps*
createWindowWithText()	Cria janela com grelha dos pontos
loadPointsWindow()	Cria janela para carregar pontos em formato JSON
loadPointsClicked()	Trata de evento associado ao botão de Carregar Pontos
timerJSON()	Temporizador invocado de 2 em 2 segundos
setZoom6()	Define nível de ampliação do *Google Maps* para 6
setClickHandler()	Define função que trata do evento associado ao clique sobre o *Google Maps*
mapClicked()	Trata do evento associado ao clique sobre o *Google Maps*
createToolbar()	Cria a *toolbar* com vários botões
loadPointsHandler()	Trata do evento associado ao clique do botão Carregar Pontos em JSON
clearPointsHandler()	Trata do evento associado ao clique do botão Limpar Pontos
showPointsHandler()	Trata do evento associado ao clique do botão Mostrar pontos em JSON

Tabela 3.17: Funções implementadas na aplicação *web getPoints*

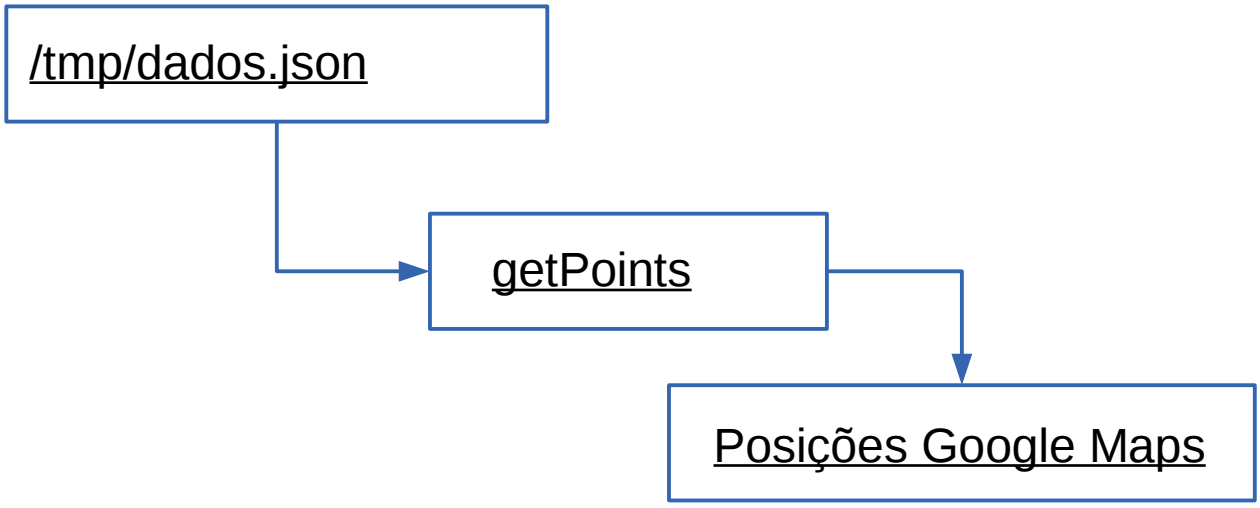

Figura 3.11: Diagrama da aplicação *web getPoints*

4 Testes

O presente capítulo descreve os testes efetuados sobre os componentes implementados, descritos no capítulo anterior.

Para se testarem os componentes, após a compilação dos mesmos, eles foram instalados no computador embarcado, após o computador embarcado se encontrar devidamente ligado a uma fonte de alimentação, a um *switch* por cabo de rede e a uma porta USB a partir de um conversor *Serial-USB*, como ilustrado na Figura 4.1. Para se alimentar o computador embarcado utilizou-se uma tensão de 15 [V]. Para se efetuar a cópia e instalação dos componentes é necessário uma ligação feita por porta série, e após essa ligação estar ativa é necessário ativar a porta de rede do computador embarcado, obtendo um endereço IP para ele. O processo é o seguinte:

- Ligar cabo USB-Série
- Ligar-se ao computador embarcado por porta série, ilustrado pela Figura 4.2
- Ativar ligação por cabo de rede e obtenção de endereço IP, ilustrado pela Figura 4.3.

As secções seguintes mostram os testes efetuados sobre cada componente.

Figura 4.1: Ligações do computador embarcado

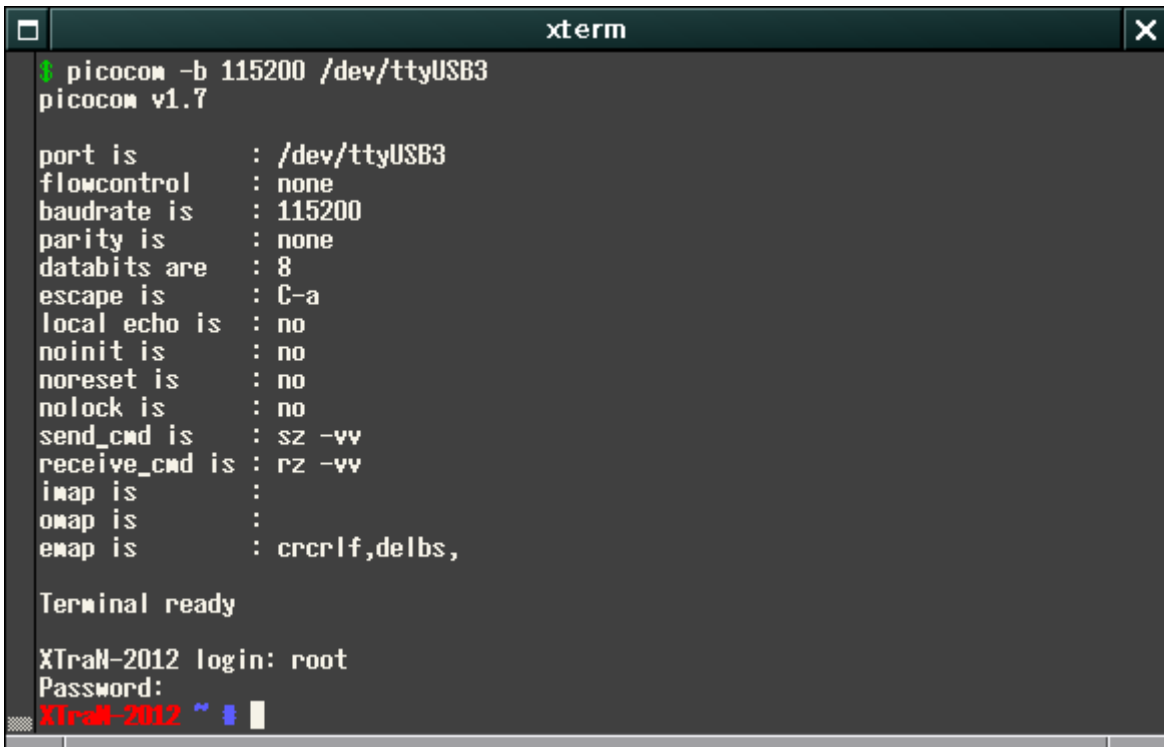

Figura 4.2: Ligação ao computador embarcado por porta série

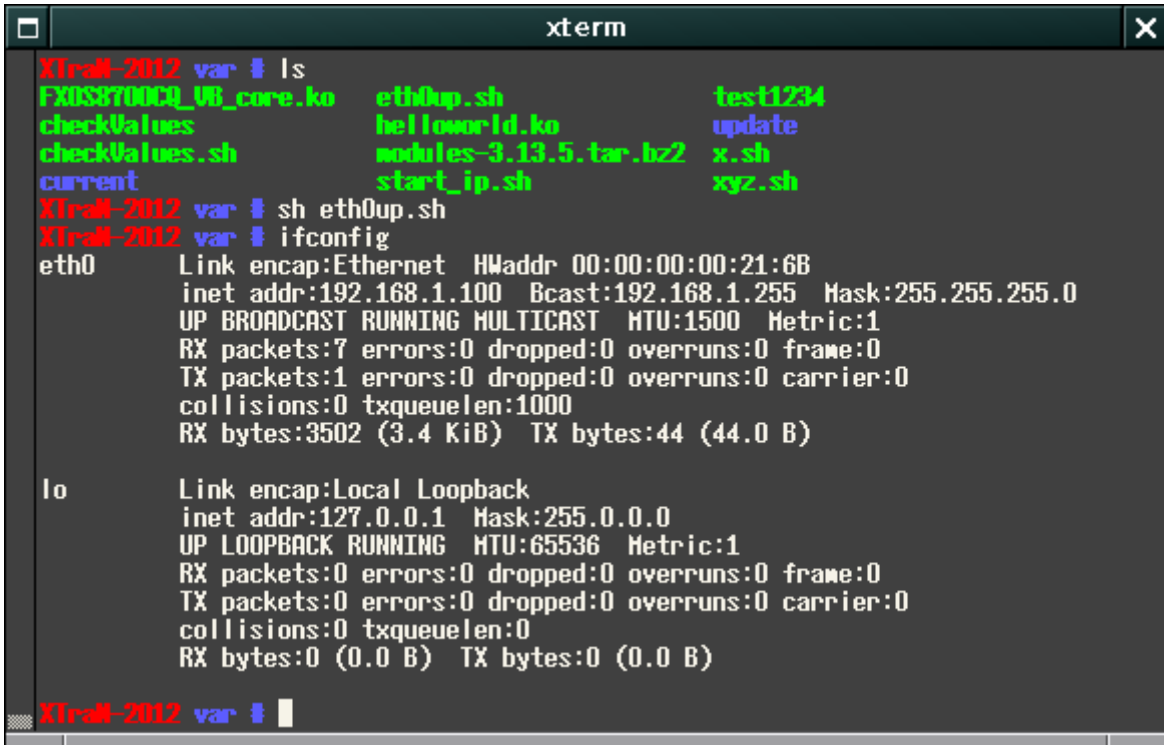

Figura 4.3: Ativação da ligação ao computador embarcado por cabo de rede

4.1 Teste *Hello World*

Tendo por base uma ligação *Ethernet*, foi feita a cópia do componente *Hello World* e carregamento do mesmo, conforme ilustrado pelas Figuras 4.4 e 4.5. Na Figura 4.5 verifica-se o bom carregamento do módulo ao aparecer a mensagem *Hello World*.

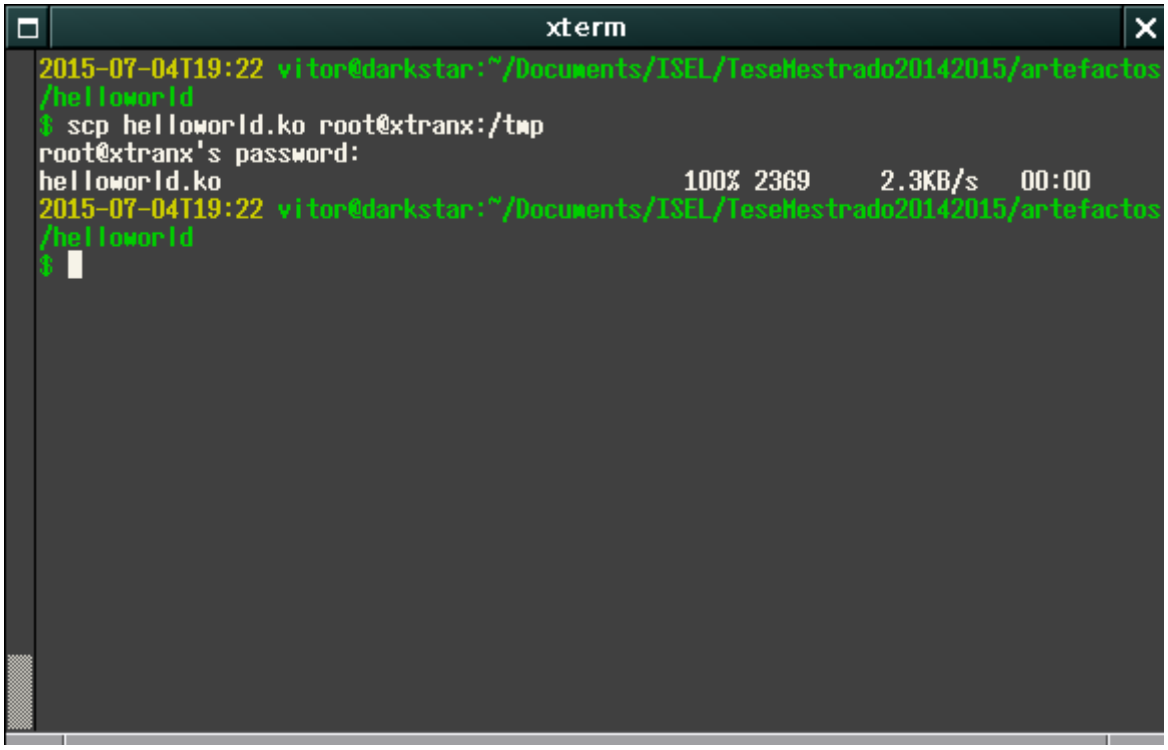

Figura 4.4: Cópia do componente *Hello World*

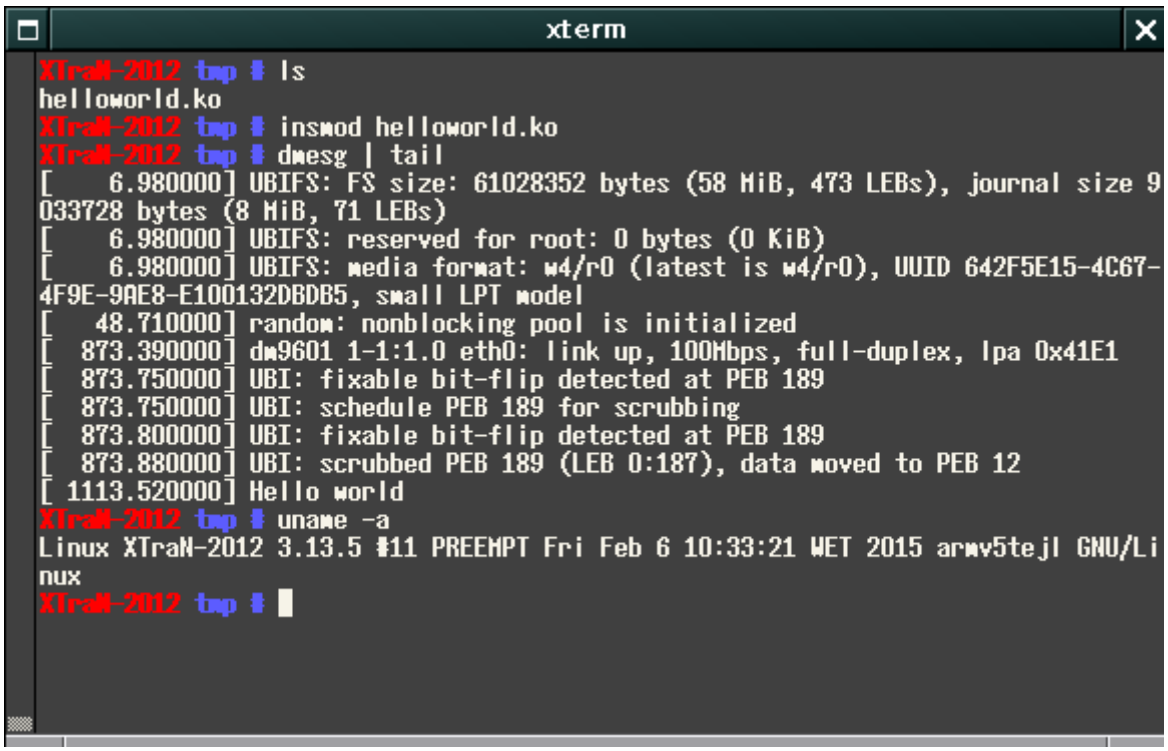

Figura 4.5: Carregamento do componente *Hello World*

4.2 Teste *Acc-driver*

O módulo *Acc-driver* foi copiado e carregado, conforme ilustrado pelas Figuras 4.6 e 4.7. Após o carregamento foi possível visualizar o conjunto de atributos criados pelo módulo, ilustrado pela Figura 4.8, na pasta */sys*, que permite que se efetue a comunicação com o dispositivo *FXOS8700CQ*.

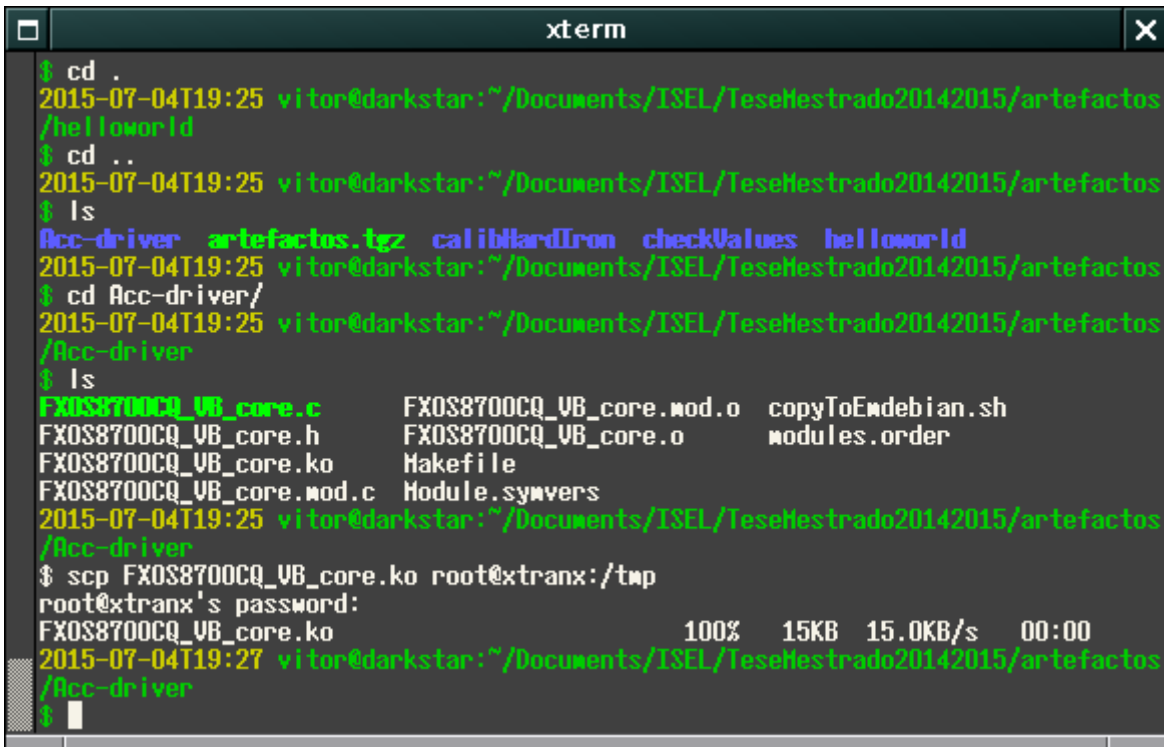

Figura 4.6: Cópia do componente *Acc-driver*

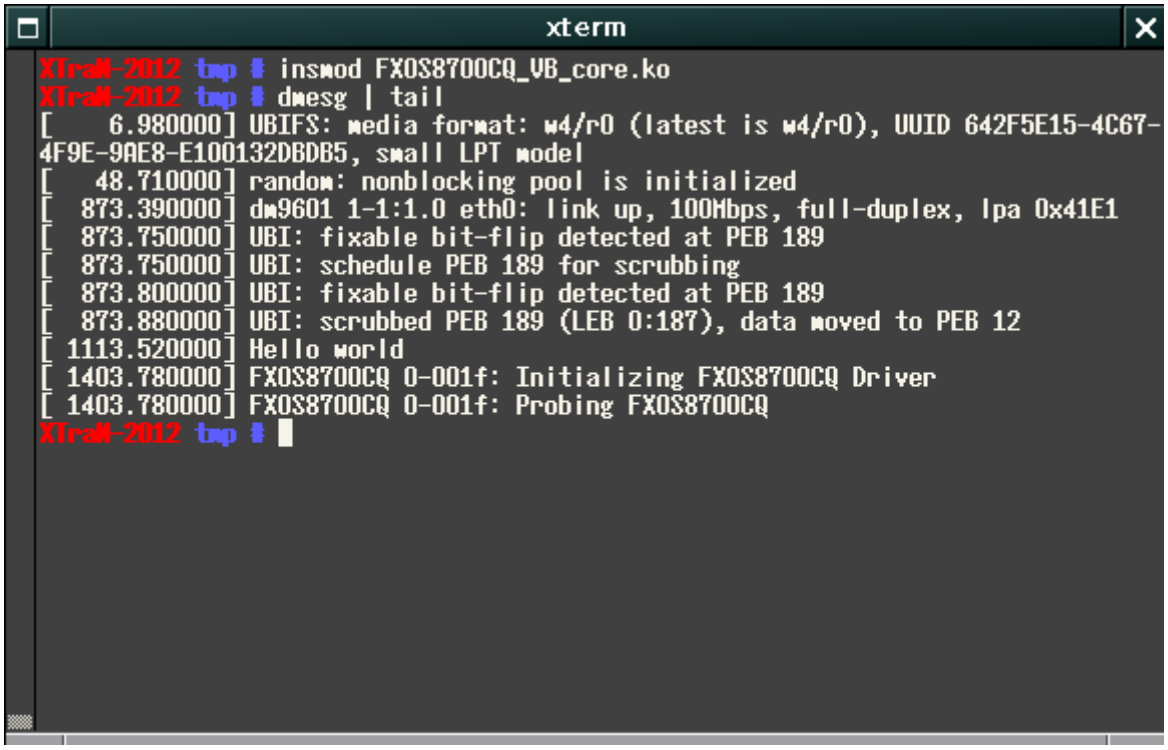

Figura 4.7: Carregamento do componente *Acc-driver*

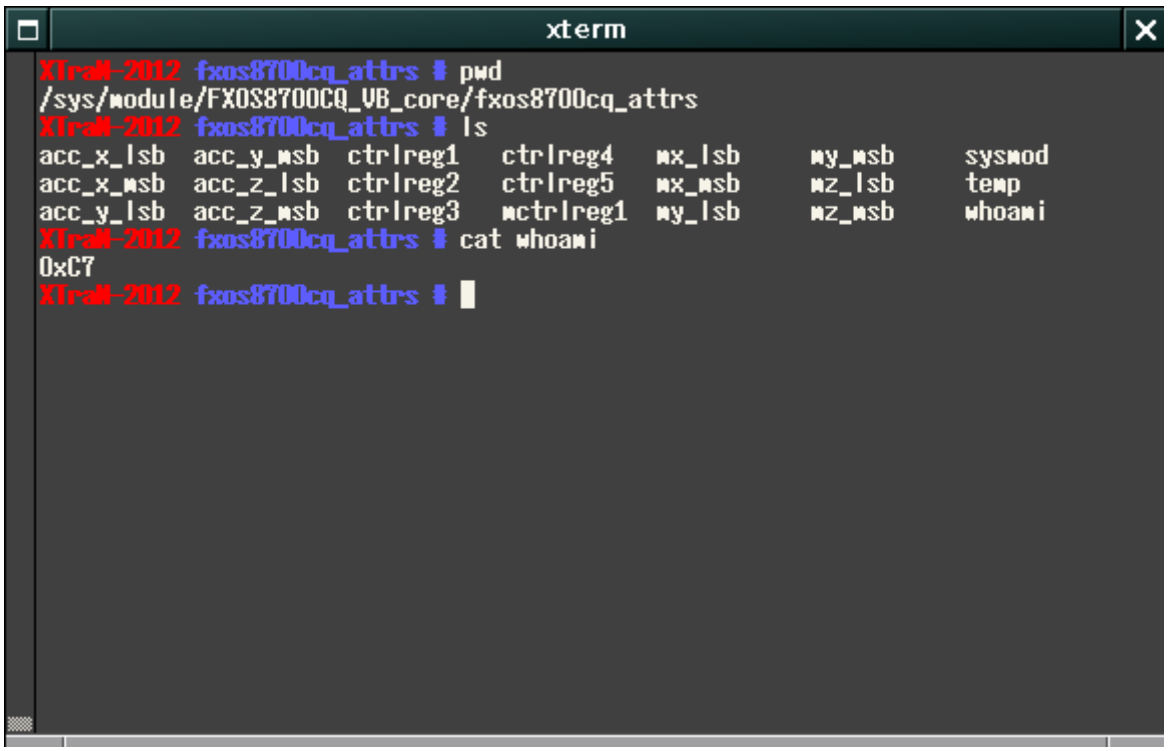

Figura 4.8: Atributos *sysfs* do componente *Acc-driver*

4.3 Teste *checkValues*

O programa *checkValues* foi copiado e executado no computador embarcado, sem lhe ter sido definido valores de calibração, conforme ilustrado nas Figuras 4.9 e 4.10. Os valores de calibração utilizados foram *0 0 0*. É de notar que quando o programa *checkValues* é executado sem valores de calibração para colmatar o efeito de *hard-iron*, o dispositivo *FXOS8700CQ* devolve valores extremamente altos para as componentes MY e MZ do vector de campo magnético. Foi devido à identificação desse problema que se decidiu no âmbito do trabalho de projeto, desenvolver o componente *calibHardIron*, para se obterem valores corretos provenientes do sensor magnetómetro.

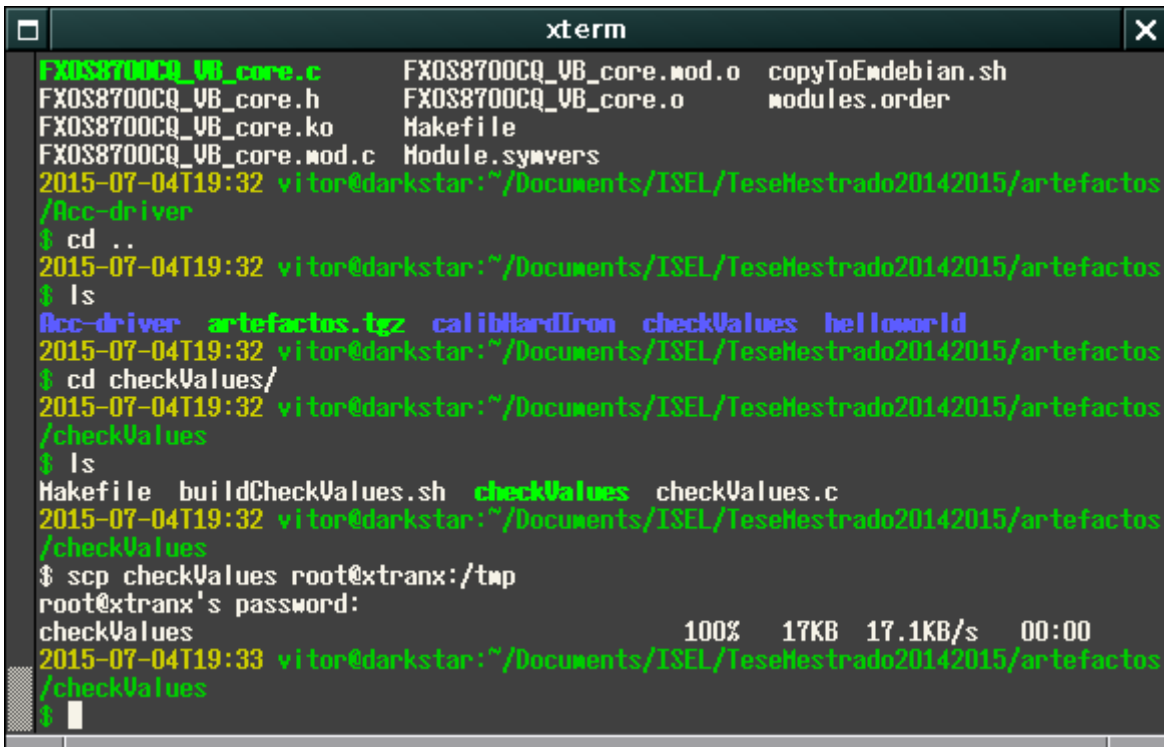

Figura 4.9: Cópia do programa *checkValues*

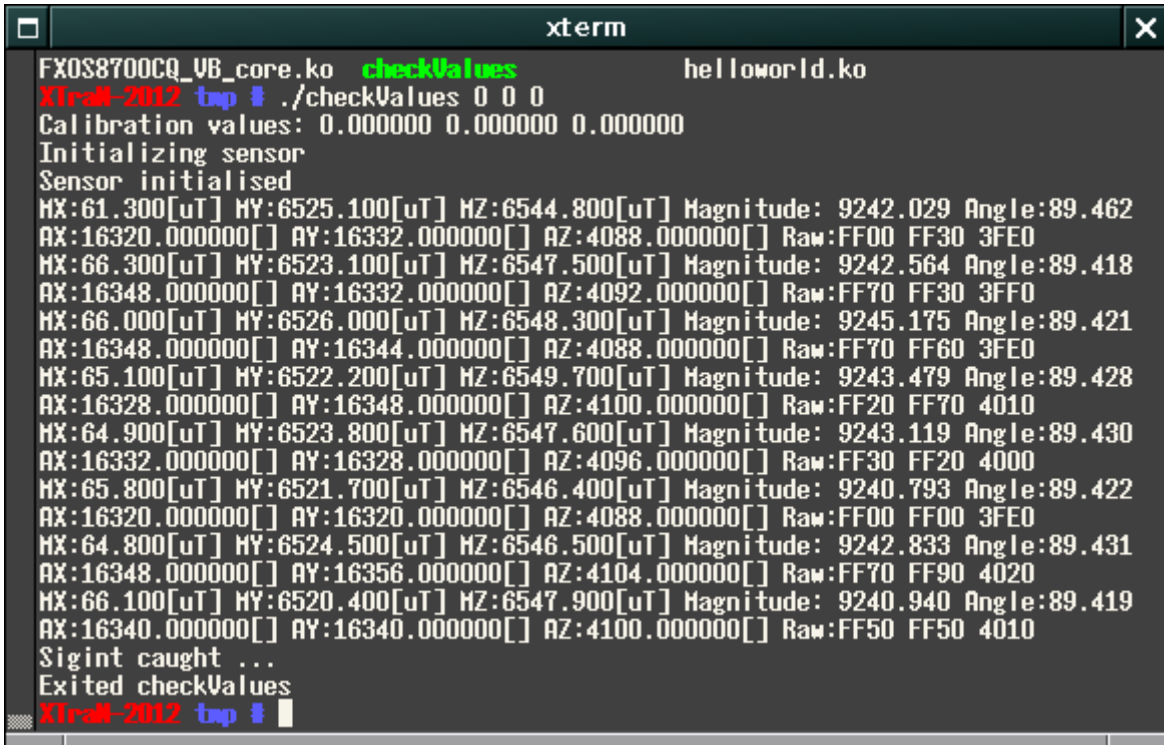

Figura 4.10: Execução do programa *checkValues* sem valores de calibração

4.4 Teste *calibHardIron*

O programa *calibHardIron* foi copiado e executado no computador embarcado, de forma a se obterem valores de calibração. Em simultâneo foi copiado o ficheiro */tmp/mxmymz.samples* que contêm amostras obtidas anteriormente para o computador embarcado utilizado. Para se obterem as amostras existentes em */tmp/mxmymz.samples* o computador embarcado foi rodado sobre si próprio enquanto o programa *checkValues* corria com valores de calibração *0 0 0*. A cópia encontra-se ilustrada pela Figura 4.11. O resultado dos valores de calibração encontram-se ilustrados pelas Figuras 4.12 e 4.13. O vector de calibração obtido encontra-se na Figura 4.13 e têm os valores *V <45.920 6519.030 6507.045>* . Após a obtenção dos valores de calibração, executou-se de novo o programa *checkValues* com os valores de calibração obtidos. Enquanto o programa estava a ser executado, movimentou-se o computador embarcado sobre si próprio, de forma a se ver o parâmetro *Angle* a reagir à nova orientação do computador embarcado, como ilustrado pela Figura 4.14. O parâmetro *Angle* corresponde ao ângulo existente para com o ponto cardeal Norte. Para se verificar que o ângulo devolvido era correcto, o valor era comparado em simultâneo com o de uma bússola que se encontrava alinhada com o computador embarcado, ilustrado pela Figura 4.15.

Verificou-se que com o computador embarcado estável numa orientação, o valor do ângulo devolvido oscilava por volta de 10 graus sobre a direção sobre a qual estava apontado. Isso significa que existirá um erro acumulado ao longo do tempo baseada nessa variação nas estimativas criadas caso fosse utilizado o computador embarcado testado num veículo de um cliente da Tecmic. Mas é de notar que é muito provável que o vector de calibração possa variar de computador embarcado, pelo que o vector de calibração calculado pode ser diferente de computador embarcado para computador embarcado.

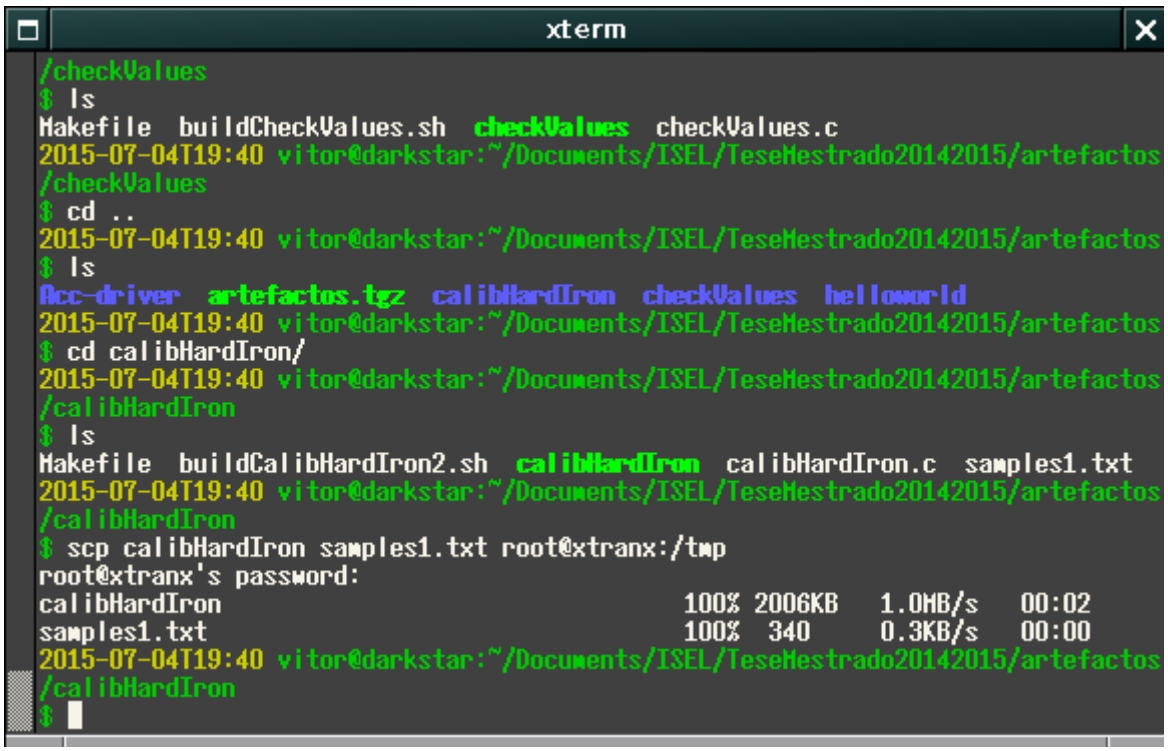

Figura 4.11: Cópia do programa *calibHardIron*

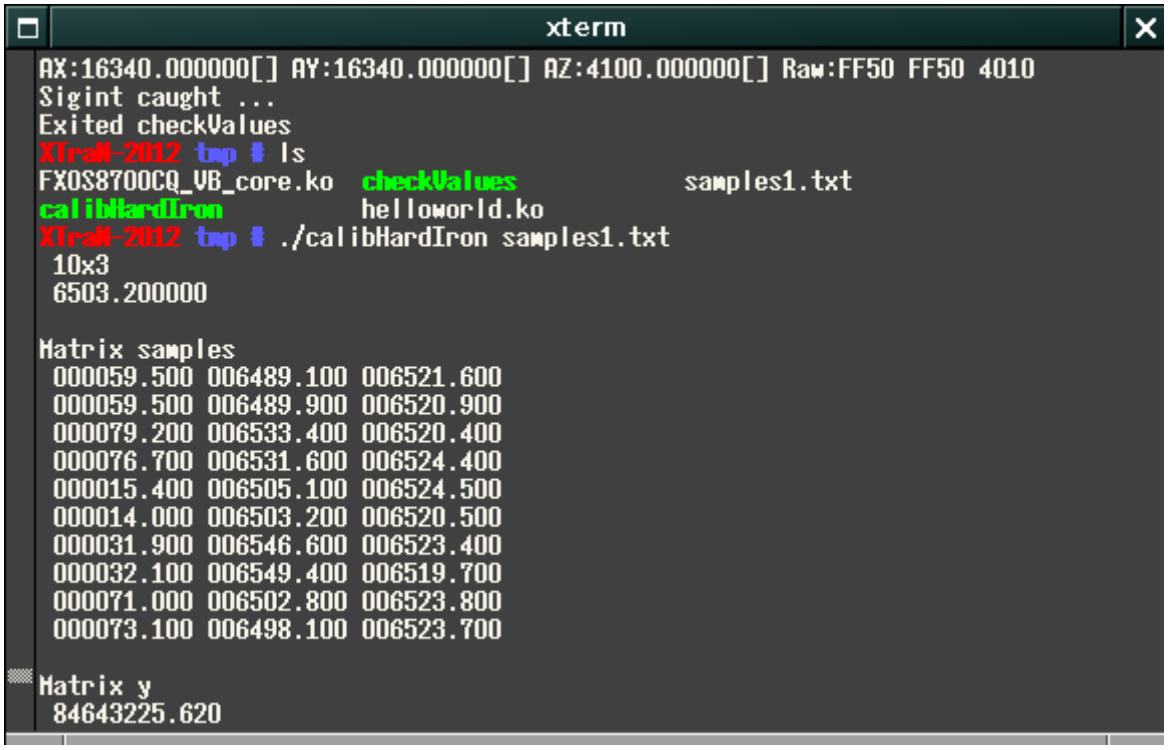

Figura 4.12: Inicio da execução do programa *calibHardIron*

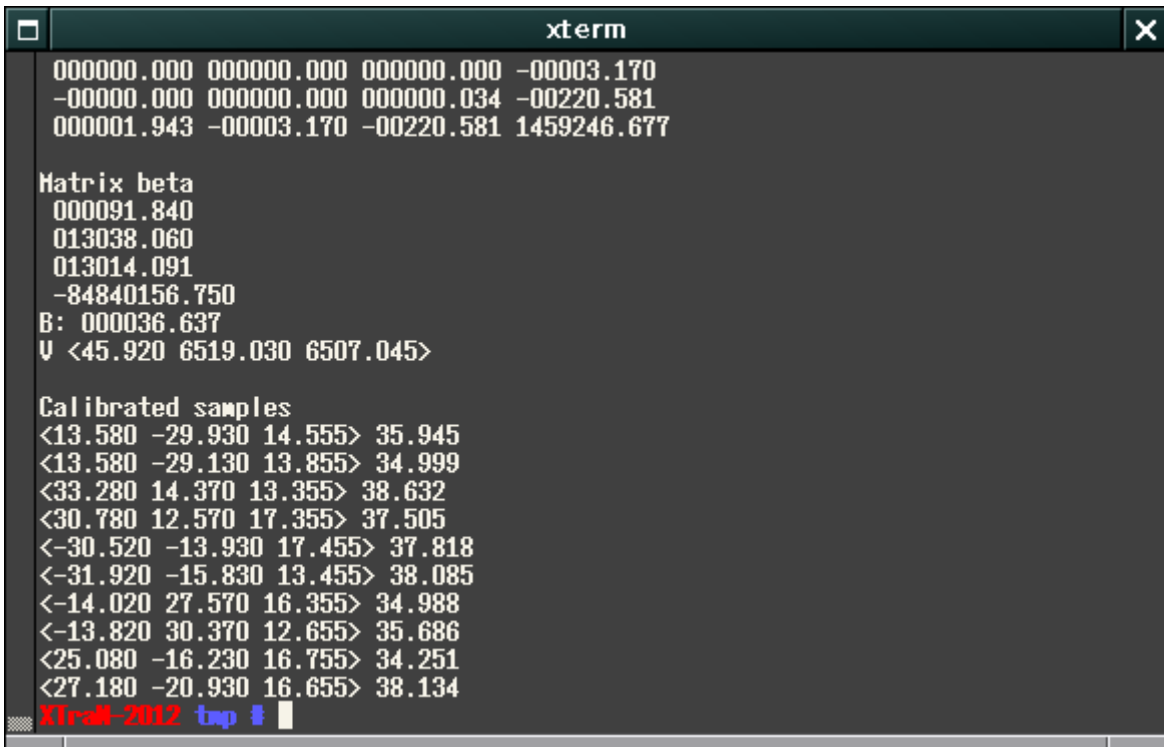

Figura 4.13: Fim da execução do programa *calibHardIron*

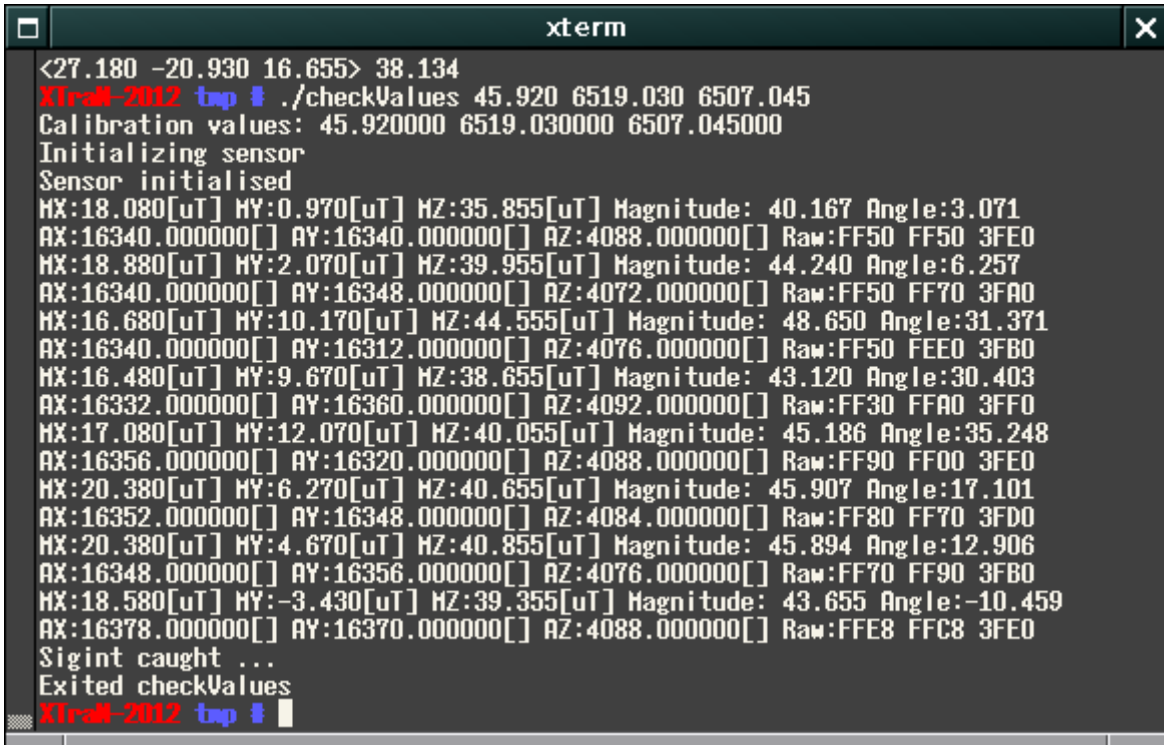

Figura 4.14: Execução do programa *checkValues* com valores de calibração

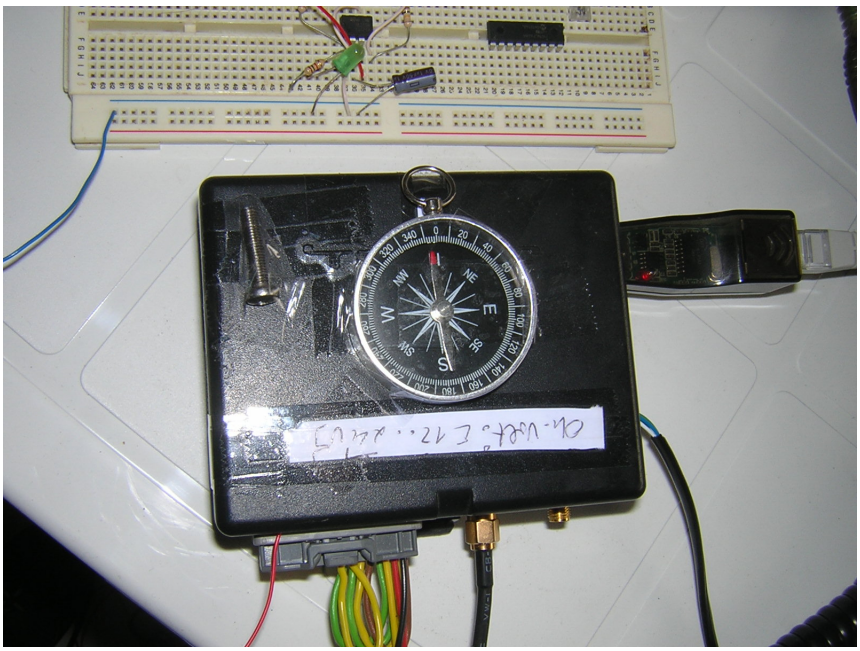

Figura 4.15: Computador embarcado com bússola real para comparar com os valores obtidos pelo programa *checkValues*

4.5 Teste *estimator*

O programa *estimator* foi testado com os valores de calibração obtidos com o programa *calibHardIron*. Ele inicializou corretamente o GPS, o dispositivo *FXOS8700CQ* e o odómetro. Foi capaz de obter a posição GPS a partir das mensagens recebidas NMEA $GPGGA, ler o valor de distância percorrida do odómetro, obter o ângulo do sensor magnetómetro e de gerar estimativas com base nas amostras obtidas. A Figura 4.16 mostra um conjunto de amostras obtidas durante a execução do teste. Na Figura 4.17 são mostrados os dados em JSON degradado gerados com as estimativas. Espera-se que existam erros devido aos erros fornecidos pelos dispositivos utilizados, que são acumulados ao longo do tempo, pois a navegação estimada ou DR está sujeita a erros.

Para se testar que o odómetro era capaz de ler dados dos ficheiros *in_rot0_raw* e *in_rot0_calibscale*, ligou-se ao pino que recebe os impulsos de contagem à saída de um circuito oscilador construído a partir do *chip 555* [46]. O circuito gerava uma onda quadrada de baixa frequência para que se fosse capaz de verificar que o ficheiro *in_rot0_raw* estava a contar os impulsos recebidos. O circuito e a ligação encontram-se ilustrados pela Figura 4.18. O esquemático do circuito encontra-se ilustrado pela Figura 4.19. A frequência de oscilação é determinada pela fórmula (4.1), que tem como variáveis os valores das resistências R_a e R_b, e o valor do condensador C. O circuito ilustrado na Figura 4.19 tinha como alvo gerar uma frequência de cerca de 4,8 [Hz].

$$freqOscilacao = \frac{1,44}{(R_a + 2R_b) \cdot C} \tag{4.1}$$

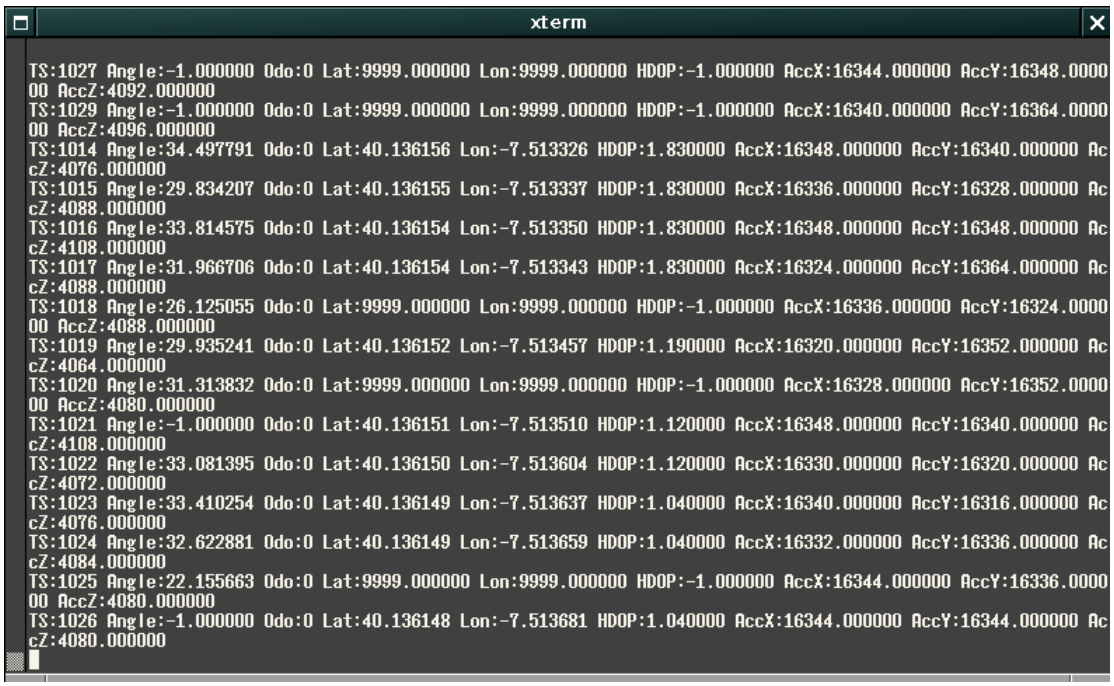

Figura 4.16: *Array* circular a ser mostrado pelo *estimator* em execução

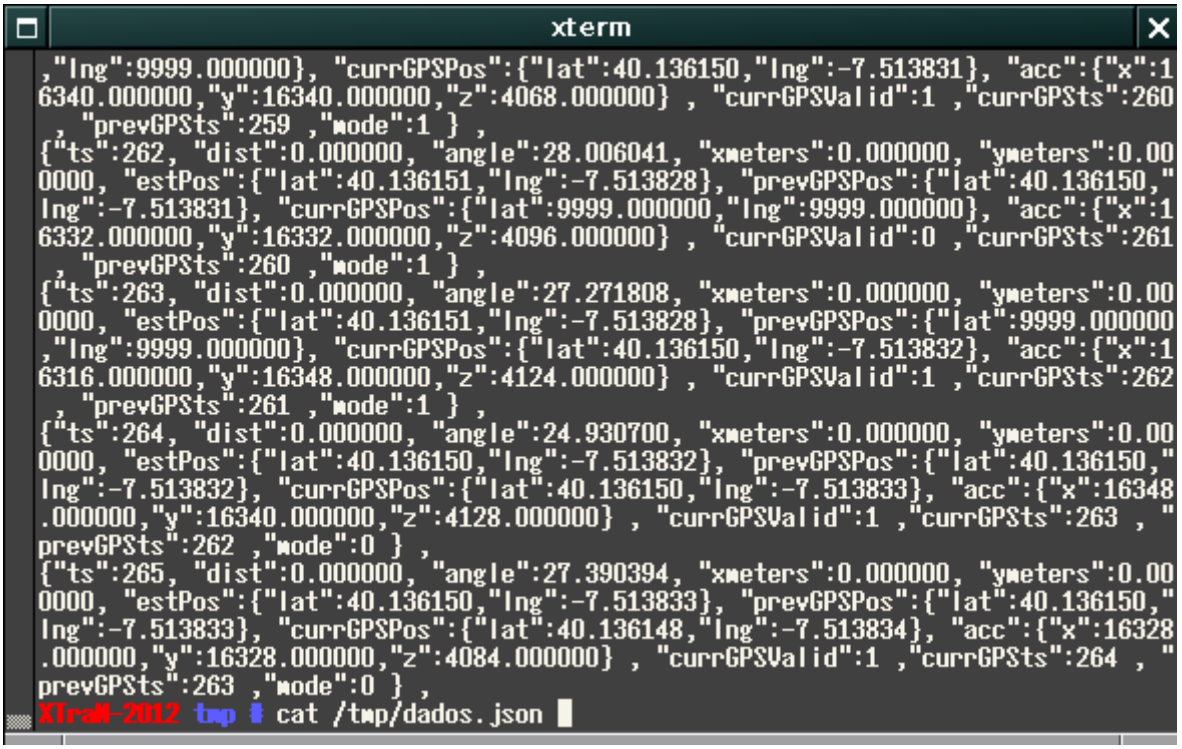

Figura 4.17: Ficheiro de estimativas JSON no computador embarcado

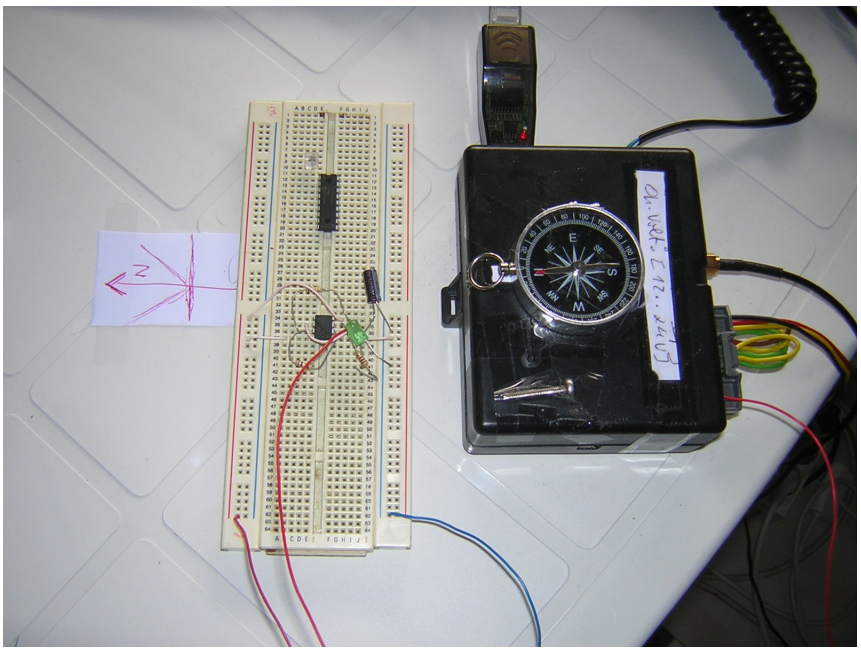

Figura 4.18: Circuito oscilador *555* ligado ao pino que recebe impulsos no computador embarcado

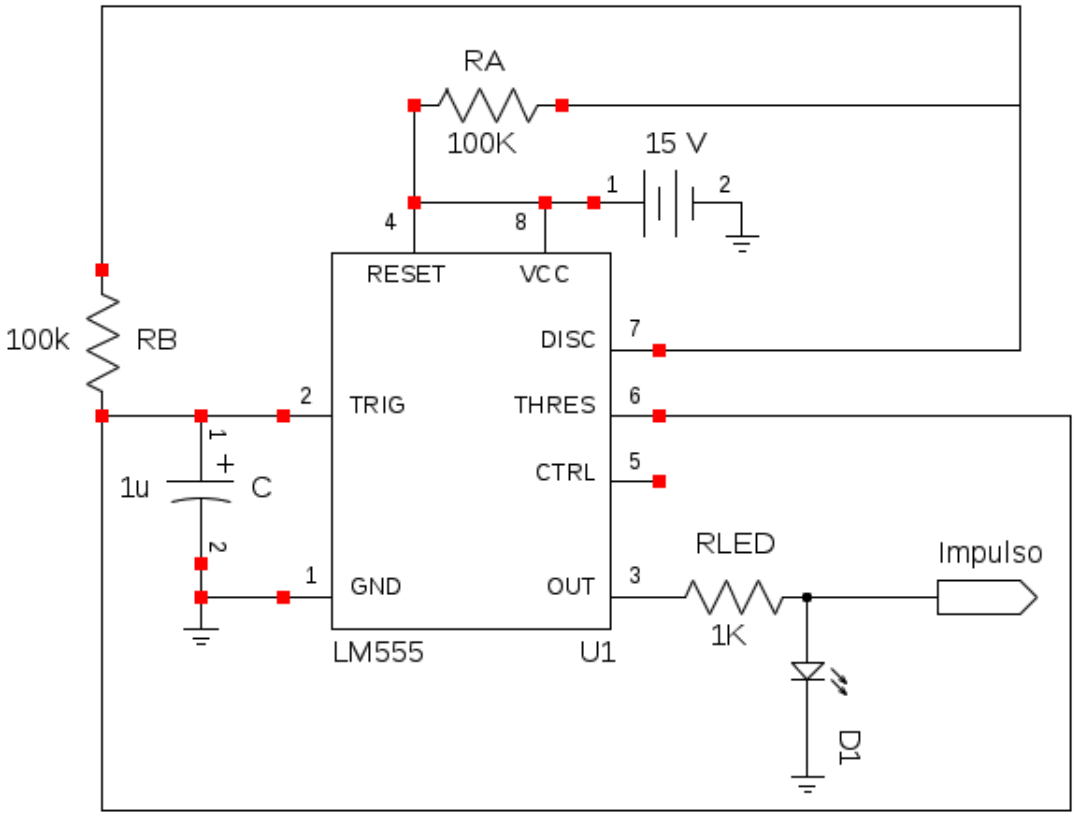

Figura 4.19: Esquemático do circuito oscilador *555*

4.5.1 Trajeto virtual no túnel da Gardunha

Para se simular o algoritmo de estimação num trajeto longo sem sinal GPS, adquiriram-se pontos de geo-referência que atravessam o túnel de Alpedrinha e túnel da Gardunha, próximos do concelho do Fundão, no sentido Alpedrinha-Fundão, a partir do *Google Maps*. O total de pontos adquiridos foram 151. O trajeto encontra-se ilustrado pela Figura 4.20.

A Figura 4.21 mostra o inicio e fim do túnel de Alpedrinha. A Figura 4.22 mostra a entrada do túnel da Gardunha e a Figura 4.23 mostra a saída do túnel da Gardunha. As vias a laranja mostram locais onde é possível ter acesso a sinal GPS, e a cinzento estão as vias que correspondem aos túneis. Os pontos adquiridos foram numerados de 1 a 151 para todo o trajeto. Ao intervalo de pontos de 20 a 34 correspondem os pontos marcados sobre o túnel de Alpedrinha, e ao intervalo de 61 a 141 correspondem os pontos marcados para o túnel da Gardunha.

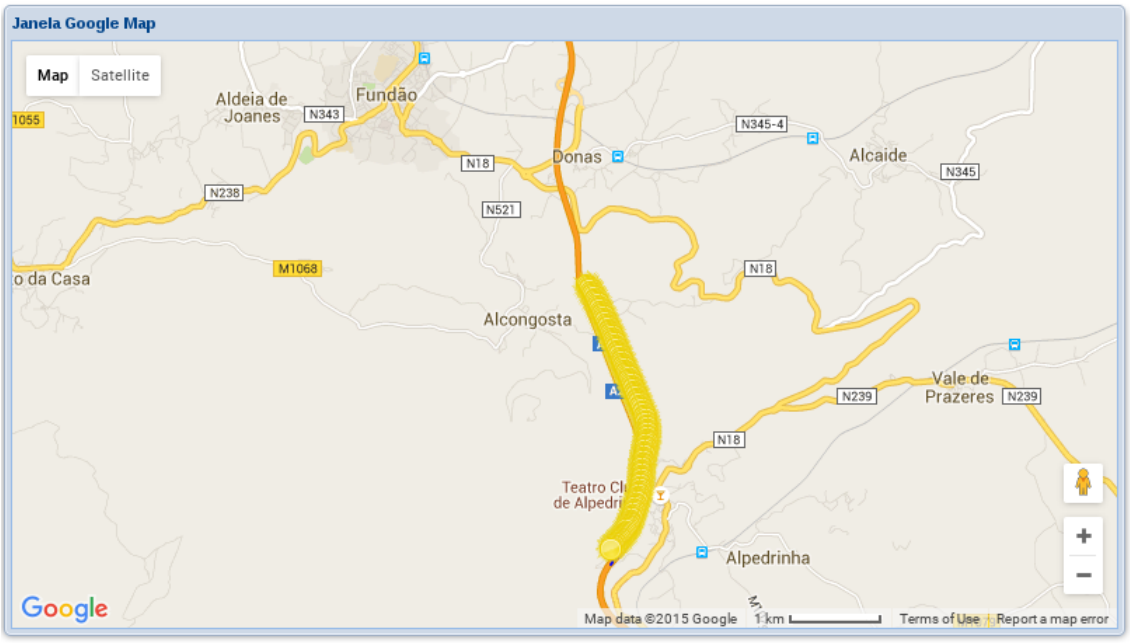

Figura 4.20: Pontos adquiridos para o túnel da Gardunha

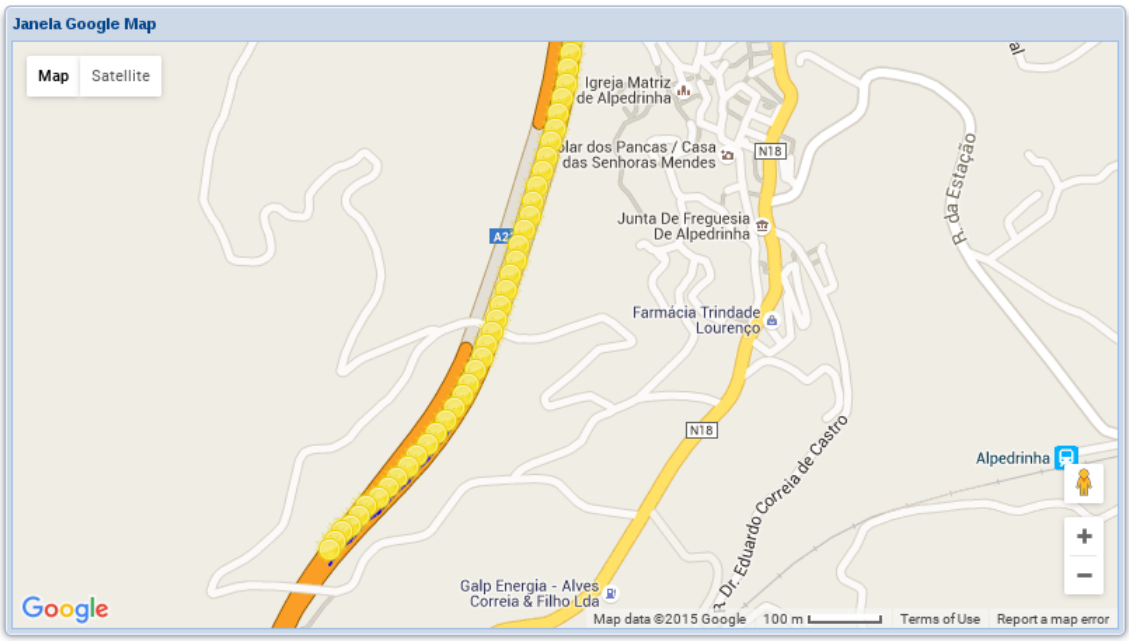

Figura 4.21: Entrada e saída do túnel de Alpedrinha

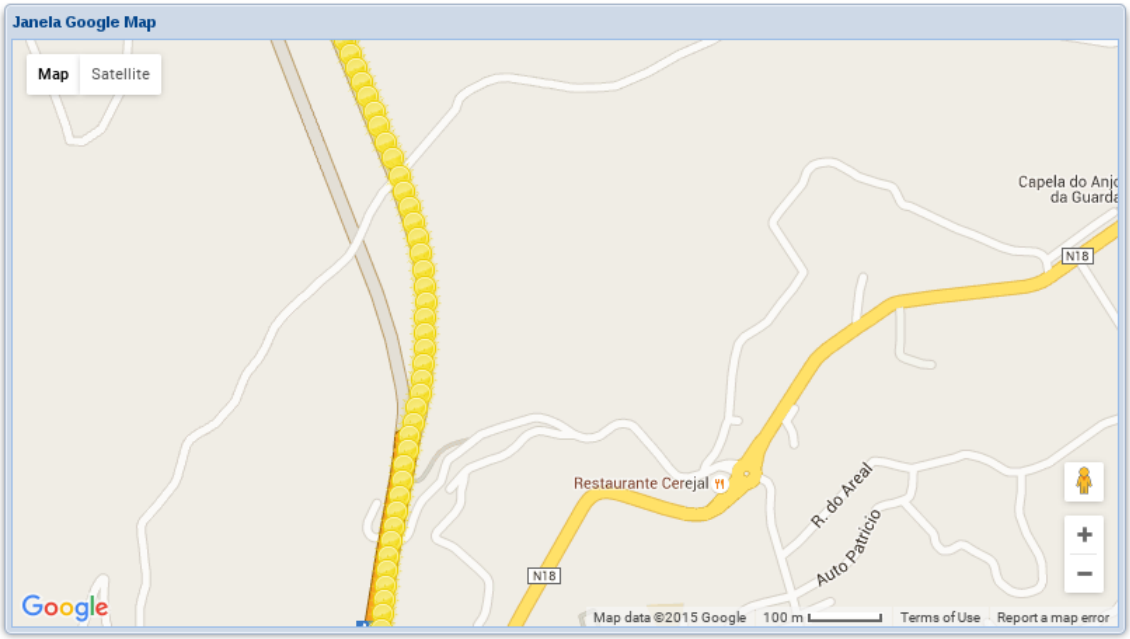

Figura 4.22: Entrada do túnel da Gardunha

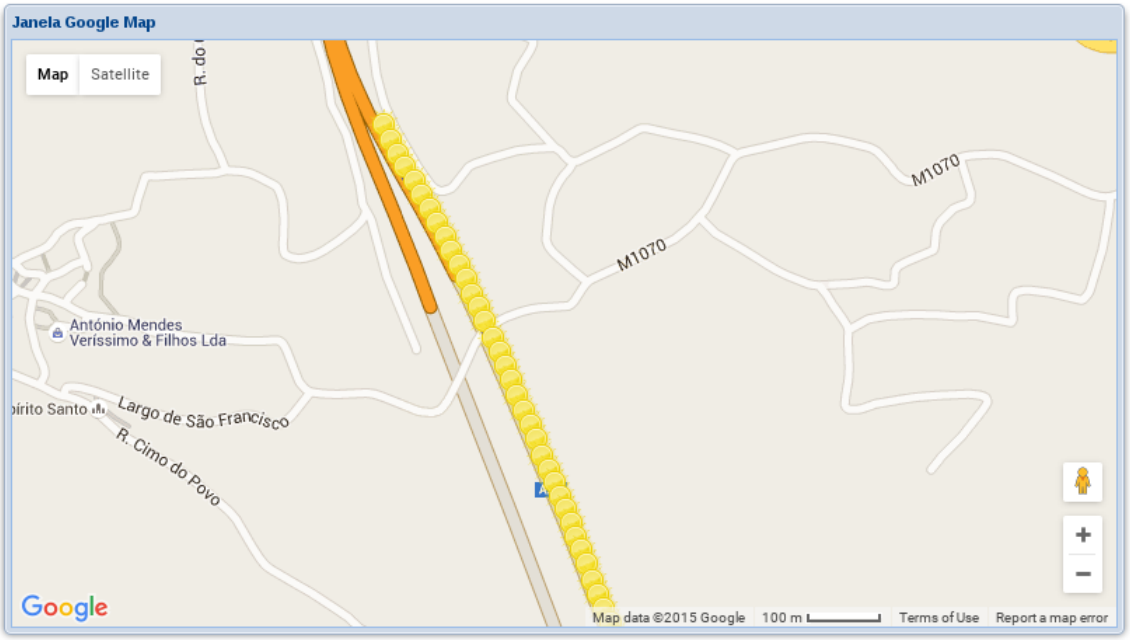

Figura 4.23: Saída do túnel da Gardunha

Foi criado um *script* Python chamado *pontosTunelGardunha.py* de forma a se obterem ângulos de magnetómetro e distâncias de odómetro. O *script* após a transformação dos dados simula o programa *estimator*, e define que para os intervalos de pontos de 20 a 34 (túnel de Alpedrinha), e de 61 a 141 (túnel da Gardunha) não há sinal GPS, ou seja, define um HDOP superior a 5. Foi definida uma resolução máxima de 10 graus para o ângulo para os dados transformados, ou seja, são gerados graus em intervalos de 10 graus. A escolha do valor 10 está relacionado com o valor de ângulo devolvido oscilar por volta de 10 graus, como indicado na secção 4.4. O túnel de Alpedrinha tem cerca de 280 metros de comprimento, e o túnel da Gardunha tem cerca de 1620 metros de comprimento [47].

A estimativa na saída do túnel de Alpedrinha encontra-se ilustrada pela Figura 4.24. Verifica-se um desvio quando comparado ao túnel a cinzento dos vários pontos, até à transição do ponto 36 para o 37. A distância entre o ponto 36 real e o ponto 36 estimado é de 41 metros. Devido à extensão do túnel ser cerca de 280 metros, tem-se um erro de 0,146 metros por cada metro percorrido.

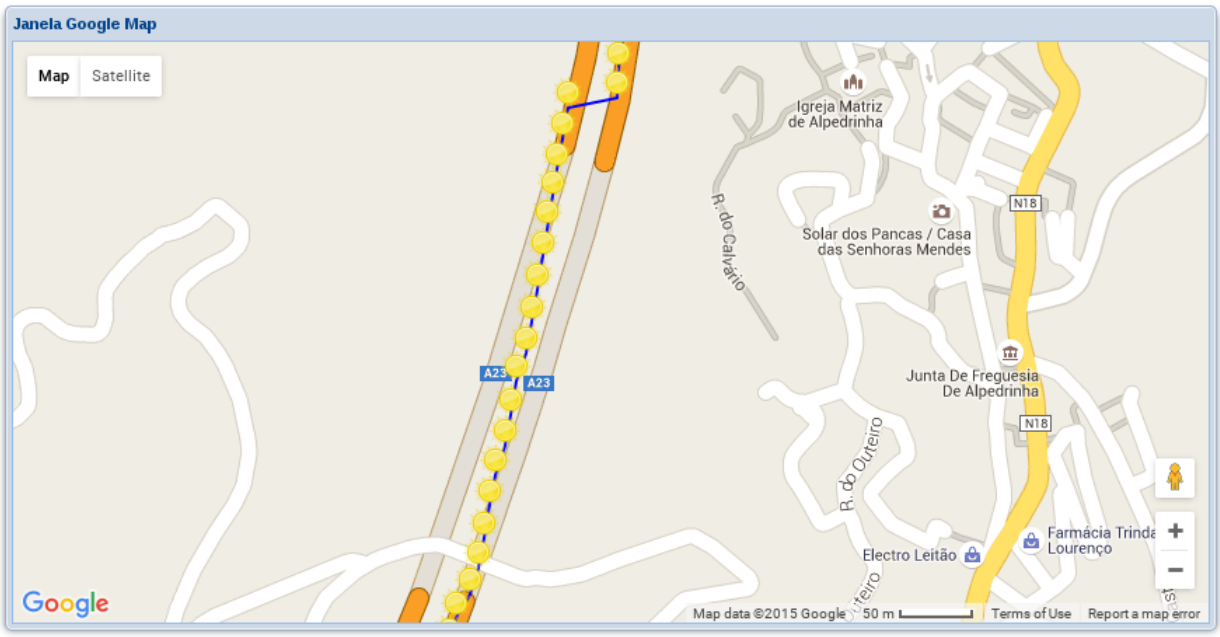

Figura 4.24: Estimativa na saída do túnel de Alpedrinha

Para o túnel da Gardunha a estimativa na saída do túnel encontra-se ilustrada pela Figura 4.25. Verifica-se um desvio quando comparado ao túnel a cinzento dos vários pontos, até à transição do ponto 142 para o 143. A distância entre o ponto 142 real e o ponto 142 estimado é de 162 metros. Devido à extensão do túnel ser cerca de 1620 metros, tem-se um erro de 0,1 metros por cada metro percorrido. A Figura 4.26 mostra uma visão de mais alto nível da estimativa para o túnel da Gardunha.

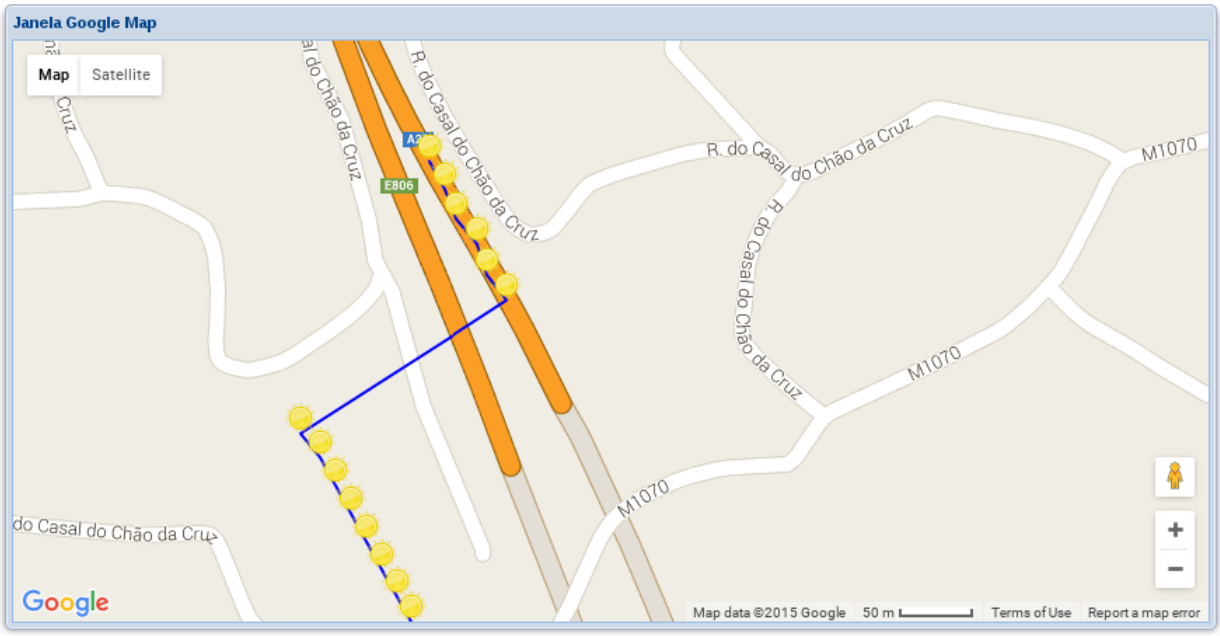

Figura 4.25: Estimativa na saída do túnel da Gardunha

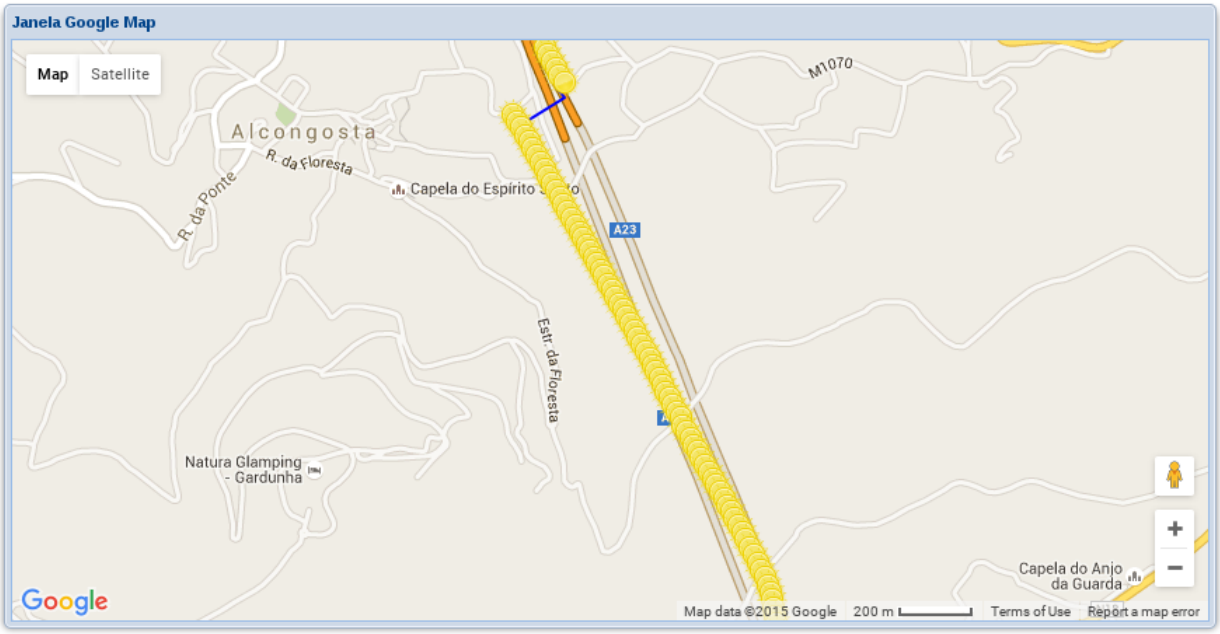

Figura 4.26: Visão de mais alto nível da estimativa do túnel da Gardunha

4.6 Teste *getPoints*

A aplicação *web getPoints* é capaz de obter pontos selecionados diretamente no *Google Maps*, mostra os pontos obtidos em formato JSON, e se for fornecido um objeto JSON com a estrutura apropriada, carrega os pontos correspondentes no *Google Maps*. Atualmente a aplicação *web* encontra-se alojada em [48].

Para se efetuarem os testes com a aplicação foram gerados vários trajetos virtuais sobre o *Google Maps*. De seguida utilizou-se a funcionalidade *Mostrar pontos em JSON* para guardar o trajeto criado, copiado o texto representativo do trajeto em JSON para o *Clipboard*. De seguida clicou-se em *Limpar Pontos*, e verificou-se que os pontos selecionados anteriormente foram limpos. Após esse passo carregaram-se os pontos anteriores utilizando a funcionalidade *Carregar pontos em JSON*, e visualizaram-se os pontos selecionados no trajeto virtual inicial. A funcionalidade *Carregar pontos em JSON* também permite visualizar os dados JSON degradados criados durante a execução do programa *estimator*.

A Figura 4.28 mostra um conjunto de vectores hipotéticos. Na Figura 4.28 o ícone do Sol corresponde a uma posição obtida por GPS, e o ícone de uma lâmpada corresponde a uma estimativa de posição. Os pontos obtidos via GPS são ligados por uma linha azul, e os pontos correspondentes a uma estimativa de posição são ligados por uma linha verde.

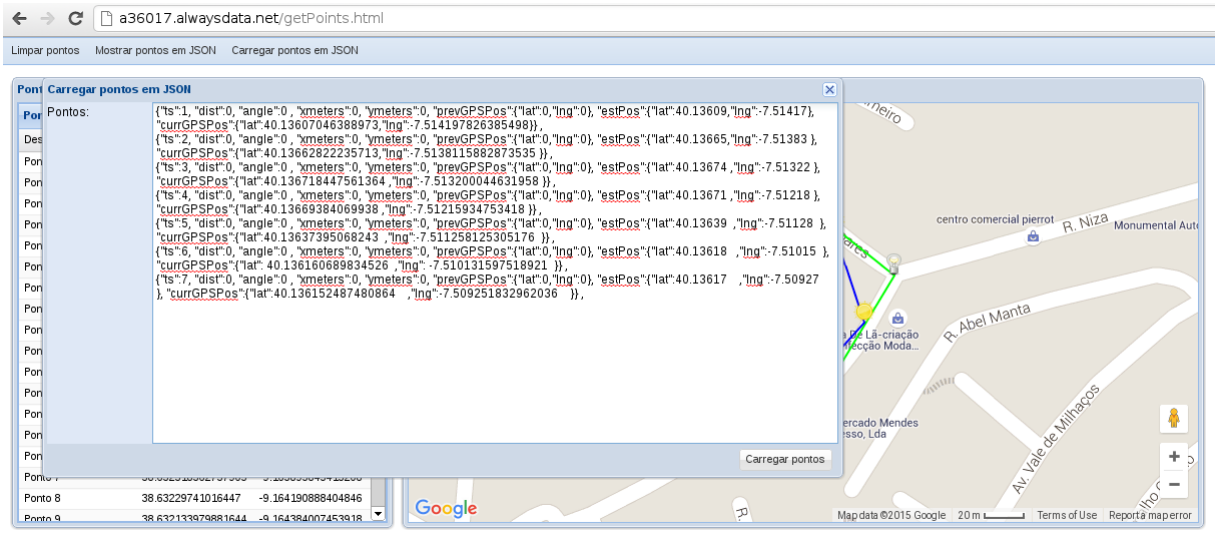

Figura 4.27: Carregamento de estimativas, adaptado de [48]

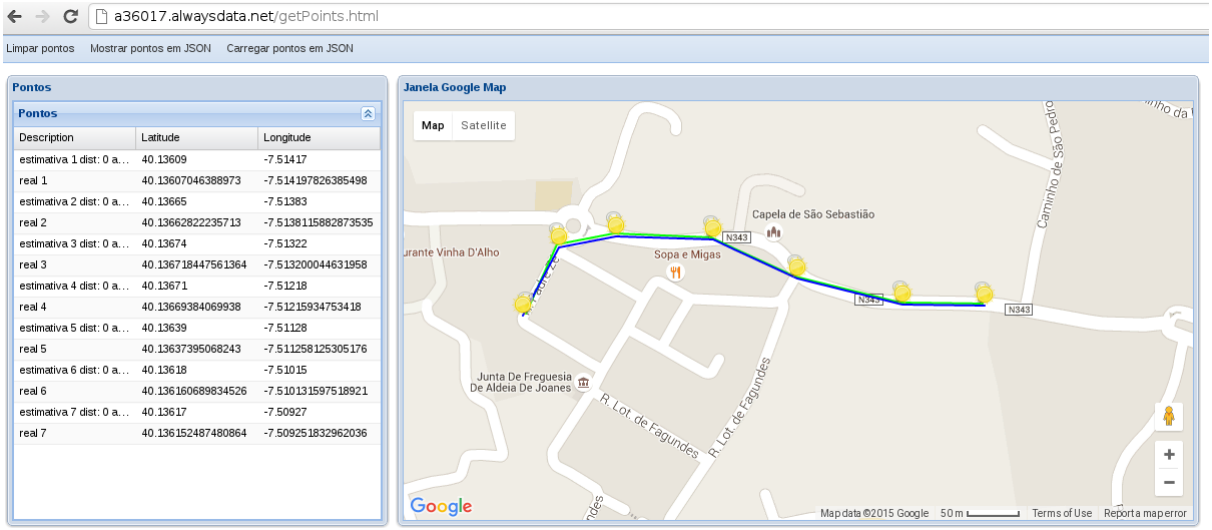

Figura 4.28: Mapa com estimativas de posição geo-referenciada, adaptado de [48]

5 Conclusões

O presente capítulo indica a que conclusões se chegaram após o desenvolvimento do sistema de navegação inercial e que trabalho futuro poderá ser feito de forma a se melhorar o sistema de navegação inercial.

Durante o decorrer do trabalho foi estudado o sistema *XtranX Passenger*, e as tecnologias utilizadas nesse sistema, de forma a se elaborar uma solução para o problema existente, que é o de caso os computadores de bordo da Tecmic percam sinal GPS, quando entram por exemplo num túnel, eles não têm a capacidade de indicar a sua posição geo-referenciada aproximada, com o auxílio de informação proveniente de outros sensores existentes.

O sistema de navegação inercial complementa o *XtranX Passenger* para que ele tenha a capacidade de utilizar navegação estimada ou DR para além de navegação electrónica com base no receptor GPS. O sistema de navegação inercial implementado é capaz de utilizar valores de posição via GPS, e valores de odómetro em simultâneo com valores dos sensores acelerómetro e magnetómetro. Com base nesse valores e no algoritmo de estimação criado, é capaz de gerar estimativas de posição geo-referenciada. Os componentes *calibHardIron* e *checkValues* permitem que se efetue a calibração do sensor magnetómetro para que funcione como bússola electrónica. O componente *estimator* gera informação em formato JSON, periodicamente, utilizando o receptor GPS, o odómetro e o magnetómetro, com a informação, apresentada na Tabela 3.13. A aplicação *web getPoints* permite visualizar as estimativas e os pontos reais em simultâneo. Os componentes do sistema de navegação inercial desenvolvidos já foram fornecidos à Tecmic.

De momento a Tecmic ainda não teve a oportunidade de testar o componente *estimator* em computadores embarcados de clientes seus, de forma a se obterem valores reais, e se comprovar se as estimativas do algoritmo são válidas, quando comparadas com os valores GPS obtidos em paralelo no momento de criação da estimativa. Aguarda-se a qualquer momento resposta da Tecmic com estimativas geradas em computadores embarcados de clientes para se verificar as mesmas.

5.1 Trabalho futuro

O sistema de navegação inercial tem muitos pontos de melhoria, como por exemplo:

- Analisar as estimativas obtidas de computadores embarcados e rever o algoritmo de estimação de forma a colmatar e corrigir potenciais falhas existentes no mesmo que atualmente não foram previstas.
- O módulo *kernel* pode ser alterado de forma a seguir a localização de dispositivos IIO (*Indus-*

trial Input Output) na árvore do sistema de ficheiros *sysfs*.

Com o sistema de navegação inercial a gerar estimativas aproximadas é possível utilizar o mesmo em vários cenários onde a perca de sinal GPS ocorra frequentemente, como por exemplo em túneis, e outros locais onde existam problemas com GPS. O túnel da Gardunha seria um bom cenário para gerar estimativas devido à sua extensão de cerca de 1620 metros. Outro túnel seria também, o túnel da Avenida João XXI em Lisboa, com cerca de 1490 metros.

O algoritmo do *estimator* pode ser adaptado para funcionar em dispositivos móveis como *Smartphones* Android, iPhone, ou outros, desde que existam componentes que sejam capazes de comunicar a partir de protocolos sem fios como por exemplo WiFi, Bluetooth e ZigBee, e que forneçam dados sobre o campo magnético (magnetómetro) e sobre a distância percorrida (odómetro). Atualmente existem já em grande parte dos *Smartphones* dispositivos receptores GPS. Também pode existir o caso em que um dispositivo com um sensor magnetómetro exista no *Smartphone*, e nesse caso conforme o tipo de *Smartphone* é necessário estudar as API (*Application Programming Interface*) existentes para o *SmartPhone* em questão para se compreender como comunicar com o magnetómetro existente. O dispositivo móvel tem a vantagem de ter a capacidade de mostrar mapas no seu ecrã, e se possível apresentar em tempo real uma correção com base nos mapas existentes da estimativa gerada.

Bibliografia

[1] Sitio da empresa Tecmic, Junho 2015
http://www.tecmic.pt/

[2] Definição de GPS, Junho 2015
http://pt.wikipedia.org/wiki/Sistema_de_posicionamento_global

[3] Áreas de atuação da Tecmic, Junho 2015
http://www.tecmic.pt/tecmic/quem-somos/

[4] Definição de ASIC, Junho 2015
http://pt.wikipedia.org/wiki/ASIC

[5] Sistema *XtranX Passenger*, Junho 2015
http://www.tecmic.pt/pt-br/portfolio/xtran-passenger/

[6] Imagem do computador de bordo, Junho 2015
http://www.tecmic.pt/wp-content/uploads/2014/01/Tecmic-Equipamento-10.png

[7] Sistema operativo Linux, Junho 2015
http://pt.wikipedia.org/wiki/Linux

[8] Como desenvolver para o Kernel Linux, Junho 2015
https://www.kernel.org/doc/Documentation/HOWTO

[9] Kernel Linux 3.13.5, Junho 2015
https://www.kernel.org/pub/linux/kernel/v3.0/linux-3.13.5.tar.xz

[10] Definição de navegação, Setembro 2015
https://pt.wikipedia.org/wiki/Navega%C3%A7%C3%A3o

[11] Definição de navegação electrónica, Setembro 2015
https://pt.wikipedia.org/wiki/Navega%C3%A7%C3%A3o_eletr%C3%B4nica

[12] Definição de navegação visual, Setembro 2015
https://pt.wikipedia.org/wiki/Navega%C3%A7%C3%A3o_visual

[13] Definição de navegação astronómica, Setembro 2015
https://pt.wikipedia.org/wiki/Navega%C3%A7%C3%A3o_astron%C3%B4mica

[14] Definição de estrela Polar, Setembro 2015
https://pt.wikipedia.org/wiki/Polaris

[15] Definição de navegação estimada, Setembro 2015
https://pt.wikipedia.org/wiki/Navega%C3%A7%C3%A3o_estimada

[16] Definição de *dead reckoning*, Setembro 2015
https://en.wikipedia.org/wiki/Dead_reckoning

[17] Definição de Sistema de navegação inercial, Setembro 2015
https://en.wikipedia.org/wiki/Inertial_navigation_system

[18] Sistema de navegação inercial LN3, Setembro 2015
https://en.wikipedia.org/wiki/LN-3_Inertial_Navigation_System

[19] Processador ARM AT91SAM9260, Junho 2015

http://www.atmel.com/devices/SAM9260.aspx

[20] Sensor acelerómetro e magnetómetro FXOS8700CQ, Junho 2015
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FXOS8700CQ

[21] Atmel, SAM9260 Summary, Junho 2015
http://www.atmel.com/Images/6221s.pdfSAM9260Summary

[22] Imagem do microcontrolador AT91SAM9260, Junho 2015
http://cn.newmaker.com/pro_75774.html

[23] Bus I2C, Setembro 2015
https://en.wikipedia.org/wiki/I%C2%B2C

[24] Wikipedia, Earth Magnetic Field, Setembro 2015
https://en.wikipedia.org/wiki/Earth%27s_magnetic_field

[25] Imagem do sensor FXOS87O00CQ, Junho 2015
http://www.silica.com/fileadmin/02_Products/Productdetails/Freescale/
Silica_Freescale_FXOS8700CQ-icon.jpg

[26] *Data sheet* do sensor FXOS8700CQ, Junho 2015
http://cache.freescale.com/files/sensors/doc/data_sheet/FXOS8700CQ.pdf?
fpsp=1

[27] Imagem do sistema de coordenadas para representar campos magnéticos na Terra, Setembro 2015
https://en.wikipedia.org/wiki/Earth%27s_magnetic_field#/media/File:
XYZ-DIS_magnetic_field_coordinates.svg

[28] Robert Love, "Linux Kernel Development", Third edition, Addison-Wesley

[29] Função ioctl, Setembro 2015
http://linux.die.net/man/2/ioctl

[30] Protocolo NMEA, Setembro 2015
https://en.wikipedia.org/wiki/NMEA_0183

[31] Mensagens NMEA, Setembro 2015
http://aprs.gids.nl/nmea/

[32] HDOP, Setembro 2015
https://en.wikipedia.org/wiki/Dilution_of_precision_%28GPS%29

[33] Sistema de coordenadas latitude e longitude, Setembro 2015
https://en.wikipedia.org/wiki/Geographic_coordinate_system#/media/File:ECEF_ENU_Longitude_Latitude_relationships.svg

[34] Exemplos de mensagens $GPGGA, Setembro 2015
http://www.gpsinformation.org/dale/nmea.html

[35] Sítio com descrição do CANbus, Junho 2015
https://en.wikipedia.org/wiki/Canbus

[36] Arquiteturas suportadas pela glibc, Setembro 2015
https://sourceware.org/glibc/wiki/ABIList

[37] Sítio do VirtualBox, Junho 2015
https://www.virtualbox.org/

[38] Sítio do Vagrant, Junho 2015
https://www.vagrantup.com/

[39] Vagrant box com o Ubuntu 10.04, Junho 2015
http://files.vagrantup.com/lucid32.box

[40] Sítio da *release* do Ubuntu 10.04, Junho 2015
http://releases.ubuntu.com/10.04/

[41] Página sobre as ferramentas de sistemas embebidos, Junho 2015
http://www.emdebian.org/crosstools.html

[42] Sítio da biblioteca GSL, Junho 2015
https://www.gnu.org/software/gsl/
http://mirrors.fe.up.pt/pub/gnu/gsl/gsl-1.16.tar.gz

[43] *Application note* AN4246, Junho 2015
http://www.freescale.com/files/sensors/doc/app_note/AN4246.pdf

[44] Métodos para calcular a matriz inversa, Setembro 2015
https://en.wikipedia.org/wiki/Invertible_matrix#Methods_of_matrix_inversion

[45] Algoritmo de eliminação Gauss-Jordan , Setembro 2015
https://en.wikipedia.org/wiki/Gauss%E2%80%93Jordan_elimination

[46] Temporizador 555, Setembro 2015
http://www.ti.com/lit/ds/symlink/ne555.pdf

[47] Wikipedia, Túnel da Gardunha, Setembro 2015
https://pt.wikipedia.org/wiki/T%C3%BAnel_da_Gardunha

[48] Aplicação Web getPoints, Setembro 2015
http://a36017.alwaysdata.net/getPoints.html