



UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

UNIDADE ACADÊMICA DE SERRA TALHADA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

**Plataforma com ajuste de equilíbrio para
transporte de cargas frágeis utilizando
recursos de um *smartphone***

Por

Leonardo de Lima Souza

Serra Talhada,
dezembro/2015



UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE SERRA TALHADA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

LEONARDO DE LIMA SOUZA

**Plataforma com ajuste de equilíbrio para
transporte de cargas frágeis utilizando
recursos de um *smartphone***

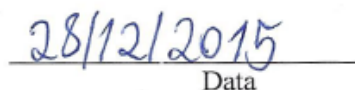
Trabalho de Conclusão de Curso apresentada ao
Curso de Bacharelado em Sistemas de Informação
da Unidade Acadêmica de Serra Talhada da
Universidade Federal Rural de Pernambuco como
requisito parcial à obtenção do grau de Bacharel.

Orientador: Prof. M.e Carlos André Batista

Serra Talhada,
dezembro/2015

Com base no disposto na **Lei Federal N° 9.610**, de 19 de fevereiro de 1998. [...] Autorizo para fins acadêmicos e científico a UFRPE/UAST, a divulgação e reprodução TOTAL, desta monografia intitulada **Plataforma com ajuste de equilíbrio para transporte de cargas frágeis utilizando recursos de um *smartphone***, sem ressarcimento dos direitos autorais, da obra, a partir da data abaixo indicada ou até que a manifestação em sentido contrário de minha parte determine a cessação desta autorização.


Assinatura


Data

Ficha catalográfica

S729p Souza, Leonardo de Lima

Plataforma com ajuste de equilíbrio para transporte de cargas frágeis utilizando recursos de um *smartphone* / Leonardo de Lima Souza. – Serra Talhada: O autor, 2015.
158 f.: il.

Orientador: Carlos André Batista.
Monografia (Bacharelado em Sistemas da Informação) – Universidade Federal Rural de Pernambuco. Unidade Acadêmica de Serra Talhada, Serra Talhada, 2015.
Inclui referências e apêndice.

1. Android. 2. Arduino. 3. acelerômetro. I. Batista, Carlos André, orientador. II. Título.

CDD 004

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE SERRA TALHADA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

LEONARDO DE LIMA SOUZA

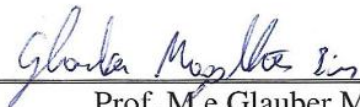
**Plataforma com ajuste de equilíbrio para transporte de cargas frágeis utilizando
recursos de um *smartphone***

Trabalho de Conclusão de Curso julgado adequado para obtenção do título de Bacharel em
Sistemas de Informação, defendida e aprovada por unanimidade em dia 21/dezembro/2015
pela banca examinadora.

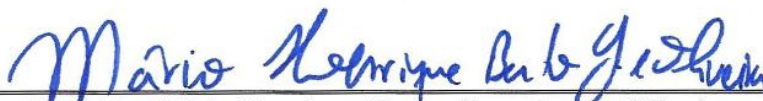
Banca Examinadora:



Prof. M.e Carlos André Batista
Orientador
Universidade Federal Rural de Pernambuco



Prof. M.e Glauber Magalhães Pires
Universidade Federal Rural de Pernambuco



Prof. D.r Mário Henrique Bento Gonçalves e Oliveira
Universidade Federal Rural de Pernambuco

Dedico este trabalho aos meus pais, Cícero Virginio de Souza e Maria do Carmo de Lima Souza por todo apoio, dedicação e paciência, ao meu tio, Miguel Leonardo Lima Filho e sua esposa Marilene pelo apoio e por terem me acolhido em sua residência durante todo o período acadêmico, à minha irmã Isabel, aos meus avôs maternos, Miguel Leonardo de Lima e Maria Paulina de Lima e paternos, João Virginio Neto e Maria Silvestre Souza.

AGRADECIMENTOS

Agradeço primeiramente a Deus por me permitir alcançar este momento tão especial.

A todos os professores que contribuíram de alguma forma para meu aprendizado, desde o ensino básico até os dias atuais, em especial os professores do curso de Sistemas de Informação da Unidade Acadêmica de Serra Talhada - UAST, principalmente ao professor Carlos André Batista por me aceitar como orientando e pelo esforço prestado durante todo o projeto.

Aos professores Glauber Magalhães Pires e Mário Henrique Bento Gonçalves e Oliveira por aceitarem participar como membros da banca examinadora.

A todos os colegas e amigos de curso pelo companheirismo ao enfrentar toda a trajetória envolvida na graduação e pelos momentos de diversão proporcionados até mesmo em alguns dias estressantes.

*“Porque melhor é a sabedoria do que os
rubis; e tudo o que mais se deseja não se pode
comparar com ela.”*

Provérbios 8:11

RESUMO

O transporte de cargas evolui constantemente por todas as partes do mundo e o avanço tecnológico nessa área permitiu o desenvolvimento de soluções que ajudam manter a estabilidade do veículo e consequentemente a conservação do material transportado. Entretanto, no que se refere ao transporte de cargas frágeis as tecnologias desenvolvidas ainda demonstram insuficiência, considerando, por exemplo, que na maioria dos casos a suspensão utilizada em caminhões destinados ao transporte de chapas de vidro é a mesma presente em caminhões que transportam qualquer outro material e o mesmo acontece em ambulâncias no transporte de pacientes. Na criação das suspensões presentes nos veículos mais modernos é utilizada uma considerável quantidade de sensores e do mesmo modo, em vários projetos desenvolvidos por amadores usando plataformas de prototipagem de baixo custo tem envolvido o uso de diversos sensores para solucionar problemas do cotidiano. Este trabalho aborda o desenvolvimento de um protótipo para transporte de materiais frágeis com nivelamento automático que utiliza a plataforma de prototipagem Arduino para controle de atuadores e um *smartphone* Android que serve como central de processamento e também central dos sensores necessários ao nivelamento da plataforma. A plataforma desenvolvida conta com duas chapas, superior e inferior, separadas por hastes verticais móveis responsáveis por manter a chapa superior nivelada, na qual é colocada a carga. O desenvolvimento enfatizou principalmente a funcionalidade de correção de inclinações aplicadas à plataforma e por meio de experimentos constatou-se uma correção média de desnivelamento acima de 50% na maioria das inclinações aplicadas à plataforma.

Palavras-chave: Android, Arduino, acelerômetro, nivelamento.

ABSTRACT

The cargo transportation is constantly evolving all over the world and technological advances in this area allowed the development of solutions that help maintain the stability of the vehicle and consequently the conservation of the transported material. However, with regard to the transport of fragile cargos technologies also developed demonstrate inadequate, considering, for example, that in most cases the suspension used on trucks for the transport of glass sheets is the same as present in trucks carrying any material and the same is true in ambulances to transport patients. The creation of the suspensions present in most modern vehicles is used a considerable amount of sensors and likewise in several projects developed by amateurs using low-cost prototyping platforms have involved the use of multiple sensors to solve everyday problems. This work describes the development of a prototype for the transport of fragile materials with automatic leveling utilizing Arduino prototyping platform for controlling actuators and an Android smartphone which serves as the central processing unit and also sensors to the central platform leveling. The developed platform comprises two plates, upper and lower, separated by a movable vertical stems responsible for keeping the upper level plate, to which the cargo is placed. The development emphasized mainly inclination correction function applied to the platform and by experiments found an average correction unevenness above 50% in most inclinations applied to the platform.

Keywords: Android, Arduino, accelerometer, leveling.

LISTA DE FIGURAS

Figura 2.1 – Arduino UNO	24
Figura 2.2 – <i>Protoboard</i> de 400 pontos (orifícios)	25
Figura 2.3 – Arquitetura do Android OS	27
Figura 2.4 – Processo de criação do APK	29
Figura 2.5 – Principais sensores e outros componentes do Big Dog	33
Figura 3.1 – Ilustração de acelerômetro	36
Figura 3.2 – Conexão entre um <i>smartphone</i> e um Arduino	37
Figura 3.3 – Exemplo de modelo com triângulo de sustentação para plataforma	39
Figura 4.1 – Tela inicial do aplicativo aferidor	41
Figura 4.2 – Giro de 90 graus do <i>smartphone</i>	42
Figura 4.3 – Relação entre os ângulos do servo motor e os da chapa superior da plataforma	43
Figura 4.4 – Fluxograma: algoritmo do <i>smartphone</i> aferidor	45
Figura 4.5 – Tela inicial do aplicativo Sensor Analyzer	46
Figura 4.6 – Tela de configuração e <i>status</i>	47
Figura 4.7 – Tela de configuração e <i>status</i> – execução	48
Figura 4.8 – Fluxograma: algoritmo do <i>smartphone</i> carga	49
Figura 4.9 – Tela inicial Controle Plataforma	52
Figura 4.10 – Tela <i>status</i> : com ângulos x e y dos <i>smartphones</i> aferidor e carga	54
Figura 4.11 – Tela <i>status</i> : exemplo de configuração	55
Figura 4.12 – Esquema da comunicação sem fio entre os softwares desenvolvidos	56
Figura 4.13 – Esquema elétrico com fonte CC	57
Figura 4.14 – Esquema elétrico com fonte CC	58
Figura 4.15 – Funcionamento das hastes móveis	59
Figura 4.16 – Vista frontal do projeto final	60
Figura 4.17 – Vista lateral esquerda	61
Figura 4.18 – Vista lateral direita	61
Figura 4.19 – Inclinações no eixo x	63
Figura 4.20 – Inclinações no eixo x com valores absolutos	64
Figura 4.21 – Variância e desvio padrão	64
Figura 4.22 – Inclinações no eixo y	65
Figura 4.23 – Inclinações no eixo y com valores absolutos	65
Figura 4.24 – Inclinações na diagonal	66
Figura 4.25 – Inclinações na diagonal com valores absolutos	66
Figura 4.26 – Inclinações executadas em menor tempo	67
Figura 4.27 – Tela <i>status</i> : inclinações no eixo x	67
Figura 4.28 – Tela <i>status</i> : inclinações no eixo x sem a plataforma	68
Figura 4.29 – Tela <i>status</i> : inclinações no eixo y	69

Figura 4.30 – Tela <i>status</i> : inclinações no eixo y sem a plataforma	69
Figura 4.31 – Tela <i>status</i> : inclinações na diagonal	70
Figura 4.32 – Tela <i>status</i> : inclinações na diagona sem a plataforma	71
Figura 5.1 – Organização de arquivos do código fonte do Aferidor	77
Figura 5.2 – Organização de arquivos do código fonte do sensor analyzer	100
Figura 5.3 – Organização de arquivos do código fonte do software de controle	121

LISTA DE QUADROS

Quadro 2.1 – Sensores suportados pela plataforma Android	30
Quadro 4.1 – Conversão para ângulos	42
Quadro 4.2 – Opções de configurações do aferidor	53

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i> , Interface de Programação de Aplicação
APK	<i>Android Application Package</i> , Pacote de Aplicativo Android
bps	bits por segundo
CC	Corrente Contínua
cm	centímetro
DARPA	<i>Defense Advanced Research Projects Agency</i> , Agência de Projetos de Pesquisa Avançada de Defesa
GPS	<i>Global Positioning System</i> , Sistema de Posicionamento Global
dex	<i>Dalvik Executable</i> , executável <i>Dalvik</i>
hPa	hectopascal
HW	hardware
IDE	<i>Integrated Development Environment</i> , Ambiente de Desenvolvimento Integrado
IP	<i>Internet Protocol</i> , Protocolo de Internet
JOGL	<i>Java Open Graphics Library</i> , Biblioteca Gráfica Livre para Java
JVM	<i>Java Virtual Machine</i> , Máquina Virtual Java
KB	<i>kilobyte</i>
kg	quilograma
LED	<i>Light Emitting Diode</i> , Diodo Emissor de Luz
LiDAR	<i>Light Detection And Ranging</i> , detecção de luz e variância
lx	Lux, intensidade de iluminação
m	metro
mA	miliampere
mbar	milibar
MDF	<i>Medium-Density Fiberboard</i> , Placa de Fibra de Madeira de Média Densidade
MIT	<i>Massachusetts Institute of Technology</i> , Instituto de Tecnologia de Massachusetts
ms	milissegundo
m/s ²	metros por segundo quadrado
NFC	<i>Near Field Communication</i> , Comunicação por Campo de Proximidade
OTG	<i>On The Go</i>
rad/s	Radiano por segundo

RCX	<i>Robotic Command Explorer</i> , Explorador de Comando Robótico
s	segundo
SDK	<i>Software Development Kit Android</i> , Kit de Desenvolvimento de Software
SMS	<i>Short Message Service</i> , Serviço de Mensagens Curtas
SQL	<i>Structured Query Language</i> , Linguagem de Consulta Estruturada
SW	software
USB	<i>Universal Serial Bus</i> , Barramento Serial Universal
V	Volts
μ T	microtesla

SUMÁRIO

1	INTRODUÇÃO	16
1.1	Objetivos	17
1.2	Motivação	17
2	REFERENCIAL TEÓRICO	18
2.1	Robótica	18
2.2	Arduino	23
2.3	Android OS	26
2.4	Trabalhos relacionados	31
2.4.1	Aplicativo Em Android Para Controle De Unidades Robóticas Móveis Com Arduino	31
2.4.2	Big Dog	32
3	METODOLOGIA	34
3.1	Desenvolvimento para Android	34
3.2	Desenvolvimento para Arduino	36
3.3	Comunicação entre Android e Arduino	37
3.4	Confecção da Plataforma	38
3.5	Testes	39
4	DESENVOLVIMENTO DO PROJETO	40
4.1	Desenvolvimento para Android	40
4.2	Desenvolvimento para Arduino	49
4.3	Comunicação entre Android e Arduino	50
4.4	Desenvolvimento do controle	51
4.5	Confecção da Plataforma	56
4.6	Testes, experimentos e resultados	62
5	CONCLUSÃO	71
5.1	Considerações finais	71
5.2	Contribuições deste trabalho	72
5.3	Proposta para trabalhos futuros	72
	REFERÊNCIAS BIBLIOGRÁFICAS	74
	APÊNDICE A – CÓDIGO FONTE DO APLICATIVO AFERIDOR	77
	APÊNDICE B – CÓDIGO FONTE DO APLICATIVO <i>SENSOR ANALYZER</i>	100
	APÊNDICE C – CÓDIGO FONTE DO SOFTWARE DE CONTROLE	121
	APÊNDICE D – CÓDIGO FONTE DO <i>FIRMWARE</i> DO ARDUINO	157

1 Introdução

O transporte de cargas frágeis em veículos terrestres está suscetível a estradas irregulares, motoristas imprudentes e situações adversas que podem comprometer o estado do material transportado.

Diversas soluções foram desenvolvidas ao decorrer dos anos, como por exemplo, amortecedores, que podem ser hidráulicos (convencionais, a base de ar e óleo ou pressurizados, com gás hidrogênio e óleo) e eletrônicos, controle de freios, variados tipos de suspensão, etc.. Uma das mais relevantes se trata da suspensão inteligente, que envolve, entre outros, o controle da resistência dos amortecedores, da intensidade das frenagens, da tração individual para cada roda etc., sendo encontrada principalmente nos ônibus mais modernos, tentando ao máximo manter o veículo nivelado mesmo em vias inclinadas e/ou em situação adversas que podem causar acidentes.

Todas essas soluções envolvem principalmente a utilização de vários sensores que monitoram o contexto de cada alteração, que pode ou não gerar uma reação dos componentes do sistema para manter a estabilidade do transporte. A utilização de sensores está também cada vez mais presente no desenvolvimento de projetos feitos e divulgados mundialmente por amadores ou profissionais que, utilizando plataformas de prototipagem de baixo custo, conseguem solucionar algum problema ou criar alternativas à realização de determinadas tarefas ou desenvolver novas possibilidades de entretenimento.

Os adeptos dessas tecnologias de prototipagem vêm aumentando constantemente e uma das mais populares entre elas é o Arduino, uma plataforma de baixo custo criada na Itália, voltada inicialmente para fins educativos. Existem vários sensores e componentes que podem ser adicionados ao Arduino para concretizar muitas ideias, projetos ou soluções.

O conjunto sensorial presente em um *smartphone* também é bastante considerável, tal como outros recursos que podem ser empregados em situações diferentes das quais estão intrinsicamente ligados. Este trabalho aborda a utilização do Arduino juntamente com um *smartphone* Android de modo que se tire proveito de seus meios de comunicação e sensores para construção de um protótipo de plataforma para transporte de cargas frágeis.

1.1 Objetivos

Este trabalho tem como objetivo geral criar um protótipo de plataforma para transporte de cargas frágeis usando equipamentos e tecnologias mais acessíveis, como por exemplo, os sensores de um *smartphone*, para manter o nivelamento e reduzir desperdícios e/ou danos físicos ao material transportado.

Os objetivos específicos são:

- Desenvolver o software Android para leitura de sensores do *smartphone*;
- Desenvolver o software para o Arduino usado no controle da plataforma;
- Estabelecer a comunicação entre *smartphone* e Arduino;
- Criar a plataforma móvel: projeto mecânico e eletroeletrônico;
- Validar o projeto por meio de experimentos.

1.2 Motivação

E evidente a evolução dos veículos terrestres direcionados ao transporte de pessoas, animais, materiais sólidos, como madeira, ferro, plástico, entre outros. Entretanto, essa evolução mostra-se insuficiente ou até mesmo estagnada quando se trata de alguns tipos de cargas frágeis como chapas vidro, pacientes em ambulâncias ou alguns tipos de líquidos.

O transporte de chapas de vidro, por exemplo, é normalmente feito em caminhões e algumas técnicas foram desenvolvidas para este fim, uma das mais utilizadas é por meio de um cavalete no qual a chapa de vidro é colocada na posição vertical, fixada ao cavalete e à carroceria do caminhão de forma que todo impacto ou desnível ocorrido ao caminhão é diretamente transmitido ao material, o que pode ocasionar trincas ou quebras irreversíveis.

O transporte de pacientes em ambulância compartilha de características similares às citadas, porém com o agravante de se tratar de um ser humano e o fato de que o tempo gasto até chegar ao hospital pode ser decisivo para salvar uma vida. Contudo, o aumento da velocidade durante o percurso pode intensificar os abalos ou impactos sofridos pelo paciente e assim piorar seu quadro de saúde.

Evidencia-se que o desenvolvimento de um mecanismo que utilize recursos mais acessíveis, que se torne mais popular e amenize os prejuízos presentes nos contextos citados e também em outros é algo muito importante.

Por outro lado a popularização de ambas as plataformas livres, Android e Arduino, tornou possível o desenvolvimento de várias ideias e projetos, que envolvem os mais diversos tipos de automação, desde controlar a iluminação de uma residência até mesmo ligar um carro a partir de um celular.

A maioria dos projetos que dispõem de aplicativos para dispositivos móveis voltados para comunicação com Arduino usa o *smartphone* basicamente como controle remoto, ou seja, existe um ou mais equipamentos que podem ser controlados por meio do mesmo, por exemplo, o acionamento das luzes, portas e/ou qualquer outro equipamento eletrônico na automação residencial, controle de assessorios veiculares, como volume do som ou acendimento de faróis e controle de robôs ou brinquedos para entretenimento.

Entretanto, um *smartphone* dispõe de uma variada gama de recursos que na maioria dos casos não é aproveitada, como por exemplo, os diversos sensores, que dependendo do dispositivo, podem variar em número e categoria, as formas de conexões e transmissão de dados e, além disso, o processador, que ao passar do tempo tem evoluído significativamente, passando a ter um alto nível de processamento e múltiplos núcleos.

2 Referencial Teórico

O presente trabalho está relacionado diretamente com três áreas: a robótica, a tecnologia *mobile*, que no caso é utilizado o Android, e a prototipação, por meio do Arduino, uma plataforma de prototipagem que é usada para o controle de atuadores.

2.1 Robótica

Há vários anos a humanidade vem sempre idealizando a construção de utensílios que realizam ou ajudam numa determinada tarefa, sendo estes, servos obedientes e que sempre estariam dispostos para missões ambiciosas que lhes era conferida pelos seus criadores.

Registros antigos da mitologia grega contêm relatos sobre pessoas que criaram seres para suprir necessidades específicas. Um deles chamado Pigmalião era um escultor que achava todas as mulheres ao seu redor imperfeitas, então decidiu esculpir uma mulher ideal segundo os seus critérios, no final ele ficou muito apaixonado pela sua criação, mas lamentava-se por esta ser imóvel. Ao ver todo seu sofrimento, Afrodite a Deusa do amor, transformou a escultura numa mulher verdadeira que logo após se casou com o seu criador. (AZEVEDO et al., 2010).

Outro mito grego interessante refere-se à história de um deus renegado conhecido como Vulcano ou Hefestos, ele costumava forjar raios, joias e aparatos metálicos e mecânicos em sua oficina que ficava na boca do vulcão na ilha de Lemnos e, fez para si mesmo, servos metálicos conhecidos como ciclopes para o ajudar em seus trabalhos (AZEVEDO et al., 2010).

Crenças e mitologias a parte, no decorrer dos tempos a humanidade continuou almejando tais ideias e, por coincidência ou não, um dos estudiosos mais conhecidos historicamente como um precursor da área em questão foi um matemático e engenheiro grego chamado Ctesibius que viveu cerca de 285-222 a.C. em Alexandria e arquitetou diversos aparelhos robóticos, o mais famoso foi a *clepsidra* ou relógio de água, que foi um dos primeiros sistemas para medir o tempo criado pelo homem. (AZEVEDO et al., 2010).

Outro que se destacou foi Heron de Alexandria, geômetra e engenheiro grego que viveu depois de Cristo e desenvolveu várias invenções na área da automação, dentre elas a primeira máquina de vender bebidas da história, na qual colocava-se uma moeda e recebia um jato de água. Criou máquinas que podiam se mover para frente e para trás por meio de engrenagens em um sistema que utilizava energia cinética de grãos de trigo que caíam de um recipiente no topo da máquina. Foi o responsável também pela criação do primeiro motor a vapor da história. (AZEVEDO et al., 2010).

Outro influente estudioso nessa área, e em várias outras, foi Leonardo da Vinci, ele tinha uma visão bem a frente do seu tempo, idealizando projetos como helicóptero, tanque de guerra, calculadora e uso de energia solar. Porém um pequeno número de seus projetos foi de fato construído. Um bem interessante que ficou conhecido como “Robô Leonardo” tinha a semelhança com um cavaleiro com armadura e realizava alguns movimentos humanos, como se sentar, mover os braços e levantar a viseira do capacete automaticamente como se uma pessoa estivesse em seu interior (AZEVEDO et al., 2010). Segundo Ruic (2012) uma réplica

desse projeto foi construída e comprovou-se a sua eficiência, hoje faz parte do museu em homenagem ao inventor, em Florença, Itália.

Jacques de Vaucanson, um inventor e artista francês, foi também bastante importante ao criar, em 1738, o primeiro robô funcional, um humanoide que tocava flauta, assim como um pato que se alimentava. Ao divulgar suas obras, a tecnologia avançou a ponto das pessoas preverem o uso de criações mecânicas como força de trabalho. (AZEVEDO et al., 2010).

Por volta de 1921 o termo robô passa a ser presente no cenário das artes como literatura, cinema e peças de ficção de científica. Um dos pioneiros na popularização do termo foi escritor tcheco Karel Capek, que escreveu a peça R.U.R. (*Rossum's Universal Robots*) em que um cientista chamado Rossum desenvolve uma substância química usada na construção de robôs humanoides que deveriam ser obedientes e fazer todo o trabalho físico, porém após a construção de um grande número de robôs eles se tornaram inteligentes e dominaram o mundo. (AZEVEDO et al., 2010).

Dai em diante várias obras de ficção envolvendo robôs que amedrontavam seres humanos surgiram, entre elas: Frankenstein (1818), de Mary Shelley, muitas vezes considerado o primeiro romance de ficção científica e Metrópoles (1927), um filme alemão de Fritz Lang, no qual demonstra uma preocupação com a mecanização da vida industrial nos grandes centros urbanos (AZEVEDO et al., 2010).

Inserindo uma nova opinião bastante diferente dessa onda de ideias que pregavam medo e insegurança quanto ao uso de robôs, Isaac Asimov, a partir da década de 1940, publicou diversos estudos relacionados a esta temática, onde se defendia que os robôs, na verdade, desempenham funções para ajudar e proteger o ser humano contra o mal (AZEVEDO et al., 2010).

Asimov também criou as famosas três leis fundamentais da robótica, que afirmam:

Um robô não pode causar dano a um ser humano nem, por omissão, permitir que um ser humano sofra;

Um robô deve obedecer às ordens dadas por seres humanos, exceto quando essas ordens entrarem em conflito com a Primeira Lei;

Um robô deve proteger sua própria existência, desde que essa proteção não se choque com a Primeira nem com a Segunda Lei da robótica.

A partir deste contexto cheio de opiniões contraditórias se iniciou a criação e utilização de robôs nas indústrias, principalmente automobilística, a qual foi a pioneira quando a General Motors resolveu usar pela primeira vez na sua linha de montagem um robô

criado por Joseph F. Engelberger, o qual é considerado o pai da robótica (AZEVEDO et al., 2010).

Entretanto, segundo Santos (2003-2004) nem todo sistema automático é considerado robô, sistemas com funções fixas como alguns brinquedos de mobilidade e ou até mesmo máquinas de comando numérico, que desempenham algumas funções fixas de controle em atividades industriais, não são considerados robôs. Para ser considerado como tal o dispositivo deve ter a capacidade de programação e adaptabilidade ao problema prático.

Ainda segundo Santos (2003-2004) os robôs podem ser classificados de acordo com sua geração da seguinte forma:

- 1^a – Executores: repetem uma sequência de instruções pré-gravada como a pintura ou soldadura;
- 2^a – Controlados por sensores: possuem malhas fechadas de realimentação sensorial e tomam decisões com base nos sensores;
- 3^a – Controlados por visão: a malha fechada de controle inclui um sistema de visão onde imagens são processadas;
- 4^a – Controle adaptativo: podem reprogramar as suas ações com base nos seus sensores;
- 5^a – Com inteligência artificial: usa técnicas de inteligência artificial para tomar as suas decisões e até resolver problemas.

Azevedo et al. (2010) destacam também quais são os componentes normalmente encontrados num robô:

- Controlador – parte central de um robô, dotada de um microprocessador e memória para execução de seu(s) programa(s);
- Sensores – responsáveis por detectar sinais como tato, imagens, sons, rotação, luz, cor, etc.;
- Atuadores – podem ser motores de diversos tipos, como mecânicos, elétricos, hidráulicos ou pneumáticos; servem para mover o robô e seus manipuladores;
- Manipuladores – são membros como braços e garras, a variedade de movimentos que um manipulador pode realizar é medida em graus de liberdade;

- Engrenagens – elementos mecânicos compostos de rodas dentadas. Quando duas engrenagens estão em contato, a engrenagem que fornece a força e rotação para a outra é dita engrenagem motora, e a outra é dita engrenagem movida;
- Eixo – peça que liga um motor a engrenagens ou rodas;
- Fonte de energia – define como o controlador e os demais componentes eletrônicos serão alimentados, que tipo de bateria e/ou gerador é usado;
- Fiação – transmitem sinais entre o controlador, os sensores, os atuadores e também para a alimentação desses componentes;
- Estrutura – a “carcaça” do robô, formada por um conjunto de peças de tamanhos, formatos e cores diversas, e em alguns casos, rodas, parafusos e placas.

Ao passar dos anos, a robótica evoluiu para diversas áreas, como por exemplo, entretenimento (brinquedos, monstros de filmes, etc.), realização de atividades à distância, como os robôs específicos para desarmar bombas, atividade de alta precisão, como cirurgias, exploração de ambientes de alto risco ou inacessíveis ao ser humano (regiões vulcânicas, fundo do mar, superfícies de outros planetas, etc.), educação, entre outras.

Na educação, a robótica tem recebido especial atenção por meio da utilização de Kits robóticos que contêm diversos componentes para montagem de vários tipos de robôs dependendo da criatividade do aluno. Pretende-se com isso incentivar alunos a conhecer a importância desta área e descobrir novos talentos. A seguir veem-se alguns dos Kits utilizados.

- *Kits Alfa*

Os Kits de Robótica Alfa foram desenvolvidos pela empresa PNCA, localizada nos Estados Unidos. Os mais conhecidos são: Kit ALFA Hobby e o kit ALFA Educ 2008. O primeiro tem foco em iniciantes autônomos no estudo de robótica, já o segundo é destinado à instituições de ensino. Ambos são bastante semelhantes, contendo: um módulo de controle MC2.5, o programa LEGAL 2008, que é um ambiente de programação utilizado na construção dos projetos, alguns tipos de sensores como: de contato, de temperatura, de cor e infravermelho; Um cabo USB (*Universal Serial Bus*, Barramento Serial Universal) para conexão com um PC, motores e bases para motores, algumas rodas e peças metálicas estruturais para montagem, porcas e parafusos diversos.

- *Lego Mindstorms*

É um dos Kits de robótica mais conhecido. É resultado da parceria entre o *Media Lab* do MIT (*Massachusetts Institute of Technology*, Instituto de Tecnologia de Massachusetts) e o *LEGO Group*. Foi lançado comercialmente em 1998 e tem como foco principal o ensino de robótica na educação. Existem diferentes Kits disponíveis no mercado, mas normalmente eles são bem similares, constituídos por um conjunto de peças de plástico, tijolos cheios e vazados (característicos das peças LEGO de montar convencionais), rodas, motores, eixos, engrenagens, polias, correntes, sensores de toque, de luminosidade e de temperatura. Tudo é controlado por um processador programável e o primeiro Kit tinha o nome do seu controlador, o RCX (*Robotic Command Explorer*, Explorador de Comando Robótico). Os Kits mais recentes são: o NXT 1.0 e o NXT 2.0.

Neste projeto é usado um Kit de robótica Arduino, ele tem foco na criação de protótipos de projetos que envolvem eletrônica, mecânica e robótica. O tópico seguinte aborda com mais detalhes essa tecnologia.

2.2 Arduino

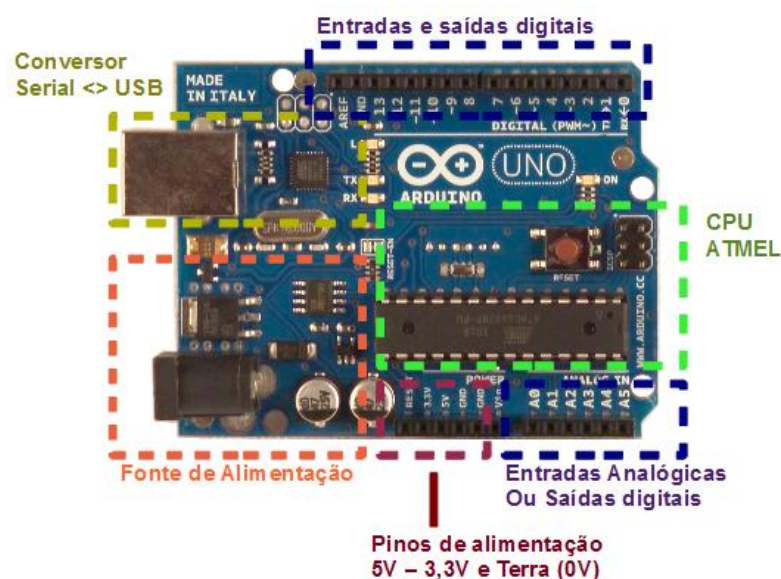
Arduino é uma plataforma de prototipagem eletrônica de código aberto baseada em hardware e software flexíveis e fáceis de usar. Aplica-se aos mais diversos tipos de pessoas e interesses, como artistas, designers ou qualquer pessoa interessada em criar objetos ou ambientes interativos com facilidade para resolver problemas do cotidiano (ARDUINO, 2015).

Ele surgiu em 2005 a partir de um projeto iniciado na cidade de Ivrea na Itália. Tinha um objetivo educacional, com intuito de ser inserido em projetos escolares, porém com um orçamento menor que outros sistemas de prototipagem disponíveis naquela época (LABDEGARAGEM, 2015). O sucesso nessa fase foi tão expressivo que mais de 50 mil unidades foram vendidas e rendeu até um documentário em 2010 sobre a trajetória de evolução do Arduino (SOARES, 2013). Com o alcance do sucesso algumas variações do projeto original foram lançadas no mercado, inclusive um modelo desenvolvido por uma empresa Estadunidense.

O Arduino é formado por uma pequena placa de circuito eletrônico que contém uma controladora, ou seja, uma unidade de processamento, Atmel AVR de 8 bits, pinos digitais e

analógicos de entrada e saída, uma conexão USB, que permite a comunicação com computadores e outros dispositivos (também utilizada durante a programação) (SOARES, 2013). É alimentado eletricamente por meio da própria USB ou por uma fonte externa ligada na entrada de CC (Corrente Contínua) com tensão mínima de 6 V e máxima de 20 V sendo altamente recomendado a utilização do intervalo de 7 a 12 V, dispõem de uma memória *Flash* de 32 KB (*kilobyte*) para armazenamento do código executável, sendo que 0,5 KB é usado para o *bootloader* (ARDUINO UNO, 2015). Veja o Arduino UNO na **Figura 2.1**. UNO em italiano significa um e ele foi o primeiro a ser desenvolvido no projeto original.

Figura 2.1 – Arduino UNO



Fonte: adaptado de Basconcello Filho (2012a)

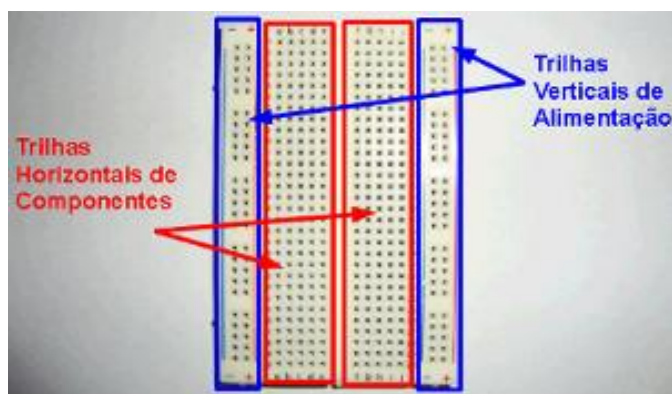
Na programação é utilizada uma linguagem própria, baseada na linguagem da plataforma de prototipagem Wiring (que é muito similar a C/C++), podendo-se também usar outras linguagens como C, C++ e Java. É comumente usado o ambiente de desenvolvimento oficial, que também se chama Arduino, ele foi desenvolvido, em sua maioria, na linguagem Java e pode ser baixado gratuitamente na Internet. Tem como principais características ser simples, intuitivo e multiplataforma, sendo compatível com sistemas Windows, MAC e Linux. (BASCONCELLO, 2012b).

Por padrão o Arduino não dispõe de sensores ou conexões sem fio ou quaisquer outras funcionalidades mais complexas, mas isso pode ser facilmente contornado por meio do acoplamento de circuitos externos através de pinos com posições padronizadas denominados *Shields* (BASCONCELLO, 2012b). Com isso se torna possível a construção de diversos projetos usando essa tecnologia em áreas como automação comercial, residencial, veicular,

monitoramento do ambiente ao seu redor, etc. De acordo com Gonçalves (2013) a utilização do Arduino na área da robótica é muito frequente pela grande flexibilidade de escalabilidade do sistema em desenvolvimento.

O Arduino poder ser adquirido individualmente ou incluso em kits de robótica ou eletrônica. Existe uma grande variedade de kits disponíveis para compra na Internet e, além do próprio Arduino, eles são constituídos por outros componentes eletrônicos como capacitores, resistores, LEDs (*Light Emitting Diode*, Diodo Emissor de Luz), motores, sensores, interruptores, entre outros. Um dos mais importantes componentes é a *protoboard* (Veja **Figura 2.2**), que é utilizada na prototipagem dos projetos. Ela é formada por uma base plástica que contém inúmeros orifícios destinados à inserção de terminais de componentes eletrônicos, internamente existem ligações entre os orifícios, permitindo a montagem de circuitos sem a utilização de solda, o que representa uma grande vantagem. (SOUZA, 2013)

Figura 2.2 – Protoboard de 400 pontos (orifícios)



Fonte: Labdegaragem (2011)

A maioria dos projetos construídos envolvendo Arduino e monitoramento sensorial pressupõe a compra de inúmeros sensores (p. ex. sensores de temperatura, vibração, aceleração, etc.), porém em alguns desses projetos, dependendo da aplicação, talvez não seja necessário adquirir esses dispositivos, levando-se em consideração que atualmente a maioria das pessoas possui um *smartphone* e este, por sua vez, dispõe de uma gama considerável de sensores que podem ser aproveitados. Existem também outras propostas que abrangem a interação entre esses dois artefatos, entretanto, geralmente o *smartphone* funciona somente como um controle remoto.

Este trabalho tem como uma das principais concepções utilizar de modo mais efetivo os sensores disponíveis em *smartphones* da plataforma Android, bem como outros recursos

como as conexões sem fio *wi-fi* e *bluetooth*, para realizar determinadas tarefas. Veja no próximo tópico mais detalhes sobre essa interessante plataforma *mobile*.

2.3 Android OS

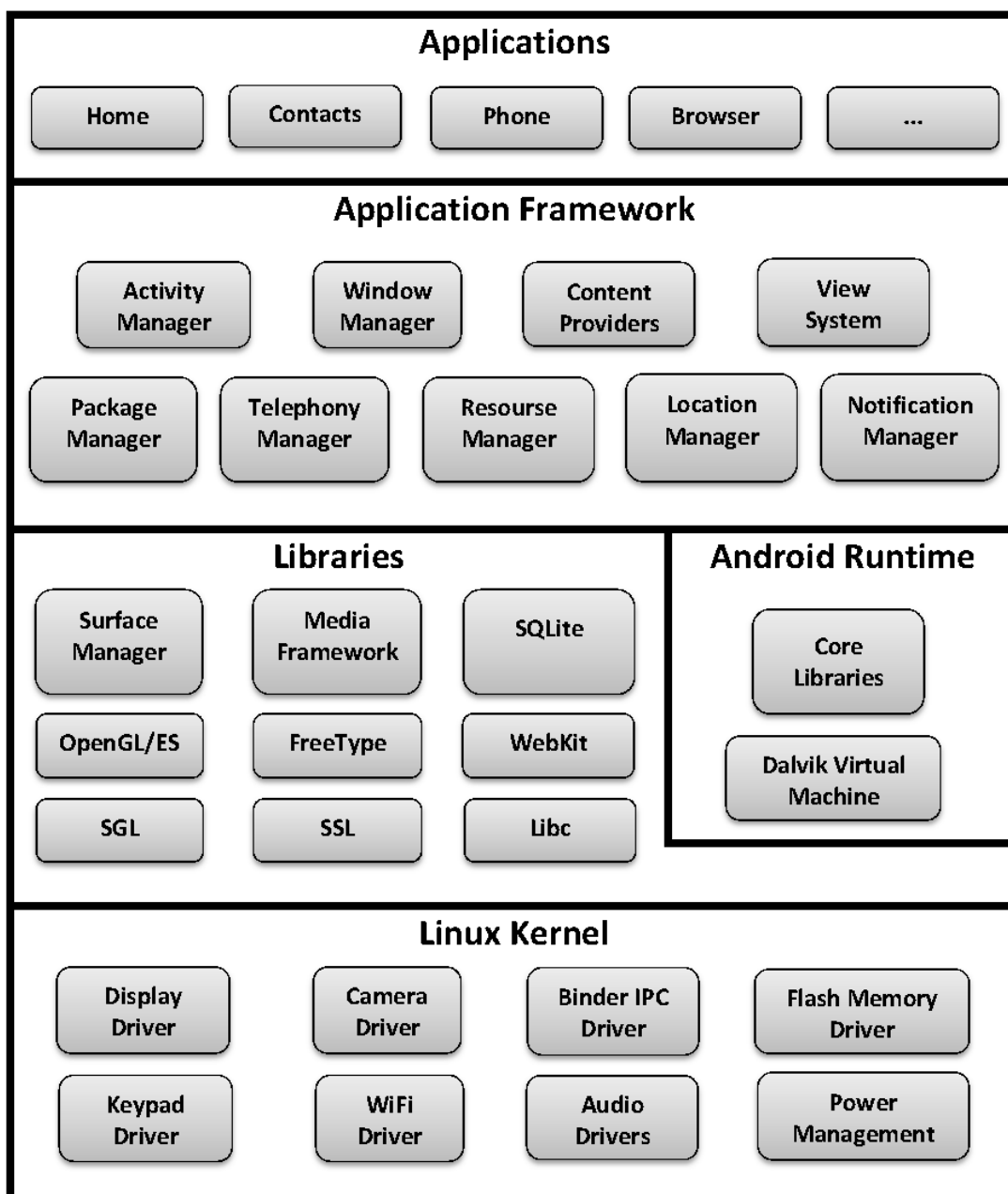
O projeto Android tem seu início ligado à empresa Android Inc., fundada por Andy Rubinera, Nick Sears e Chris White, em outubro de 2003 na cidade de Palo Alto, Califórnia. Inicialmente a empresa desenvolvia tecnologia independente de outras organizações e mantinha segredo absoluto sobre os seus projetos. Dentre estes tinha um direcionamento à criação de um sistema operacional para *smartphones*, tendo em vista o crescente mercado mundial de dispositivos móveis, o que entraria diretamente em concorrência com outros sistemas da categoria, como por exemplo, o Symbian, desenvolvido pela Nokia e Windows Mobile, da Microsoft. Entretanto a falta de investimentos impossibilitava a continuidade do projeto. (GUIMARÃES, 2013).

Dois anos mais tarde, em 17 de agosto de 2005, a Google adquiriu a companhia e colocou todo seu time de desenvolvedores, liderados por Andy Rubinera, que passou a integrar o corpo de membros da empresa, para trabalhar em uma plataforma móvel baseada em Linux. Em 5 de novembro de 2007 foi criado oficialmente o projeto Android, com objetivo de desenvolver um sistema *mobile* sob uma licença de código aberto e construído sobre o *Kernel* do Linux versão 2.6. (GUIMARÃES, 2013).

O projeto teve uma evolução significava após a fundação, ainda em 2007, da *Open Handset Alliance* que de acordo com Ableson (2009) é um grupo de empresas liderado pela Google que inclui operadoras de telefonia móvel, fabricantes de aparelhos portáteis, fabricantes de componentes, provedores de plataformas e soluções de software e empresas de *marketing*, todos com o objetivo de acelerar a inovação na área de celulares e oferecer aos consumidores uma melhor experiência, mais rica e mais barata em telefonia móvel.

A plataforma Android tem como base central o *Kernel* do Linux e, somando-se a este, dispõe de uma vasta quantidade de bibliotecas, componentes, interface gráfica e outros recursos que estão distribuídos numa arquitetura dividida em cinco camadas: Aplicação, *Framework* de Aplicação, Bibliotecas, Ambiente de Execução Android e o *Kernel* do Linux, veja **Figura 2.3** e em seguida uma descrição mais abrangente das camadas da arquitetura.

Figura 2.3 – Arquitetura do Android OS



Fonte: adaptado de Android Developer (2015)

Applications (Aplicações): o Android possui nativamente diversos aplicativos como programa de envio de mensagem SMS (*Short Message Service*, Serviço de Mensagens Curtas), cliente de e-mail, mapas, calendário, navegador de Internet e outros. Oficialmente a linguagem Java é utilizada no desenvolvimento de todos os aplicativos para Android (ANDROID, 2015). Os aplicativos desenvolvidos ficam nesta camada e são o que os usuários finais encontram no Android.

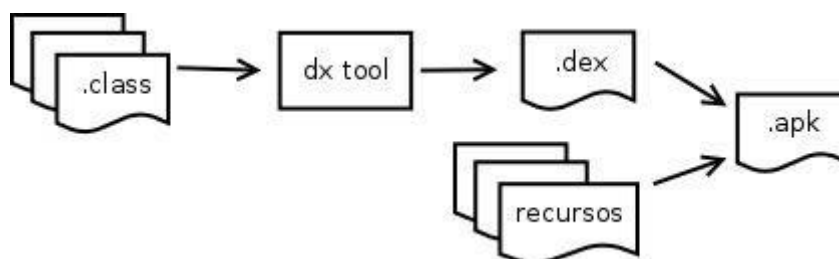
Application Framework (Framework de Aplicação): é o ambiente que disponibiliza os recursos para auxiliar o desenvolvedor a realizar o seu trabalho. É a parte do Android que está mais bem documentada e detalhada, pelo fato de ser a camada que fornece aos desenvolvedores a possibilidade de criarem aplicações inovadoras para o mercado (FERNANDES & LAURINDO, 2011). O *Framework* de Aplicação permite o desenvolvimento de novos aplicativos com reaproveitamento das funcionalidades presentes na plataforma.

Libraries (Bibliotecas): o Android OS possui bibliotecas nativas C/C++, muitas delas produzidas pela comunidade de código livre. Elas facilitam o desenvolvimento de aplicações e possibilitam o acesso às funcionalidades de hardware. Entre elas estão um rápido mecanismo de renderização web (utilizado pelo Safari, Chrome e outros navegadores), um completo banco de dados SQL (*Structured Query Language*, Linguagem de Consulta Estruturada), bibliotecas de gráfico 3D e uma camada de certificado de segurança (FERNANDES & LAURINDO, 2011). O banco de dados do Android é o SQLite, no qual são utilizados comandos SQL padrão, facilitando o desenvolvimento de aplicativos que necessitam de armazenamento, entretanto, de acordo com Monteiro (2012, p. 133), ao contrário da maioria dos bancos de dados SQL, este, não necessita de um processo servidor e as tabelas, *views*, índices e *triggers* (componentes padrões em diversos tipos de servidores de banco de dados) são armazenados e um único arquivo em disco, no qual são realizadas as operações de leitura e escrita. Destaca ainda que o banco de dados é acessível por qualquer classe da aplicação que o criou, mas não pode ser acessado por outra.

Android Runtime (Ambiente de Execução Android): como os aplicativos Android são escritos em Java é necessário uma máquina virtual Java para execução dos mesmos, esta máquina virtual Java chama-se *Dalvik*, ela é especial para executar aplicações Java em dispositivos móveis (FERNANDES & LAURINDO, 2011). Convencionalmente o código fonte Java é compilado utilizando o compilador Java e, após, geram-se arquivos no formato .class chamados *bytecodes* que são executados na JVM (*Java Virtual Machine*, Máquina Virtual Java) porém, no Android, segundo Monteiro (2012, p. 37), após esse processo, o bytecode é recompilado pela ferramenta “dx”, que está inclusa no SDK Android (*Software Development Kit* Android, Kit de Desenvolvimento de Software Android) gerando, então, arquivos .dex (*Dalvik Executable*, executável *Dalvik*) que serão executados pela máquina virtual *Dalvik*. Ainda segundo Monteiro (2012, p. 38), “Depois de criado o arquivo .dex e todos os recursos utilizados na aplicação, como imagens e ícones, são adicionados em um

arquivo .apk , que é o aplicativo propriamente dito, capaz de ser instalado em um dispositivo.” A **Figura 2.4** resume o processo de criação do APK (*Android Application Package*, Pacote de Aplicativo Android).

Figura 2.4 – Processo de criação do APK



Fonte: Monteiro (2012, p. 38)

Kernel Linux: O Android utiliza o *Kernel* do Linux para execução de serviços solicitados por aplicações, porém não se trata de um sistema embarcado, este Linux é responsável pela gerência dos recursos primários do sistema, como *threads*, memória, processos, segurança do sistema de arquivos e *drivers* de hardware (FERNANDES & LAURINDO, 2011). Cada execução de uma aplicação gera um novo processo no sistema, se necessário por falta de memória ou por inatividade do processo, o Android pode encerrar esses processos (FERNANDES & LAURINDO, 2011).

Ao passar dos anos os recursos disponibilizados nos *smartphones* aumentaram significativamente, sensores dos mais diversos tipos passaram a ser embutidos nos mais variados modelos de dispositivos. De acordo com Praciano (2015) os aparelhos Android dispõem de sensores que os permite perceberem movimentos e a orientação do dispositivo, além de conseguirem obter várias informações sobre as condições do ambiente que os cerca.

Ainda segundo Praciano (2015) estes sensores podem oferecer dados precisos aos aplicativos que os interpretam, permitindo-os monitorar o ambiente e obter suas condições atuais e alterações, por exemplo, um aplicativo de jogo pode monitorar o sensor de gravidade para calcular os gestos e movimentos do usuário, já um aplicativo de meteorologia pode usar o sensor de temperatura e de humidade para calcular e informar o ponto de orvalho (momento em que o vapor de água presente no ar ambiente passa ao estado líquido na forma de pequenas gotas por via da condensação, o chamado orvalho).

O **Quadro 2.1** contem a maioria dos sensores suportados pela plataforma Android, eles são do tipo SW (software) ou HW (hardware) ou ambos:

Quadro 2.1 – Sensores suportados pela plataforma Android

Sensor	Tipo	Descrição	Usos
Acelerômetro	HW	Mede a força da aceleração em m/s^2 aplicada ao dispositivo em todos os três eixos físicos (x, y e z), incluindo a força da gravidade.	Deteção de movimento (ao chacoalhar, ao bater, etc.).
Termômetro (temperatura ambiente)	HW	Mede a temperatura em graus Celsius.	Monitoramento da temperatura do ambiente.
Gravidade	HW e SW	Mede a gravidade em m/s^2 aplicada a um dispositivo em todos os eixos físicos.	Deteção de movimento (chacoalho, batida, toque, etc.).
Giroscópio	HW	Analisa a rotação em rad/s em torno de cada um dos 3 eixos.	Deteção da rotação (giro, virada, etc.).
Luz	HW	Detecta e analisa a intensidade da iluminação ambiente em lx.	Adaptar o brilho da tela em função da iluminação local.
Aceleração linear	HW e SW	Mede a aceleração em m/s^2 aplicada ao aparelho em todos os 3 eixos físicos (x, y e z), excluía a força da gravidade.	Monitoramento da aceleração ao longo de um único eixo.
Campo magnético	HW	Mede os valores do campo magnético ao redor do dispositivo relativo a todos os 3 eixos em μT .	Criar uma bússola.
Orientação	SW	Mede graduação da rotação que o dispositivo faz em torno dos 3 eixos físicos. Através de uma API (<i>Application Programming Interface</i> , Interface de Programação de Aplicação), o desenvolvedor pode obter dados da matriz de inclinação e de rotação, com o uso do sensor de gravidade associado ao sensor de campos magnéticos.	Determinar a posição do aparelho.
Pressão	HW	Mede a pressão ambiente do ar em hPa ou mbar.	Monitorar as alterações na pressão atmosférica.
Proximidade	HW	Mede a proximidade em relação a um objeto em cm a partir da tela.	Determinar se o smartphone está próximo ao ouvido/rosto do usuário.
Umidade relativa	HW	Mede a umidade relativa do ambiente em percentuais (%).	Monitorar o ponto de orvalho, absoluto e umidade relativa.
Vetor de rotação	SW e HW	Mede a orientação de um dispositivo, providenciando os 3 elementos do seu vetor de rotação.	Deteção de movimento e de rotação.
Temperatura	HW	Mede a temperatura do dispositivo em graus Celsius. Este sensor varia entre os diversos dispositivos Android e tem sido substituído pelos fabricantes por monitor de temperatura ambiente.	Monitorar temperaturas.

Fonte: adaptado de Praciano (2015).

Outros recursos bastante relevantes presentes na grande maioria dos *smartphones* são os tipos de conexões (*bluetooth*, *wi-fi*, *wi-fi direct*, NFC (*Near Field Communication*, Comunicação por Campo de Proximidade), etc.) que possibilitam formas simples e eficientes de troca de dados sem fio, e também o GPS (*Global Positioning System*, Sistema de Posicionamento Global) que fornece a posição e altitude instantâneas que o dispositivo se encontra em qualquer parte do planeta, que utilizado por diversos aplicativos.

2.4 Trabalhos relacionados

2.4.1 Aplicativo Em Android Para Controle De Unidades Robóticas Móveis Com Arduino

Trabalho de Conclusão de Curso desenvolvido na Universidade Federal do Piauí – UFPI, pelo discente Allan Jheyson Ramos Gonçalves no ano de 2013. É abordado o desenvolvimento de carro robô que utiliza a tecnologia livre Arduino para gerenciar os atuadores e motores responsáveis pela locomoção deste. É possível realizar movimentos de ir para frente, para trás, esquerda e direita, sendo que estas tarefas são acionadas por meio de um *smartphone* Android que, através das tecnologias *bluetooth* ou *wi-fi*, estabelece uma conexão com o Arduino que controla os motores.

Um dos principais objetivos do trabalho era conseguir manter uma comunicação estável entre o sistema Android a plataforma Arduino por meio das duas tecnologias citadas acima, os testes iniciais foram feitos e comprovou-se uma eficiente comunicação após alguns ajustes. Foram realizados dois testes, um para cada tipo de conexão utilizada, com *bluetooth* o teste foi realizado ao ar livre no campus da universidade e a comunicação mostrou-se satisfatório até 50 metros de distância, já o teste com *wi-fi* foi realizado somente em um ambiente fechado de um laboratório e mostrou-se bastante efetivo.

A principal contribuição desse trabalho está relacionada à comunicação realizada e testada em ambientes reais e que envolveu dois tipos de tecnologias livres, bem divulgadas e utilizadas mundialmente, *wi-fi* e *bluetooth*. Porém no trabalho de Gonçalves o *smartphone* é

utilizado simplesmente como um controle remoto, diferente do que ocorre neste, onde o *smartphone* é uma central de sensores, de comunicação e de processamento.

2.4.2 Big Dog

Trabalho consiste no desenvolvimento de um robô militar quadrúpede para transporte de cargas pesadas, que consegue se movimentar satisfatoriamente mesmo transportando volumes que podem chegar até 154 kg. O projeto tem como objetivo principal a criação de um robô que anda, corre, sobe por inclinações, ultrapassa obstáculos impostos pelo tipo de terreno e tenta ao máximo manter-se em equilíbrio e caminhando mesmo sendo exposto a situações adversas.

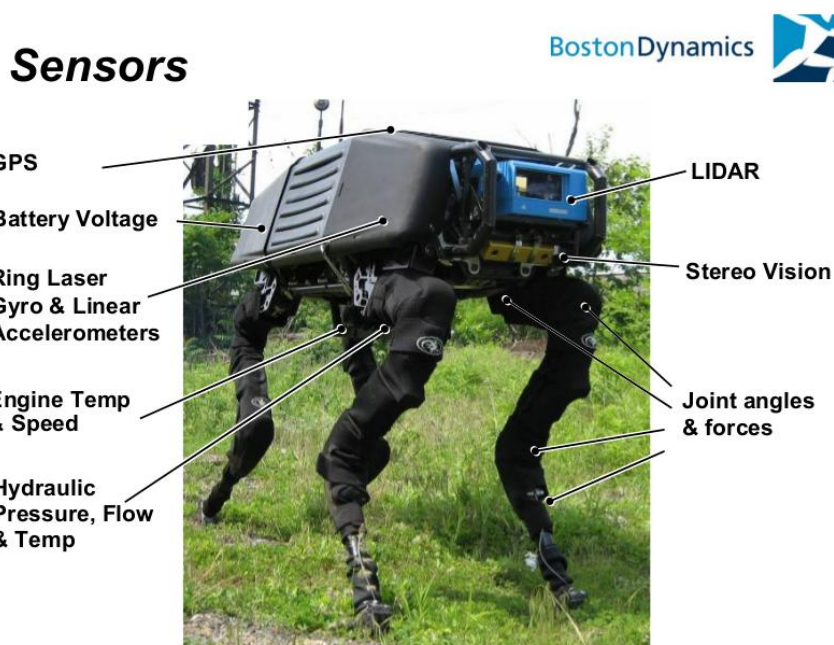
Foi construído pela Boston Dynamics, uma empresa de engenharia robótica famosa pela construção de robôs avançados caracterizados por mobilidade, agilidade, destreza e velocidade. É localizada nos EUA e desenvolve tecnologias para fins militares.

O projeto inicial foi financiado pela DARPA (*Defense Advanced Research Projects Agency*, Agência de Projetos de Pesquisa Avançada de Defesa) foi criada em fevereiro de 1958 por militares e pesquisadores americanos, numa reação dos Estados Unidos à vitória tecnológica da então União Soviética com o lançamento do primeiro satélite artificial, o Sputnik 1 (DARPA, 2015). Outras fases do projeto também foram financiadas pelo Laboratório de Pesquisa do Exército.

O Big Dog é alimentado por um motor à combustão que impulsiona um sistema de acionamento hidráulico que coordena os atuadores permitindo as quatro pernas se articular como um animal, com características de absorção do impacto ao pisar no solo e reciclagem da energia de um passo para o próximo.

Ele tem cerca de 1 m de altura, 1,1 m de comprimento, pesa 109 kg e pode movimentar-se em velocidade de até 6 km/h. É capaz de subir inclinações de até 35 graus, atravessar escombros e caminhar na neve ou terrenos lamacentos. Veja os principais sensores e outros componentes do Big Dog na **Figura 2.5**.

Figura 2.5 – Principais sensores e outros componentes do Big Dog



Fonte: Boston Dynamics (2008)

Para manter todo o sistema em funcionamento são utilizados diversos sensores espalhados por locais estratégicos, desde um sensor de força localizado logo acima de cada pé, que lembra um tornozelo, até os que atuam no sistema hidráulico para monitorar a pressão e temperatura do óleo utilizado ou em outros componentes internos como a bateria. Os sensores de locomoção e equilíbrio incluem: posição conjunta, força conjunta, contato com o solo, um giroscópio e um acelerômetro.

Dispõe de um sistema de visão estéreo composto por: duas câmeras, um computador e um Software específico de visão, que é utilizado para localizar e identificar objetos presentes no ambiente. Possui também um LiDAR (*Light Detection And Ranging*, detecção de luz e variância), é uma tecnologia semelhante ao radar, porém, ao contrário deste, que emite ondas de rádio para detecção de características, o LiDAR emite pulsos de luz por *laser* e, no Big Dog, é usado principalmente para permiti-lo acompanhar a trajetória de locomoção de um ser humano, sendo este o seu líder, o qual usa um marcador retro reflexivo e é seguido a uma distância fixa, baseando-se na velocidade do mesmo.

A integração e processamento dos dados e o estabelecimento da comunicação com o usuário são feitos por meio de um computador de bordo, que recebe os comandos de um controle remoto à distância para definir a trajetória do movimento, as informações provenientes dos sensores também são armazenadas para gerar estatísticas de desempenho e falhas, bem como, apoio operacional.

Os comandos enviados pelo controlador são de alto nível, isso significa que eles somente especificam, por exemplo, a direção, a velocidade e o modo de locomoção do robô, enquanto a parte de análise de solo, equilíbrio e adequação é feita de forma automática pelo conjunto de componentes do sistema.

O projeto Big Dog vem a contribuir com este trabalho no que se refere à utilização do acelerômetro e giroscópio (sensores que podem ser encontrados na maioria dos *smartphones*) para manter-se em equilíbrio no transporte de cargas, mesmo sendo exposto a ambientes irregulares e instáveis. O fato do uso da comunicação a distância para controle de direção e movimentos referentes à locomoção serem separados do controle de equilíbrio, ou seja, o equilíbrio do dispositivo e consequentemente a preservação da carga ser totalmente independente não importando a direção tomada, é uma técnica bastante notável que está presente também no desenvolvimento deste trabalho.

3 Metodologia

3.1 Desenvolvimento para Android

Esta fase tem início com um estudo para adquirir habilidades para o desenvolvimento de aplicativos direcionados à plataforma Android. A linguagem de programação oficial, ou seja, Java foi preferida, levando-se em consideração a abrangente documentação e suporte disponíveis oficialmente para plataforma em questão; a robustez, os recursos existentes e o fato desta ser considerada uma linguagem de alto nível, isto é, a existência de um grau de abstração relativamente elevado, deixando o código de máquina, em si, o mais distante possível e aproximando-se da linguagem humana, também foram levados em conta.

Requisitos e ferramentas para o desenvolvimento:

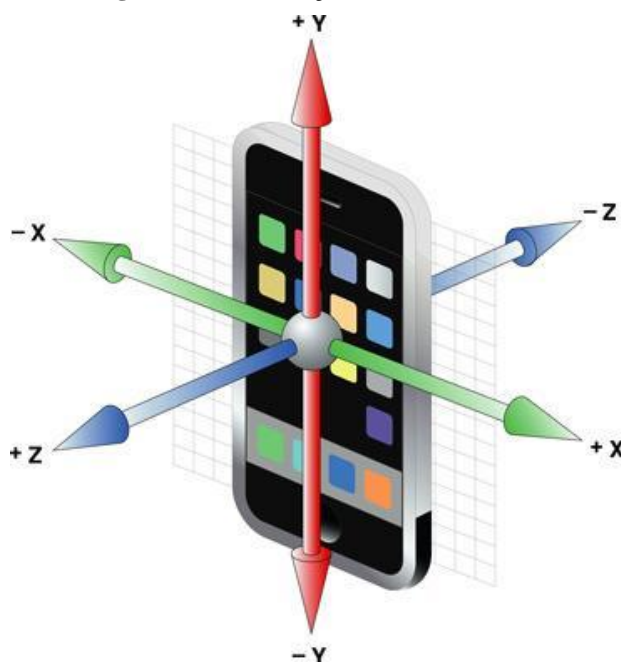
- SDK Android (*Software Development Kit* Android, Kit de Desenvolvimento de Software Android), que inclui todas as bibliotecas, ferramentas, compiladores, máquina virtual Dalvik e outros recursos necessários à criação de aplicativos para plataforma Android;

- IDE (*Integrated Development Environment*, Ambiente de Desenvolvimento Integrado), é um programa de computador que objetiva apoiar o desenvolvimento de software, facilitando e agilizando o processo, pois, é constituído de editores inteligentes, com análise e detecção automática de possíveis erros na codificação, gerenciamento dos arquivos e recursos de vários projetos, bem como, integração com os compiladores e/ou interpretadores das linguagens utilizadas. Na Internet existem variados IDEs direcionados a inúmeros tipos de linguagens, assim como para o desenvolvimento Android. O Eclipse é um dos mais populares, é livre, de código aberto e personalizável às necessidades de cada desenvolvedor ou situação por meio de um sistema de *plug-ins*, que são criados por pessoas em diferentes partes do mundo. Contudo neste trabalho é utilizado o Android Studio, que foi baseado no IntelliJ IDEA, um IDE para programação Java idealizado pela JetBrains (empresa especializada na criação de IDEs). A escolha do Android Studio consiste no fato deste ter sido adotado oficialmente como o ambiente de desenvolvimento Android pela Google e também desfruta de uma expressiva quantidade de funcionalidades, além de que, este se mostrou mais estável e leve em comparação com o Eclipse em testes realizados previamente no computador usado neste projeto, que executa um sistema operacional Linux.

Após estudo e entendimento mínimo dos conceitos relacionados à programação Android, o desenvolvimento enfatiza duas funcionalidades principais, a primeira refere-se ao estabelecimento da comunicação sem fio, que neste caso preferiu-se usar *wi-fi*, entre o smartphone e outro dispositivo (*notebook*) que realizará o controle de algumas funcionalidades disponíveis no projeto, como por exemplo, ativar e desativar a captura de sensores, bem como o salvamento de seus dados para análises futuras.

A segunda funcionalidade é relacionada à parte do aplicativo responsável pelo gerenciamento dos sensores. Neste projeto é usado principalmente o acelerômetro, que está presente na maioria dos *smartphones* da atualidade. O acelerômetro é o sensor que permite identificar a orientação física do dispositivo, ou seja, através dele consegue-se saber quando um dispositivo está inclinado para a esquerda ou direita, para frente ou para trás, pra baixo ou para cima. Como se podem perceber na **Figura 3.1** os dados do acelerômetro provem dos valores de X, Y e Z, que formam um plano cartesiano virtual, sendo que em ambos os eixos ocorre uma variação de -10 a 10 dependendo do posicionamento.

Figura 3.1 – Ilustração de acelerômetro



Fonte: NASCIMENTO (2012)

A cada oscilação os valores são analisados, processados e encaminhados, via USB (*Universal Serial Bus*, Barramento Serial Universal), ao Arduino, que é responsável pelo gerenciamento dos atuadores (motores), que modificam o posicionamento da plataforma.

3.2 Desenvolvimento para Arduino

Esta etapa também se inicia com um estudo dos conceitos básicos relacionados ao desenvolvimento para Arduino, que envolve programação de *firmwares*, que são instruções operacionais programadas diretamente no hardware de um equipamento eletrônico, sendo armazenadas permanentemente num circuito integrado (chip) de memória.

Em seguida concentra-se, principalmente, na implementação de duas funcionalidades, a primeira é relacionada à comunicação com *smartphone*, que é feita através da porta USB, a segunda refere-se ao controle dos atuadores, que são responsáveis pelo nivelamento da plataforma.

No desenvolvimento para o Arduino optou-se por utilizar a linguagem oficial, ela é baseada na linguagem da plataforma de prototipagem *Wiring*, muito similar às linguagens C e C++. A escolha é fundamentada no fato de existir várias bibliotecas, disponíveis oficialmente,

que podem ser utilizadas em diferentes contextos ou necessidades, além do suporte *online* mais acessível e também por suprir todos os requisitos do projeto.

O ambiente para codificação selecionado também é o oficial e chama-se Arduino, mesmo nome da plataforma e pode ser obtido através da Internet no próprio site do fabricante. Caracteriza-se por ter uma interface gráfica simples, intuitiva e possuir recursos relevantes, com, por exemplo, a importação automática das bibliotecas pré-configuradas por seus desenvolvedores. Foi desenvolvido em sua maioria na linguagem de programação Java, o que lhe confere um alto nível de portabilidade, estando disponíveis instaladores para uso em diversas plataformas, como Windows, Linux e Mac.

3.3 Comunicação entre Android e Arduino

A comunicação entre as duas plataformas é por meio da tecnologia USB, servindo-se da conexão disponível na placa Arduino e da entrada micro USB encontrada nos *smartphones* Android que, normalmente, é utilizada para transferência de dados e carregamento da bateria. Entretanto, para que a comunicação se torne viável, é necessário o uso de um adaptador USB OTG (*On The Go*) conectado ao *smartphone*, para converter a entrada micro para USB padrão, o que torna possível conectar dispositivos tais como pen drives, HD externo, mouse ou teclado, entre outros, direto no *smartphone*. Na **Figura 3.2** observa-se a estrutura de conexão entre um Arduino e um *smartphone*, o qual faz uso de um adaptador OTG.

Figura 3.2 – Conexão entre um *smartphone* e um Arduino



Fonte: Cerbo (2012).

Esse tipo de conexão foi escolhido levando em consideração sua eficácia, rapidez e a baixa taxa de erros na transferência dos dados, características que são extremamente relevantes para a estrutura em questão, visto que esta depende do menor tempo de resposta possível dos atuadores (motores) para atingir um resultado satisfatório.

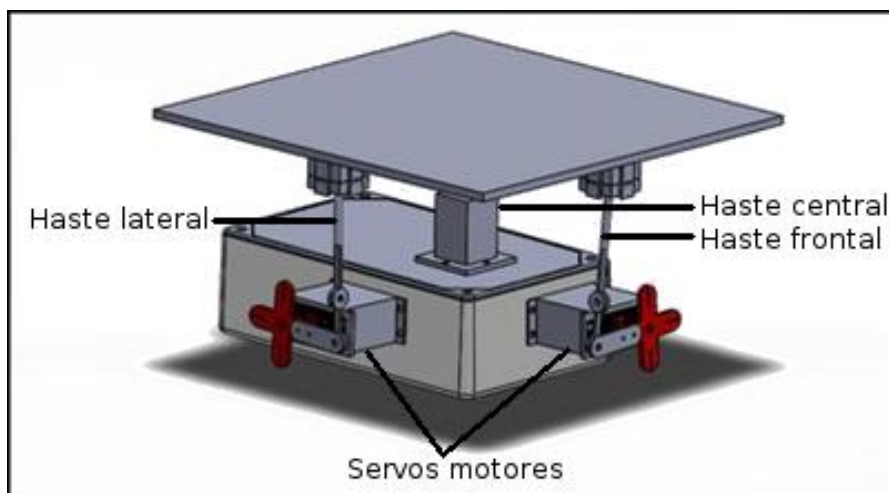
3.4 Confeção da Plataforma

A plataforma é composta basicamente por duas chapas sólidas, inferior e superior, sendo ambas em formato retangular e posicionadas horizontalmente. A superior destina-se à acomodação da carga e é sustentada por hastes móveis verticalmente colocadas desde a chapa inferior e acionadas por servos motores (motores de giro controlado eletricamente, de modo que permitem posicionar seu eixo em um ângulo específico variável de 0 a 180 graus) para mantê-la nivelada.

Na chapa inferior estão fixados os servos motores responsáveis pelo nivelamento da superior mediante o hasteamento móvel, bem como o *smartphone*, fixo horizontalmente sobre a mesma, de forma que acompanhe fielmente os seus movimentos oscilatórios e inclinações. Os dados provenientes dos sensores do *smartphone* nessa movimentação são processados e enviados, via USB, à placa Arduino (também presa à chapa inferior), que, posteriormente altera os ângulos dos servos motores para manter o nivelamento da chapa superior.

O modelo usado para montagem dos componentes possui uma haste central, vertical e fixada entre as duas chapas, inferior e superior, porém, o topo da haste conta com uma esfera móvel acoplada que permite a chapa superior inclinar-se para qualquer direção. As inclinações são controladas por duas outras hastes que se movem verticalmente por intermédio de servos motores, uma delas se localiza na parte frontal e a outra na lateral da plataforma, configurando-se, assim, um triângulo de sustentação para chapa superior e uma estrutura que a permite realizar os movimentos necessários ao seu nivelamento. Veja um exemplo da utilização desse modelo na **Figura 3.3**.

Figura 3.3 – Exemplo de modelo com triângulo de sustentação para plataforma



Fonte: adaptado de Benedict (2015)

Os materiais com quais são confeccionadas as duas chapas é o MDF (*Medium-Density Fiberboard*, placa de fibra de madeira de média densidade), ele é muito utilizado atualmente na fabricação de móveis. As hastes são elaboradas com algum tipo de metal (ferro) maleável, porém resistente.

A alimentação elétrica dos componentes pode ser realizada por meio da conexão USB entre a placa Arduino e o *smartphone*, que dispõe de uma alimentação de 5 V e corrente de 500 mA, bem como por meio de uma bateria ou uma fonte CC (Corrente Contínua) conectada à tomada.

3.5 Testes

Os diversos testes realizados para analisar a funcionalidade do projeto caracteriza a pesquisa como experimental. Os testes são iniciados colocando primeiramente um objeto sólido, como por exemplo, um cubo de madeira ou plástico, sobre a chapa superior e realizar algumas inclinações na inferior para observar o quanto esse objeto sofre desnivelamento.

Um experimento mais preciso é feito utilizando um segundo *smartphone* que é fixado horizontalmente na chapa superior executando um aplicativo que usa o acelerômetro para analisar as oscilações, sendo assim quanto menor a variação nos dados coletados pelo segundo *smartphone*, melhor comprovação funcional do mecanismo.

O teste de conexão entre o *smartphone* e o dispositivo de controle de locomoção é realizado também durante os descritos anteriormente, sendo que a distância entre os mesmos é alterada para análise de diferentes contextos.

4 Desenvolvimento do projeto

4.1 Desenvolvimento para Android

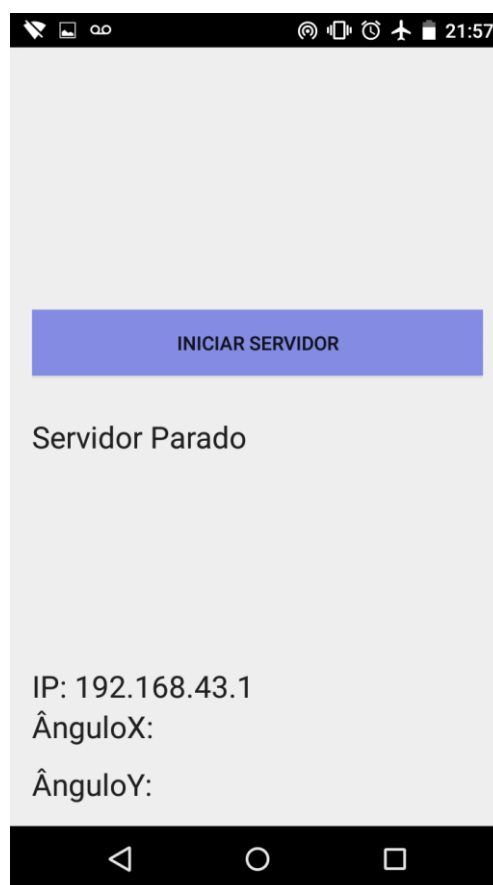
Neste projeto foram desenvolvidos dois aplicativos Android, o primeiro é essencial para o funcionamento da plataforma, sendo utilizado no gerenciamento dos sensores, processamento de dados, interação com o Arduino e comunicação com um dispositivo de controle, o *smartphone* no qual ele é instalado é fixado na chapa inferior e é denominado aferidor.

O segundo aplicativo é utilizado especialmente para os testes, sendo que o dispositivo no qual ele é instalado denomina-se *smartphone* carga e é colocado na chapa superior da plataforma enquanto que o aferidor sofre algumas oscilações na chapa inferior, com isso é possível analisar, por meio dos sensores do *smartphone* carga, o quanto este sofre alterações.

O aplicativo aferidor dispõem de servidor *socket* (um tipo de canal de comunicação entre processos via rede) que é ativado por meio de um clique em um botão encontrado na tela inicial do aplicativo. Uma vez que o servidor foi iniciado pode-se estabelecer uma conexão entre este e o software instalado no dispositivo de controle, que neste projeto é um notebook, do qual é possível realizar algumas tarefas como: ativar e desativar a captura de dados do acelerômetro bem como definir o salvamento desses dados para geração de estatísticas.

A rede usada para conexão é a *wi-fi*, onde atualmente qualquer *smartphone* Android possui a funcionalidade de criar um *hotspot wi-fi*, isto é, um ponto de acesso em que diferentes dispositivos que disponham de tecnologia *wi-fi* podem conectar-se, obter um IP e até mesmo acessar Internet, caso o aparelho no qual estiver o *hotspot* tenha outra conexão, como por exemplo, 3G. Veja na **Figura 4.1** a tela inicial do aplicativo aferidor.

Figura 4.1 – Tela inicial do aplicativo aferidor



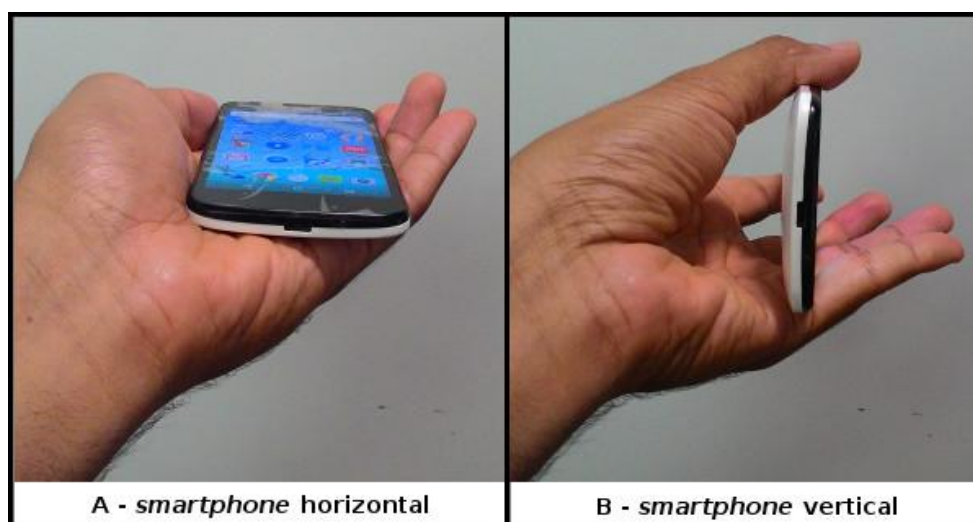
Fonte: elaborado pelo autor (2015)

Uma vez que aplicativo do aferidor é executado percebe-se que ele tem somente um botão para ativar o servidor *socket*, um texto informando o seu *status*, que é alterado para ‘Servidor iniciado’ no momento em que o botão é acionado e um segundo texto mostrando o IP (*Internet Protocol*, Protocolo de Internet) vinculado ao *smartphone* naquele momento; este IP é usado no software de controle para estabelecimento da comunicação com este servidor *socket*.

Estas outras duas informações, ÂnguloX e ÂnguloY, referem-se aos ângulos obtidos a partir dos dados x e y do acelerômetro, que podem variar no intervalo de -10 a 10 m/s² por causa da aceleração gravitacional, do qual é utilizado o intervalo de -2,65 a 2,65 m/s² que são convertidos para ângulos entre 0 e 180 graus respectivamente, os quais são enviados ao Arduino, que por sua vez modifica o ângulo dos dois servos motores responsáveis pelo nivelamento da plataforma, ou seja, o valor -2,65 m/s², ocorrido tanto em x quanto em y do acelerômetro, é convertido para o ângulo zero e aplicado ao seu respectivo servo motor; de forma análoga acontece com os valores intermediários ao intervalo e também com o 2,65 m/s², que é convertido para 180 graus.

Esse intervalo foi escolhido partindo da premissa que: o *smartphone* posicionado horizontalmente com a sua tela voltada para cima os valores de x e y do seu acelerômetro são iguais ou próximos do valor zero, porém se este *smartphone* for rotacionado 90 graus em torno de seu eixo y o valor do x é alterado para -10 ou 10 m/s², dependendo da direção da rotação. Veja **Figura 4.2** como exemplo, em **A** os valores de x e y são aproximadamente 0 (zero) m/s², porém em **B** o valor de x é alterado para cerca de -10 m/s².

Figura 4.2 – Giro de 90 graus do *smartphone*



Fonte: elaborado pelo autor (2015)

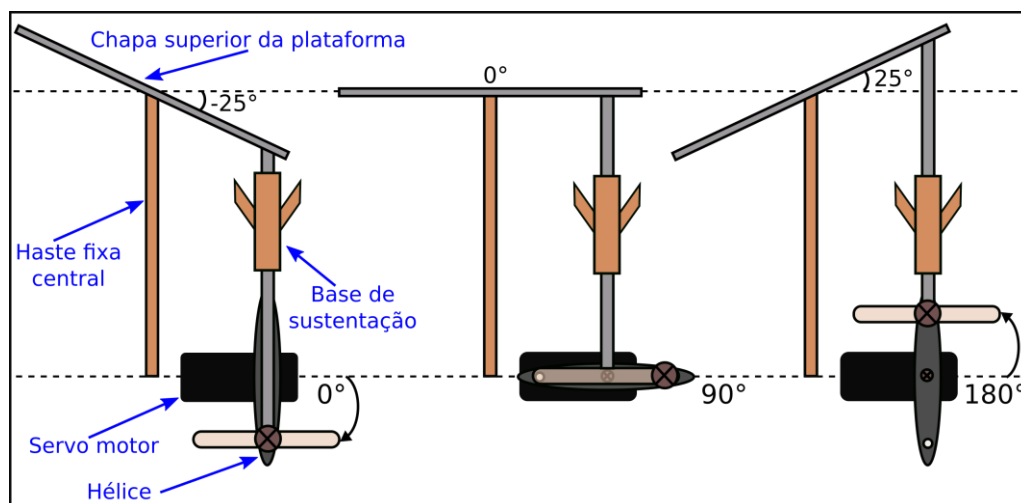
Como seria muito complicado ou desnecessário confeccionar um mecanismo que corrige um desnível de até 90 graus e também considerando a limitação imposta pela indisponibilidade dos recursos exigidos ao seu desenvolvimento, optou-se por utilizar um intervalo menor que o total oferecido pelo acelerômetro e com isso corrigir um desnível, de aproximadamente, até 25 graus ocorridos na plataforma a partir de sua posição inicial horizontal. Veja o **Quadro 4.1**, que contém a conversão dos valores e, a **Figura 4.3**, na qual se observa um esquemático com a relação entre os ângulos do servo motor e os da chapa superior da plataforma.

Quadro 4.1 – Conversão para ângulos

Intervalo do acelerômetro	Intervalo utilizado	Ângulos no servo motor	Ângulos na plataforma
-10	-2,65	0°	-25°
10	2,65	180°	25°

Fonte: elaborado pelo autor (2015)

Figura 4.3 – Relação entre os ângulos do servo motor e os da chapa superior da plataforma



Fonte: elaborado pelo autor (2015)

O tamanho da hélice usada no servo motor para movimentar verticalmente a haste também é um fator que influencia na escolha do intervalo de dados do acelerômetro, levando em consideração que: quanto maior a hélice, maior será o ângulo de correção de desnível e consequentemente é necessário o uso de um intervalo mais abrangente.

A frequência de dados do acelerômetro é de aproximadamente um novo dado a cada 10 ms, ou seja, 100 valores para cada eixo por segundo e, o processamento desta taxa de dados bem como a transmissão para o Arduino via USB, geraram travamentos causados pela baixa capacidade de processamento do Arduino se comparado ao *smartphone*, o que resultou em mau funcionamento do mecanismo como um todo. Em função disso, foram descartados alguns dados, sendo processados e transmitidos ao Arduino somente um deles a cada cinco gerados, ou seja, 20 por segundo, o que representa 20% do total.

A oscilação do acelerômetro é também um ponto a ser considerado, tendo em vista que este mantém certa variação mesmo que o *smartphone* esteja parado sobre uma superfície fixa e imóvel. Neste projeto é utilizado um *smartphone* modelo Moto G (1ª geração) da marca Motorola e a partir de testes percebeu-se que a variação nos dados do seu acelerômetro é de $0,25 \text{ m/s}^2$ para mais ou menos, isto é, mesmo que o *smartphone* seja colocado horizontalmente sobre uma superfície nivelada e imóvel seu acelerômetro estará variando entre 0 e $0,25 \text{ m/s}^2$ ou entre 0 e $-0,25 \text{ m/s}^2$ nos eixos x e y.

A partir dessas informações foram usadas três variáveis para controlar a oscilação do acelerômetro e envio de dados para o Arduino, são elas: *x_anterior*, *y_anterior* e *contador*, ambas iniciadas com valor zero. A cada novo valor do acelerômetro a variável *contador* é

incrementada em mais uma unidade, no momento em que esta atinge o valor cinco é verificado se a variação entre o valor atual de x e o seu valor anterior, $x_{anterior}$, ou se o valor atual de y e o seu valor anterior, $y_{anterior}$, é maior que $0,25 \text{ m/s}^2$ e caso verdade em algum dos dois casos, o novo valor é convertido em ângulo e enviado ao Arduino.

No caso da variação ser menor que $0,25 \text{ m/s}^2$ nenhuma informação é transmitida para o Arduino, porém os dados ainda podem ser salvos localmente ou enviados como *status* ao dispositivo de controle, do qual é possível o usuário ativar ou desativar essas duas opções, que operam de forma independente e são analisadas como se segue.

É verificado se a opção de salvar dados localmente para estatísticas futuras está ativa, caso sim, os dados são convertidos para ângulos, porém usando o intervalo de -10 a 10 m/s^2 do acelerômetro, que será convertido em ângulos de -90 a 90 graus, diferente do que ocorre na conversão do intervalo de $-2,65$ a $2,65 \text{ m/s}^2$ que é utilizado para o nivelamento da plataforma.

A forma de salvamento é em arquivos do tipo CSV, um tipo de arquivo de texto onde cada linha representa um registro de dados separados por um caractere especial que normalmente é um ponto-vírgula e que posteriormente pode ser aberto em um software de planilha eletrônica, como por exemplo, o Excel ou Libre Office Calc, para análises.

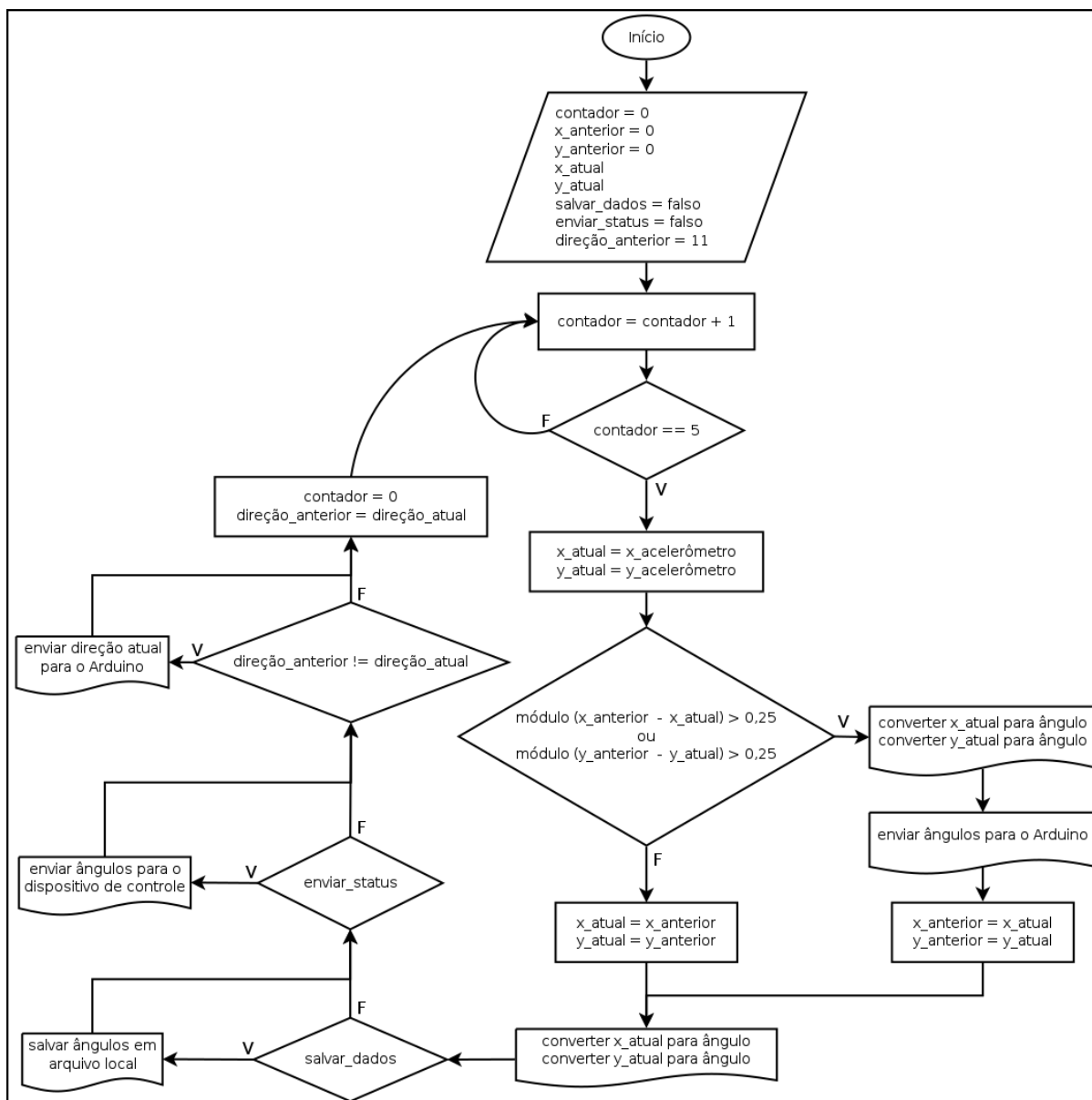
Essa característica do CSV é usada no projeto de modo que cada linha do arquivo gerado contém o registro do horário em horas, minutos e segundos, seguido do valor do ângulo x e do valor do ângulo y , ambos separados por ponto-vírgula. O arquivo gerado é nomeado com a palavra “Dados” seguida dos números referentes ao ano, mês, dia, horas, minutos e segundos separados por traço ‘-’ e salvo no diretório de trabalho do usuário, isto é, onde se encontram os pastas documentos, imagens, downloads, etc..

Em seguida, verifica-se a opção de envio de *status* em tempo real para o software de controle, caso esteja ativa, os ângulos obtidos a partir da conversão anterior são enviados via *wi-fi* para o dispositivo de controle que dispõe de um servidor *socket* para recebê-los e gerar um gráfico imediato para acompanhamento dos movimentos da plataforma.

Por último ainda é verificado se o usuário operando o dispositivo de controle mudou a direção de locomoção da plataforma, caso sim, esta nova informação é transmitida para o Arduino, que altera a direção de locomoção. O meio físico para locomoção, como por exemplo, um carro no qual a plataforma é colocada não foi desenvolvido no projeto, porém a implementação do algoritmo referente a essa funcionalidade já está implementada no software de controle, no aplicativo do *smartphone* aferidor e também no *firmware* do Arduino.

Ao final do ciclo a variável *contador* recebe valor zero para dar início uma nova contagem no próximo ciclo quando novos valores são gerados. Veja na **Figura 4.4** o fluxograma que representa a solução implementada. No **APÊNDICE A** se encontra também o código fonte do aplicativo aferidor, no qual a classe *ControlSensorEvent.java* tem a implementação do fluxograma apresentado na **Figura 4.4**.

Figura 4.4 – Fluxograma: algoritmo do *smartphone* aferidor



Fonte: elaborado pelo autor (2015)

O segundo aplicativo desenvolvido tem suas funcionalidades semelhantes ao primeiro, dispondo da mesma forma de salvamento de dados localmente e também envio de *status* para o controle. A **Figura 4.5** apresenta a tela inicial do aplicativo.

Figura 4.5 – Tela inicial do aplicativo Sensor Analyzer

Fonte: elaborado pelo autor (2015)

A primeira informação que se tem é a quantidade de sensores presentes no dispositivo, que no caso do smartphone usado no projeto são seis, logo abaixo se encontra uma lista com todos os sensores e informações individuais, como nome, tipo e fabricante, para cada um deles. Ao clicar no item da lista referente ao acelerômetro o usuário é direcionado para tela de configuração e *status*, a que pode ser visualizada na **Figura 4.6**.

Figura 4.6 – Tela de configuração e status

Fonte: elaborado pelo autor (2015)

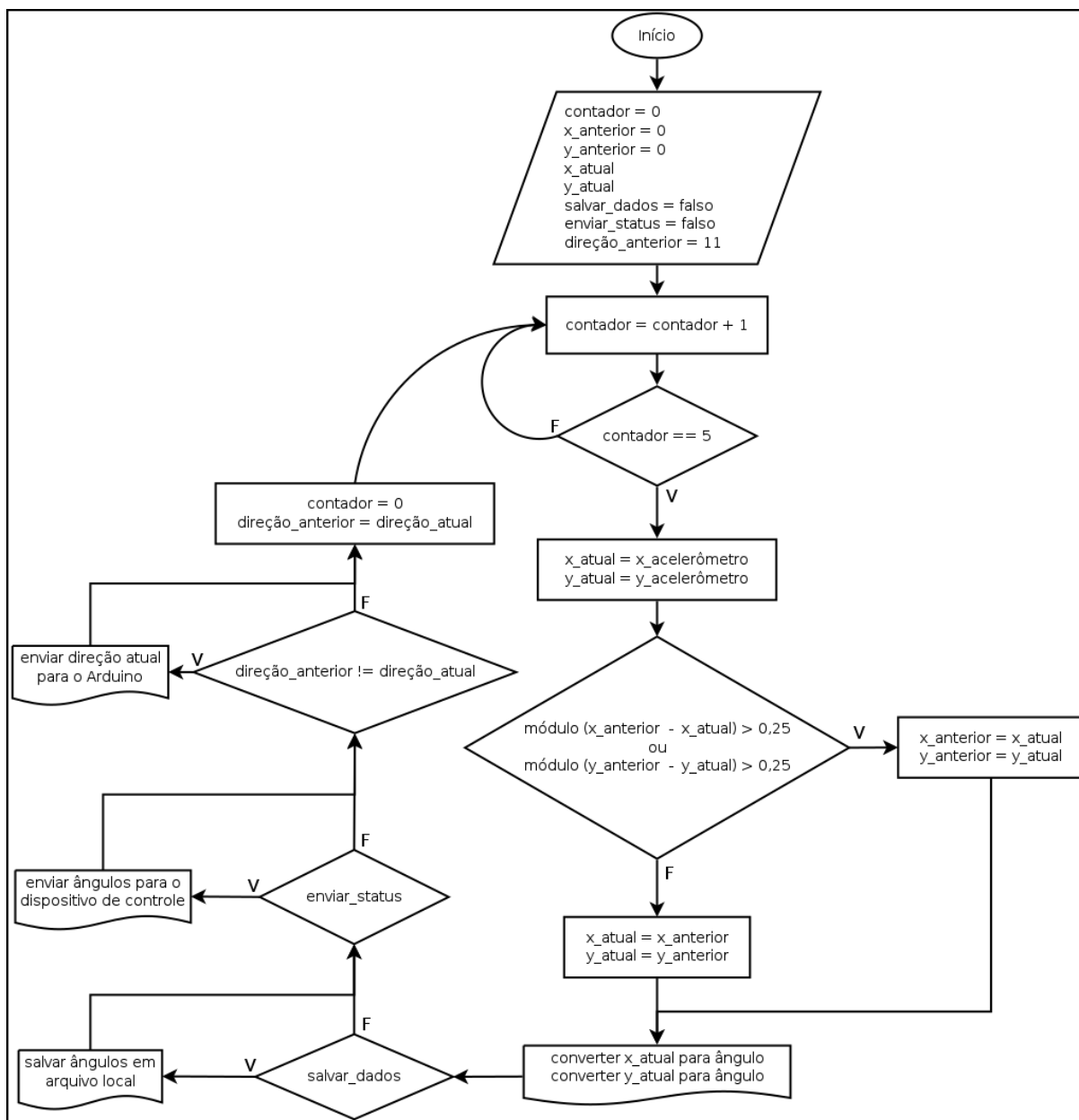
A tela vista na **Figura 4.6** é a principal do aplicativo, ela é dividida em três partes: a primeira é informativa e exibe em tempo real os valores dos ângulos x e y que serão salvos e/ou enviados ao controle; a segunda se refere à função de habilitar ou desabilitar o envio de *status* para o controle via *socket* por meio do IP informado no primeiro campo de texto; a terceira trata da funcionalidade de salvamento dos ângulos em arquivo CSV, seguindo mesmo modelo usado no aferidor, porém neste o usuário pode digitar o nome do arquivo que será gerado e salvo.

Ambas, as duas funcionalidades, são independentes, isto é, pode-se escolher desabilitar uma delas sem que isso interfira na execução da outra ou deixá-las habilitadas e acionar o botão ‘Iniciar’ para começar a captura dos dados como mostrado na **Figura 4.7**.

Figura 4.7 – Tela de configuração e status – execução

Fonte: elaborado pelo autor (2015)

O algoritmo usado neste aplicativo é bastante similar ao encontrado no aplicativo do *smartphone* aferidor, a diferença principal está no fato de que este não realiza comunicação com o Arduino, porém, a forma utilizada para capturar, converter e salvar os dados do acelerômetro é a mesma, veja **Figura 4.8**. No **APÊNDICE B** se encontra também o código fonte do aplicativo sensor analyzer, no qual a classe *ControlSensorEvent.java* tem a implementação do fluxograma apresentado na **Figura 4.8**.

Figura 4.8 – Fluxograma: algoritmo do *smartphone* carga

Fonte: elaborado pelo autor (2015)

4.2 Desenvolvimento para Arduino

O desenvolvimento para Arduino teve foco em duas funcionalidades principais, a primeira é relacionada à comunicação com *smartphone* através da porta USB e a segunda refere-se ao controle dos atuadores, que são dois servos motores responsáveis pelo nivelamento da plataforma.

Para a primeira se faz necessária a configuração de ambas as portas USBs, do Arduino e do *smartphone*, para um mesmo valor de taxa de dados, sendo este, 9600 bps, que foi escolhido por ser o padrão já utilizado no Arduino e por suprir as necessidades do projeto. Os dados são transmitidos em formato de *strings*, ou seja, cadeia de caracteres.

Já na segunda é usada uma biblioteca específica para controle de servo motor, ela é disponibilizada livremente pelos desenvolvedores da plataforma Arduino e já está pré-configurada no ambiente de desenvolvimento bastando apenas incluí-la no código do *firmware*.

Os *firmwares* desenvolvidos para Arduino normalmente seguem um modelo padrão preestabelecido no qual existem dois métodos: o *setup*, que é executado uma única vez no momento em que a placa é ligada e é geralmente utilizado para inicializar as variáveis e definir algumas opções de configurações necessárias ao funcionamento do *firmware*; já o segundo método, o *loop*, é executado e repetido de forma contínua enquanto o Arduino estiver ligado e nele são implementados os códigos relacionados ao objetivo a ser atingido, por exemplo, obter a temperatura instantânea de determinado ambiente por meio de um sensor específico.

Esse modelo padrão foi mantido no projeto de modo que no método *setup* são inicializadas as variáveis responsáveis pela recepção dos dados provenientes do *smartphone*, é realizada a configuração da porta USB e também configuração para definir quais são os pinos de controle que os servos motores estão ligados no Arduino bem como definição de seu ângulo inicial para 90 graus.

No método *loop* está implementado um código responsável por ficar verificando se tem algum dado na porta USB a cada ciclo de repetição e caso tenha ele é obtido, convertido e usado em diferentes atuadores na plataforma, dependendo do seu conteúdo, como é explicado no item 4.3.

4.3 Comunicação entre Android e Arduino

Para que a comunicação se tornasse possível foi necessário o acréscimo de uma biblioteca ao aplicativo Android. Essa biblioteca se chama *usb serial for android*, é livre e pode ser baixada na Internet através de um repositório no GitHub. Ela oferece uma forma

bastante simples para enviar e receber dados do Arduino no formato de conjuntos de bytes em *array*.

É sabido que um byte armazena somente um caractere e, por meio de testes, constatou-se que a cada ciclo de repetição do método *loop*, presente no Arduino, é lido também somente um byte, porém, os valores que devem ser transmitidos são compostos por mais de um caractere e destinados a diferentes atuadores.

Diante disso, elaborou-se a seguinte estratégia: cada transmissão é composta de um conjunto numérico, que representa um ângulo destinado a um dos servos motores ou uma opção de direção para locomoção, mais um caractere não numeral que informa em qual situação ou atuador o valor numérico deve ser aplicado e assim servindo como identificador.

Para a transmissão dos ângulos obtidos a partir da conversão dos valores do eixo x do acelerômetro o caractere ‘a’ é usado como identificador. No caso dos ângulos obtidos a partir de y é utilizado o ‘b’ para identificação e, de forma análoga, os valores referentes ao controle de locomoção são identificados pelo caractere ‘c’.

No *firmware* do Arduino são utilizadas duas variáveis para separar o valor do seu identificador, uma do tipo *character*, que a cada ciclo de repetição no método *loop*, recebe um caractere e logo após é verificado se este é igual a algum dos identificadores, isto é, ‘a’ ou ‘b’ ou ‘c’, se for diferente o caractere é concatenado na segunda variável, do tipo *String*, caso contrário, significa que todos os valores numéricos foram recebidos e estão armazenados e em consequência o seu identificador está na variável *character*.

Neste momento faz-se a conversão dos dados presentes na variável *String* para um único valor numeral inteiro e este é aplicado num específico servo motor referente ao seu identificador, por exemplo, caso forem lidos os caracteres ‘1’, ‘2’, ‘8’ e ‘a’ então o servo motor referente ao ângulo x terá seu eixo rotacionado para a posição 128.

O código fonte do *firmware* desenvolvido encontra-se no **APÊNDICE D**.

4.4 Desenvolvimento do controle

O software de controle é desenvolvido usando a linguagem Java e é usado para configurar as funcionalidades presentes no aplicativo aferidor, como explicado anteriormente no tópico **4.1**. A tela inicial do controle é visualizada na **Figura 4.9**.

Figura 4.9 – Tela inicial Controle Plataforma

A interface 'Controle Plataforma' apresenta um layout organizado em seções. No topo, há um campo de texto para 'IP do Smartphone' com o valor '192.168.0.100' e uma label para 'IP local para status' com o valor '192.168.0.102'. Abaixo, um checkbox 'Salvar Dados em Arquivo no Smartphone' está desativado. O centro da tela contém quatro botões: 'Iniciar captura de sensores' e 'Exibir Status' em azul, e 'Parar captura de sensores' e 'Fechar Status' em vermelho. Na base, um checkbox 'Usar teclas direcionais do teclado' também está desativado, e há cinco botões cinza representando as setas: 'Frente', 'Esquerda', 'Parar', 'Direita' e 'Atrás'.

Fonte: elaborado pelo autor (2015)

A tela inicial é dividida basicamente em três partes: a superior possui um campo de texto editável no qual é inserido o IP vinculado ao *smartphone* aferidor, que é o receptor das opções e/ou ações escolhidas e logo abaixo é apresentado o IP local do computador no qual está ocorrendo a execução, sendo este usado nos aplicativos Android para envio de *status*; na parte central encontram-se as opções para ativar ou desativar o salvamento de dados no *smartphone*, iniciar ou parar a captura de sensores assim como exibir ou fechar a tela de *status*; a parte inferior trata do controle de locomoção, realizado por meio de cliques nos botões disponibilizados ou usando teclado, caso seja marcada a opção de usar teclas direcionais do teclado.

Cada opção escolhida na tela inicial é associada com um ou dois dados, que servem como identificador da ação e são enviados via *wi-fi* ao servidor *socket* ativo no *smartphone* aferidor, que por sua vez realiza determinadas ações ou mudanças de acordo com os dados recebidos, veja **Quadro 4.2**.

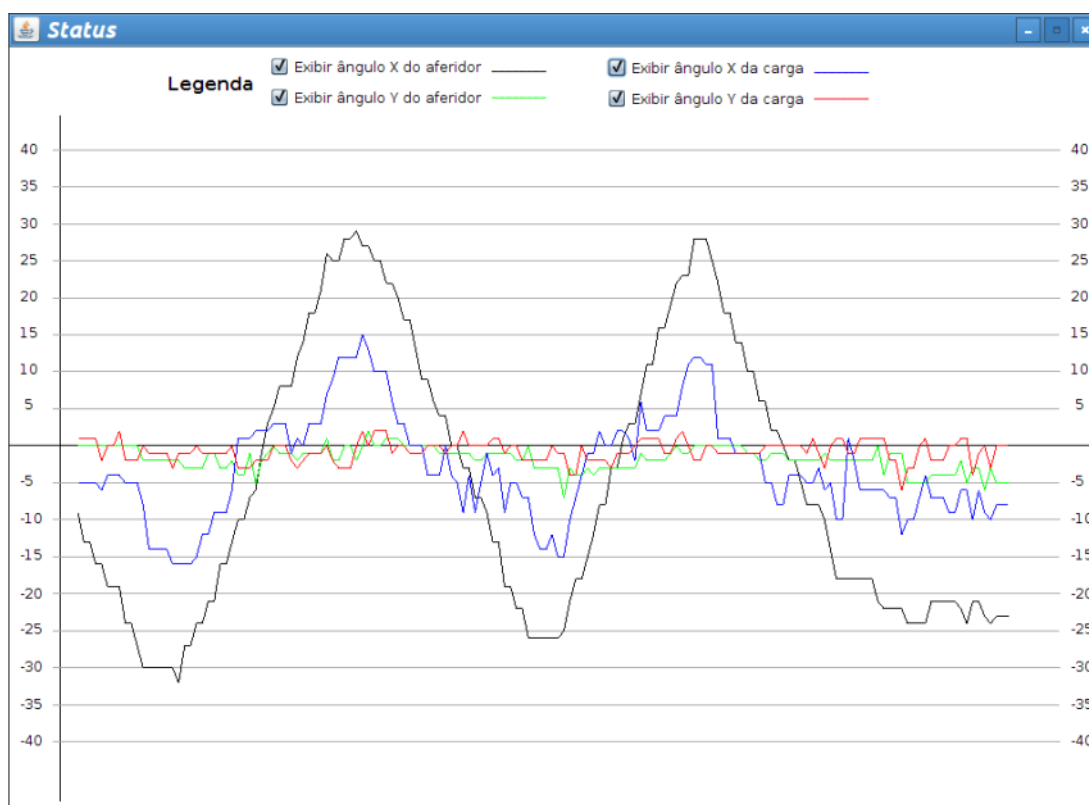
Quadro 4.2 – Opções de configurações do aferidor

Dados	Ação/mudança no aferidor	Acionamento
1	Ativar captura de sensores	Botão: iniciar captura de sensores
2	Desativar captura de sensores	Botão: parar captura de sensores
3	Ativar salvamento de ângulos em arquivo CSV localmente no celular	Checkbox (ativado): salvar dados em arquivo no <i>smartphone</i>
4	Desativar salvamento de ângulos	Checkbox (desativado): salvar dados em arquivo no <i>smartphone</i>
5 - IP local	Ativar envio de ângulos (<i>status</i>) em tempo real para o dispositivo de controle	Botão: exibir <i>status</i>
6	Desativar envio de ângulos (<i>status</i>)	Botão: fechar <i>status</i>
7	Locomover-se para frente	Botão: frente / Tecla direcional para cima
8	Locomover-se para direita	Botão: direita / Tecla direcional para direita
9	Locomover-se para trás	Botão: atrás / Tecla direcional para baixo
10	Locomover-se para esquerda	Botão: esquerda / Tecla direcional para esquerda
11	Parar locomoção	Botão: parar / Tecla enter

Fonte: elaborado pelo autor (2015)

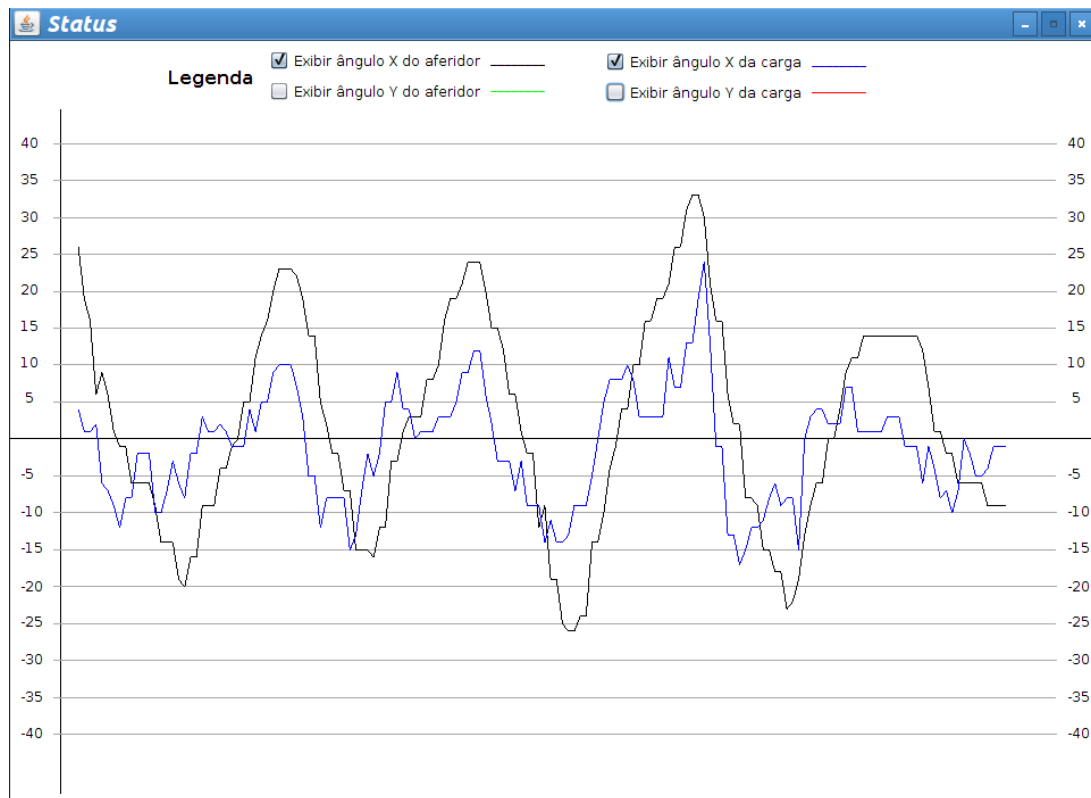
Todos os dados referentes às opções expostas no **Quadro 4.2** são enviadas por meio de um cliente *socket* e no caso do número 5, além de enviar o IP local para que o aferidor realize o emissão de *status*, são iniciados no controle dois servidores *socket* encarregados da recepção dos ângulos submetidos, tanto pelo próprio aferidor quanto pelo *smartphone* carga e, logo após é exibida a tela de *status*, a qual pode ser vista na **Figura 4.10**.

Figura 4.10 – Tela *status*: com ângulos x e y dos *smartphones* aferidor e carga



Fonte: elaborado pelo autor (2015)

A tela de *status* exibe um gráfico em tempo real dos ângulos relacionados a ambos *smartphones*, aferidor e carga, durante o funcionamento. Na legenda localizada na parte superior, cada valor correspondente aos ângulos dos eixos x e y do aferidor e do carga são identificados por cores e é possível escolher quais deles são exibidos no gráfico como visto na **Figura 4.11**.

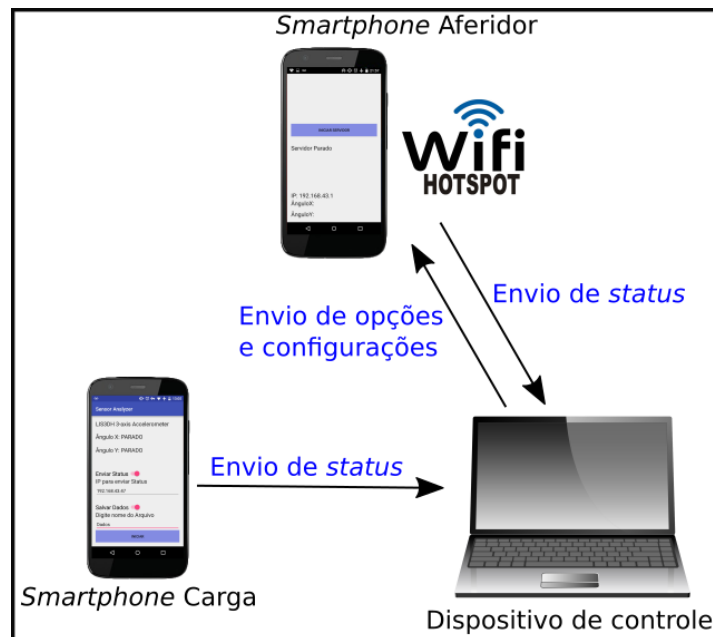
Figura 4.11 – Tela *status*: exemplo de configuração

Fonte: elaborado pelo autor (2015)

Para criação e exibição do gráfico presente na tela de *status*, bem como apresentar as variações em tempo real, fez-se necessário a utilização da biblioteca gráfica JOGL (Java *Open Graphics Library*, Biblioteca Gráfica Livre para Java), que pode ser obtida gratuitamente na Internet e importada para o projeto.

A comunicação sem fio entre os três softwares desenvolvidos está representada no esquema presente na **Figura 4.12**, onde na parte superior percebe-se o *smartphone* aferidor disponibilizando um *hotspot wi-fi* no qual o *smartphone* carga e o dispositivo de controle se conectam. Nota-se também que não existe transferência de dados entre os dois *smartphones* e que o carga se comunica com o controle somente para o envio de ângulos como *status*, enquanto que o aferidor envia ângulos como *status* e também recebe do controle as opções escolhidas pelo usuário.

Figura 4.12 – Esquema da comunicação sem fio entre os softwares desenvolvidos



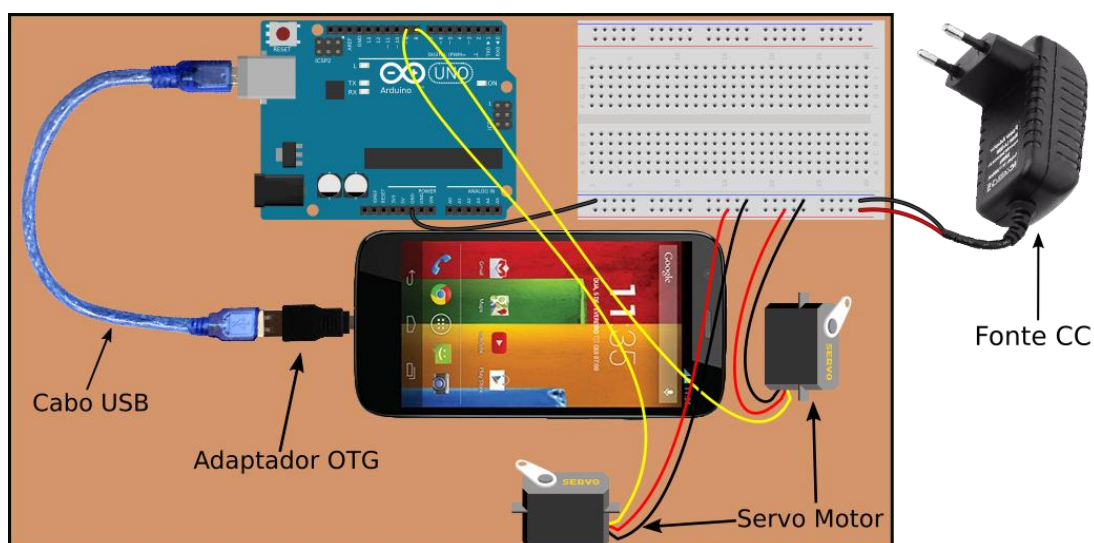
Fonte: elaborado pelo autor (2015)

4.5 Confeção da Plataforma

As chapas usadas na confecção são em MDF, com espessura de 3 mm e em formato retangular, sendo que as dimensões da inferior são 20 cm x 30 cm e a superior 20 cm x 24 cm e afastadas entre si por um espaço de 11 cm, por meio das hastes verticais, das quais, a lateral e a frontal estão localizadas 9 cm a partir do centro da plataforma.

Na inferior estão dispostos todos os componentes eletrônicos necessários ao funcionamento, veja na **Figura 4.13** o esquema elétrico utilizado, no qual a corrente elétrica positiva é representada pelos fios vermelhos e a negativa pelos pretos, já os fios amarelos representam o controle dos servos motores.

Figura 4.13 – Esquema elétrico com fonte CC

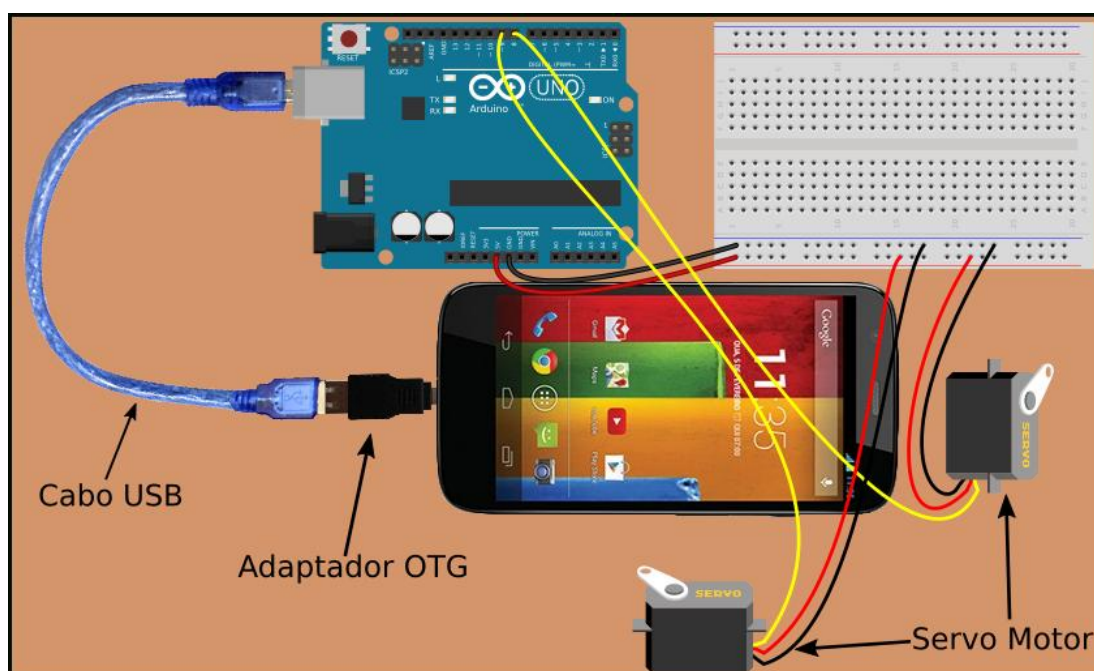


Fonte: elaborado pelo autor (2015)

Destaca-se a presença de uma fonte elétrica de corrente contínua para alimentação dos servos motores e também o *smartphone*, que além de ser utilizado nas funções já mencionadas, também, alimenta eletricamente a placa Arduino.

Entretanto, percebeu-se por meio de testes que a alimentação dos servos motores pode também ser suprida pelo próprio *smartphone*, levando em consideração que este disponibiliza uma alimentação de 5 V e 500 mA de corrente por meio da conexão micro USB e que, segundo o fabricante dos motores, cada um destes necessita de pelo menos 4,8 V e 150 mA para um funcionamento satisfatório, resultando em torno de 3 kg.cm de torque. Este modelo é exposto na **Figura 4.14**, na qual se utiliza o pino de saída de 5 V da placa Arduino para alimentar os atuadores.

Figura 4.14 – Esquema elétrico com fonte CC

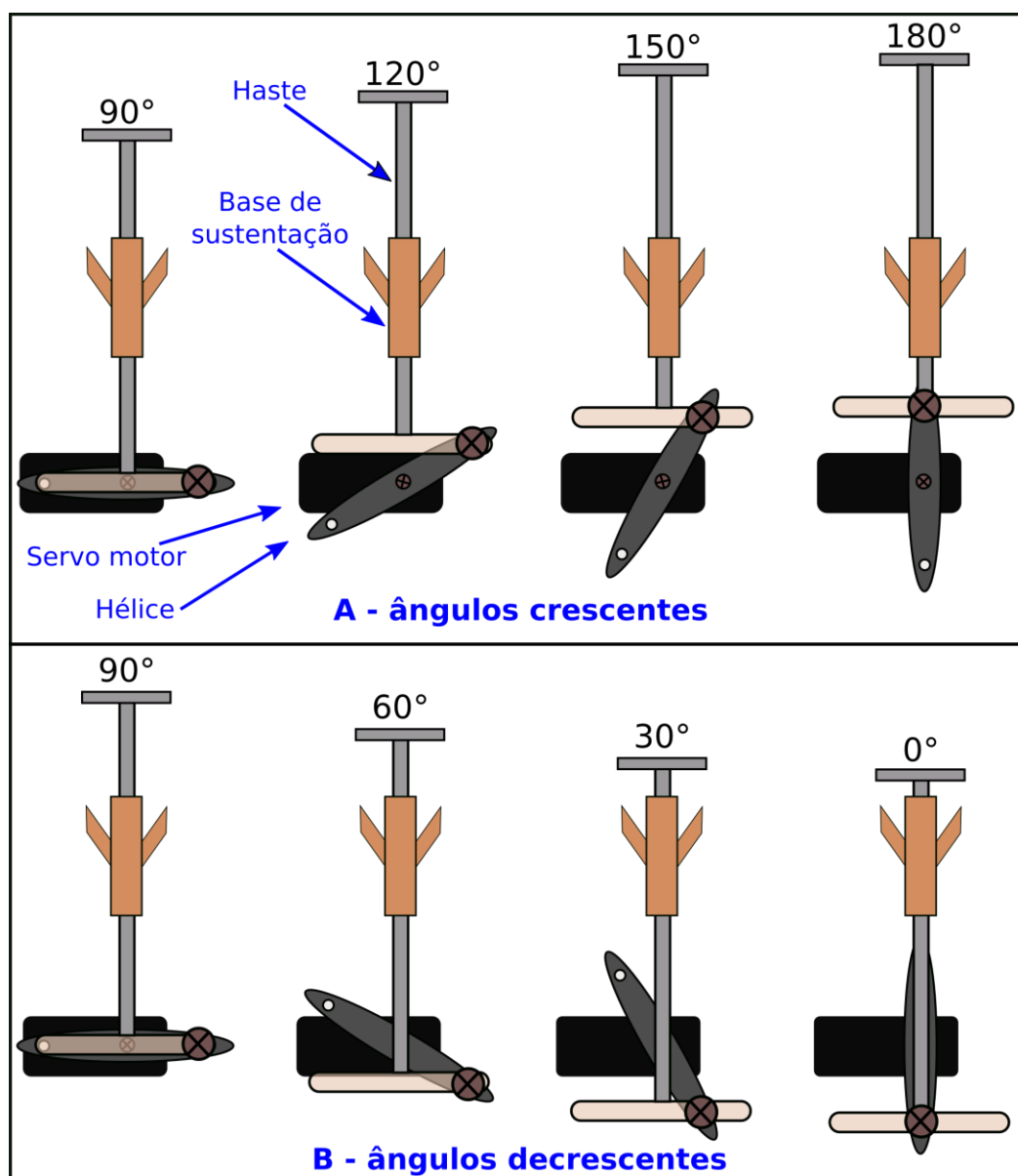


Fonte: elaborado pelo autor (2015)

Um dos desafios enfrentados foi a criação das hastes que ficam conectadas aos servos motores e que dão sustentação a chapa superior, haja vista que estas movimentam-se verticalmente, porém, sem sofrer deslocamentos horizontais ou tombamentos que poderiam comprometer o equilíbrio e/ou posicionamento da chapa superior.

Diante deste contexto elaborou-se a solução presente no modelo exposto na **Figura 4.15**, em que a alteração dos ângulos em graus do servo motor resulta na movimentação vertical da haste. Destacam-se as duas formas partindo do ângulo inicial 90° , onde em **A** é representado a elevação da haste por meio do aumento do ângulo e analogamente em **B** é apresentado o abaixamento da mesma com a diminuição do ângulo. Esse intervalo de 90° , apresentado em **A** e **B**, equivale aos 25° de correção de desnivelamento da plataforma.

Figura 4.15 – Funcionamento das hastes móveis



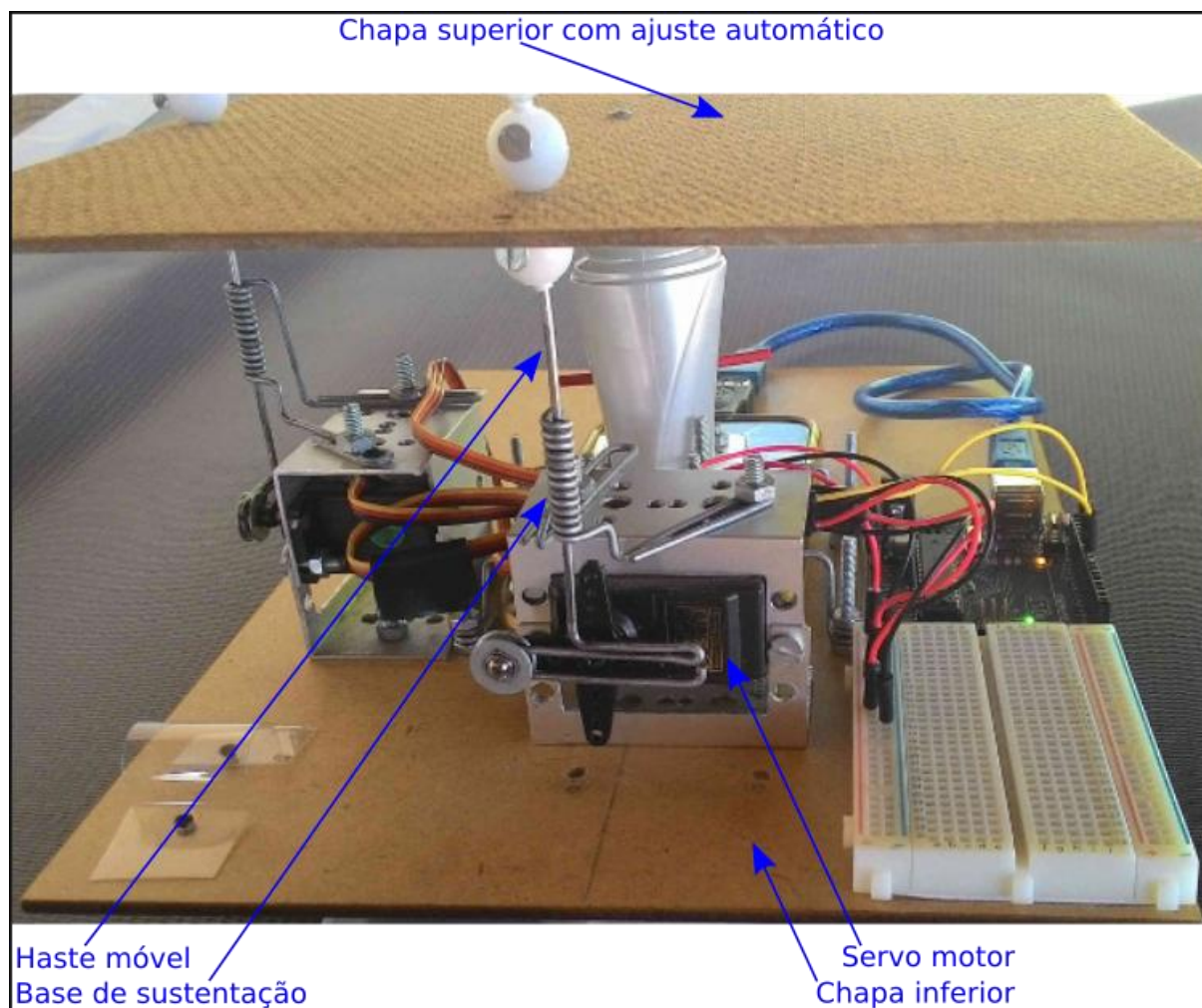
Fonte: elaborado pelo autor (2015)

Porém existe uma desvantagem nesta solução, onde intervalos iguais de ângulos do servo motor localizados em diferentes partes da sua extensão angular representam intervalos de elevação vertical diferentes, veja, por exemplo, que o intervalo de 30° compreendido entre os ângulos 90° e 120° do servo motor representa uma elevação maior que o intervalo de 30° compreendido entre 150° e 180° em **A** e analogamente ocorre em **B** ao diminuir os ângulos.

Entretanto isso também pode representar uma vantagem considerando que em torno de 90°, ponto inicial do movimento da haste, tem-se um maior arranque e velocidade, porém próximo de 0° ou de 180° nota-se suavização na finalização do movimento.

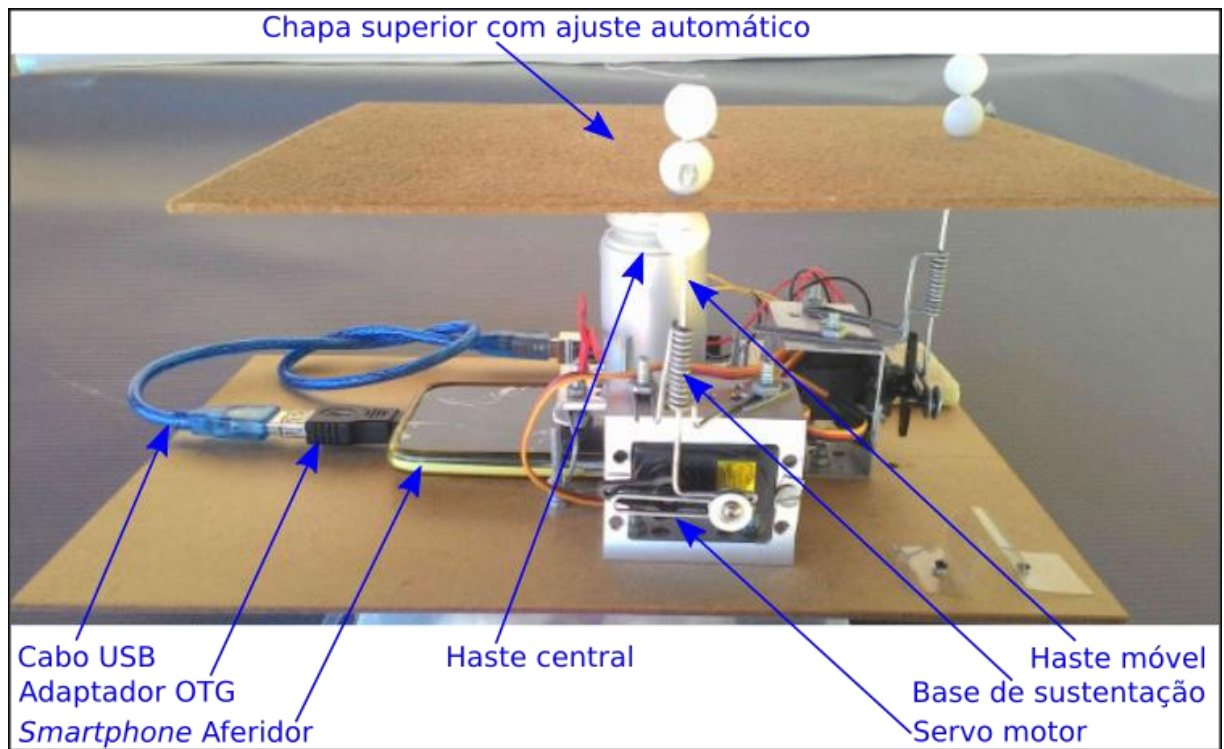
O projeto final com todos os componentes utilizados é apresentado nas **Figuras 4.16, 4.17 e 4.18.**

Figura 4.16 – Vista frontal do projeto final



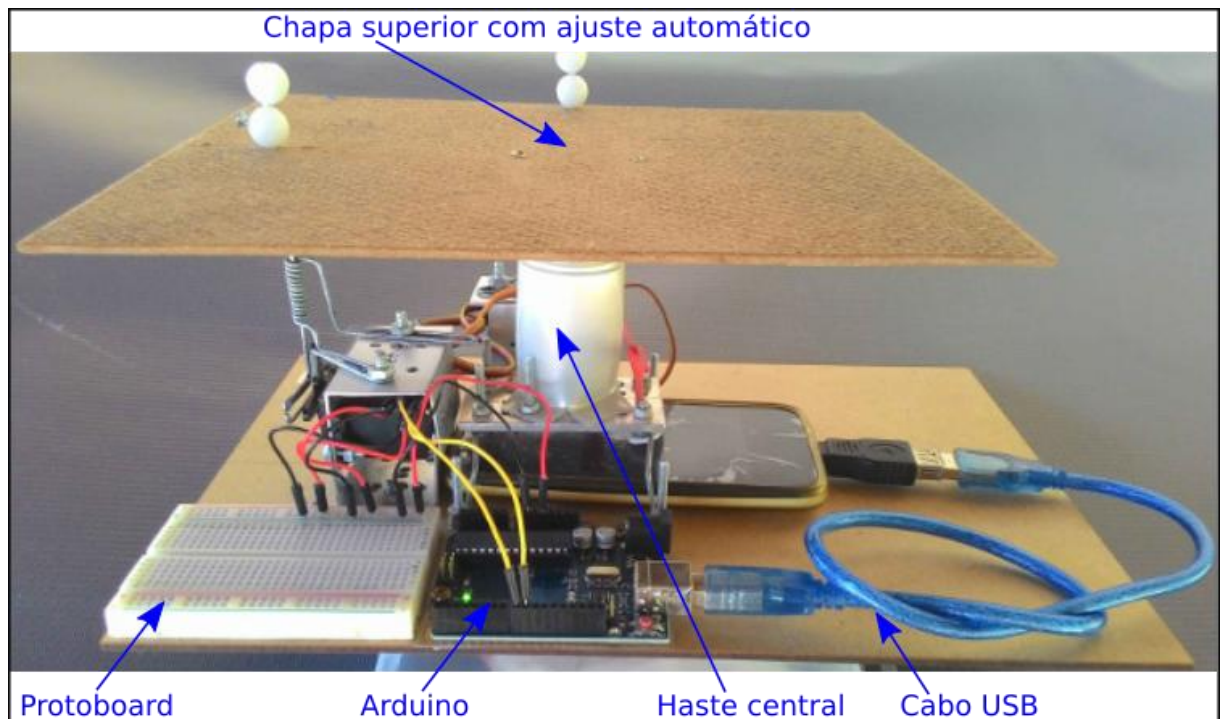
Fonte: elaborado pelo autor (2015)

Figura 4.17 – Vista lateral esquerda



Fonte: elaborado pelo autor (2015)

Figura 4.18 – Vista lateral direita



Fonte: elaborado pelo autor (2015)

4.6 Testes, experimentos e resultados

Para analisar a eficiência do mecanismo, os experimentos foram concentrados principalmente em dois tipos: com um objeto sólido e com um *smartphone*, ambos postos sobre a chapa superior, em momentos diferentes, enquanto o conjunto todo é posto em movimento para simular desníveis e oscilações. Durante os testes foi usada tanto uma fonte CC quanto o próprio *smartphone* para alimentar os atuadores e não se percebeu perda de desempenho.

Como no projeto não foi desenvolvido a parte de locomoção, a simulação foi feita manualmente com a plataforma sobre a mão de um indivíduo, que realiza os movimentos, sendo estes limitados por alguns fatores, tais como: a plataforma não atua para correções de impactos verticais, ou seja, trata somente de manter em equilíbrio a chapa superior e a carga; os desníveis corrigidos são de até 25°; os servos motores tem rotação de aproximadamente 180°/0,6s, tendo em vista que de acordo com o fabricante, com alimentação de 4,8 V consegue-se 60°/0,19s, portanto, o menor tempo de resposta dos atuadores para um desnivelamento máximo de 25° da plataforma é de 0,3 s, considerando que o servo motor precisa girar 90°.

A plataforma mostrou-se eficiente nos experimentos feitos com um cubo de madeira colocado sobre a chapa superior, que visivelmente sofreu um desnivelamento bem menor que o da chapa inferior, sendo realizadas inclinações para várias direções.

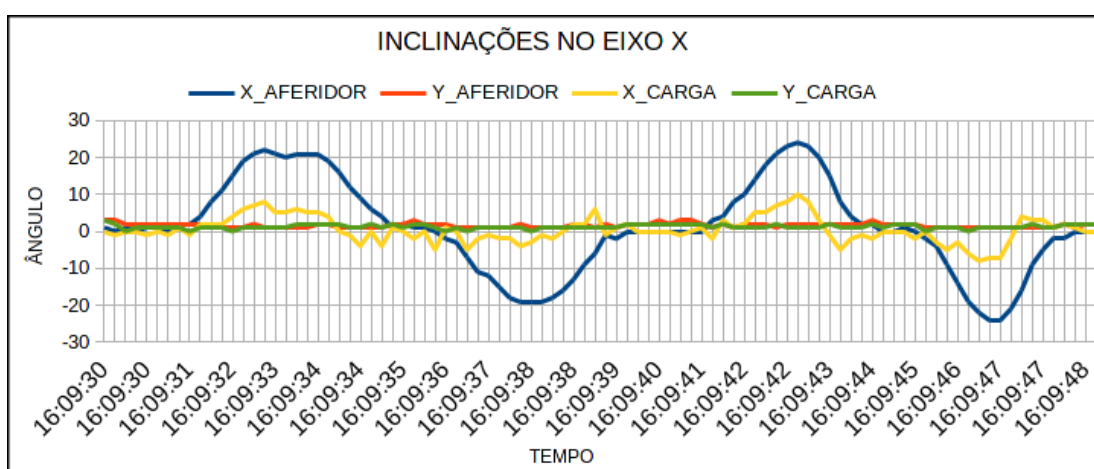
Na realização do segundo experimento usando um *smartphone* como carga e executando o aplicativo desenvolvido no projeto, notou-se que os dados do acelerômetro podem ser diferentes dependendo do seu fabricante ou modelo do dispositivo, no que se refere à variação e valor instantâneo dos dados, isto é, enquanto um deles é 5 o outro pode ser 4 ou 6 m/s², por esta razão, optou-se por usar dois *smartphones* de mesmo modelo e marca.¹

Os experimentos foram feitos de duas maneiras: a primeira usando a funcionalidade de salvamento dos ângulos em arquivos CSV, presente em ambos os aplicativos; a segunda usando o envio de ângulos em tempo real para o dispositivo de controle, no qual se pode analisar o gráfico das inclinações na tela de *status*.

No experimento usando arquivos CSV é necessário que os *smartphones* estejam com horários sincronizados, e embora configurados com ajuste automático pela Internet, notou-se sempre uma diferença de três a cinco segundos entre os mesmos, o que reflete inconsistências nos resultados, considerando a quantidade de dados salva por segundo. Em função disto o horário dos *smartphone* foi configurado manualmente.

O experimento iniciou-se aplicando à plataforma inclinações que afetam diretamente somente um dos eixos e considerando a velocidade limitada pelos motores, como se percebe na **Figura 4.19**.

Figura 4.19 – Inclinações no eixo x



Fonte: elaborado pelo autor (2015)

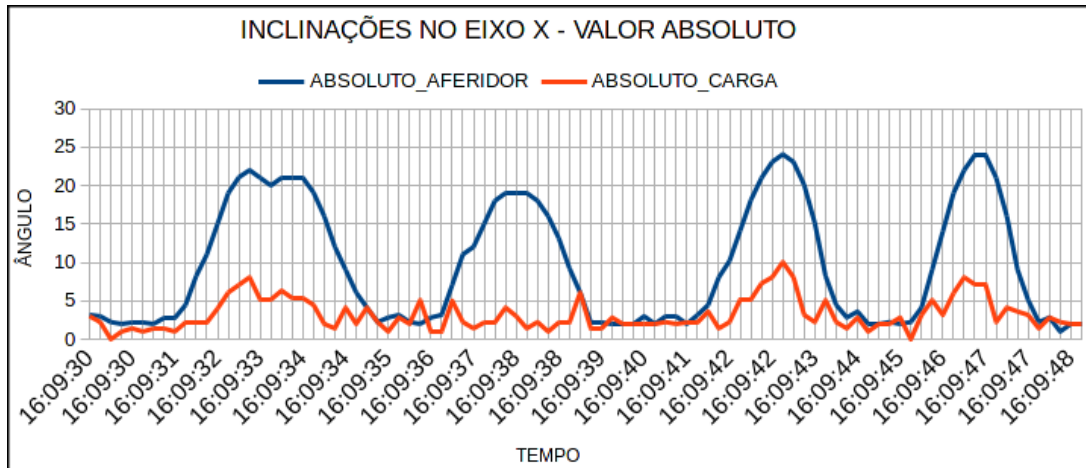
Nota-se no gráfico da **Figura 4.19** que as inclinações realizadas na chapa inferior da plataforma, que afetam diretamente o eixo x e são representadas pela linha azul, estão compreendidas em torno 20° a partir de zero e que o tempo decorrido a cada 20°, crescente ou decrescente, é de aproximadamente 1,5 s, como por exemplo, o intervalo entre 16:09:31 e 16:09:33, bem como, entre 16:09:36 e 16:09:38.

Por outro lado, as inclinações ocorridas no eixo x do *smartphone* carga e expostas na linha amarela são menores que 10°, o que representa uma correção de desnível maior que 50%, porém, ressalta-se que o tempo decorrido ao realizar cada inclinação é bastante superior ao tempo mínimo de resposta previsto dos atuadores e, em consequência, ao diminuir o tempo de execução das inclinações é reduzida também a taxa de correção do desnível.

¹ Os modelos de *smartphones* testados foram: Zenfone 5 da marca Asus e o Moto G (2ª geração) da marca Motorola.

Para expressar os resultados de modo mais concreto e/ou, melhor visualizável, cada par de ângulos, x e y, de ambos os dispositivos, foi convertido para um valor absoluto usando a fórmula: $\sqrt{x^2+y^2}$ e o resultado expõe-se no gráfico da **Figura 4.20**.

Figura 4.20 – Inclinações no eixo x com valores absolutos



Fonte: elaborado pelo autor (2015)

Aplicando-se técnicas de estatística básica, variância e desvio padrão, por meio da expressão presente na **Figura 4.21**, aos dados do gráfico de valores absolutos chega-se ao resultado de uma variância de 61,1 e desvio padrão 7,8 nos ângulos do aferidor enquanto que a carga sofre uma variância de apenas 4,4 e desvio padrão de 2,1. Conclui-se então que houve uma correção média de desnivelamento de 73,1%.

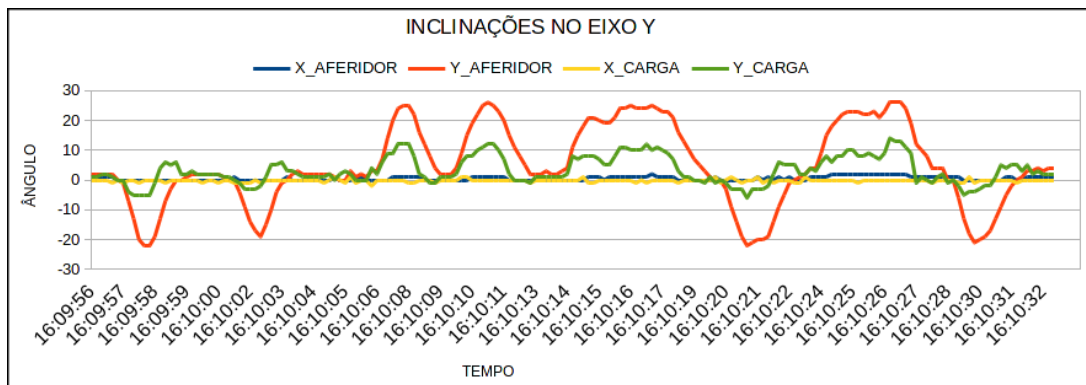
Figura 4.21 – Variância e desvio padrão

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

s^2 símbolo comumente usado para representar a variância X
 x_i representa cada um dos elementos do conjunto no somatório
 \bar{x} representa a média aritmética do conjunto
 n número de elementos do conjunto
 s o desvio padrão é a raiz quadrada da variância

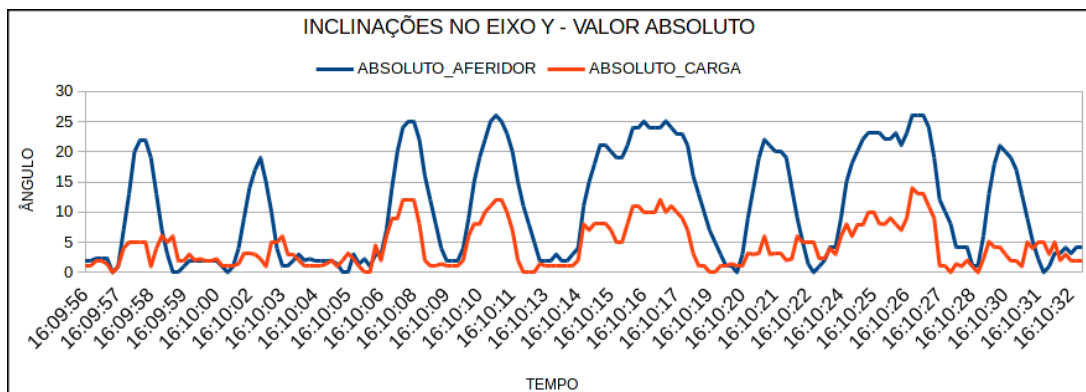
Fonte: elaborado pelo autor (2015)

De forma análoga, os resultados das inclinações aplicadas sobre o eixo y estão apresentados nas **Figuras 4.22 e 4.23**. Em ambas, o resultado é bastante semelhante ao de x, onde a porcentagem de correção se mantém acima de 50% na maior parte dos casos.

Figura 4.22 – Inclinações no eixo y

Fonte: elaborado pelo autor (2015)

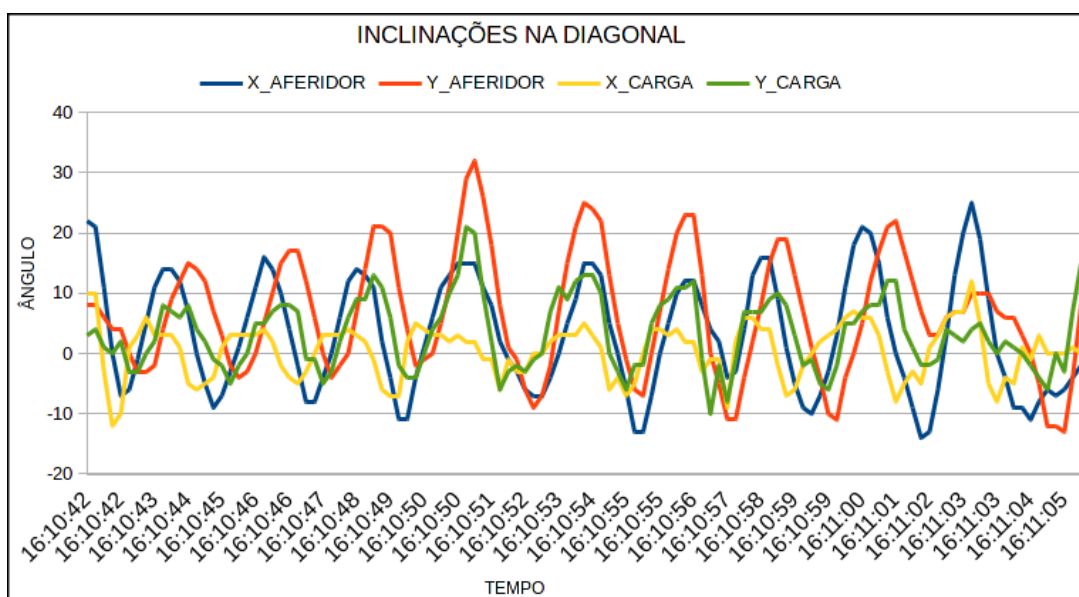
A **Figura 4.23** apresenta os valores absolutos das inclinações aplicadas sobre o eixo y, onde a variância é 80,1 e o desvio padrão é 8,9 nos ângulos do aferidor contra 12,8 de variância e 3,6 de desvio padrão nos ângulos da carga. Conclui-se então que houve uma correção média de desnivelamento de 60%.

Figura 4.23 – Inclinações no eixo y com valores absolutos

Fonte: elaborado pelo autor (2015)

As **Figuras 4.24** e **4.25** apresentam inclinações diagonais que implicam alterações angulares em todos os eixos.

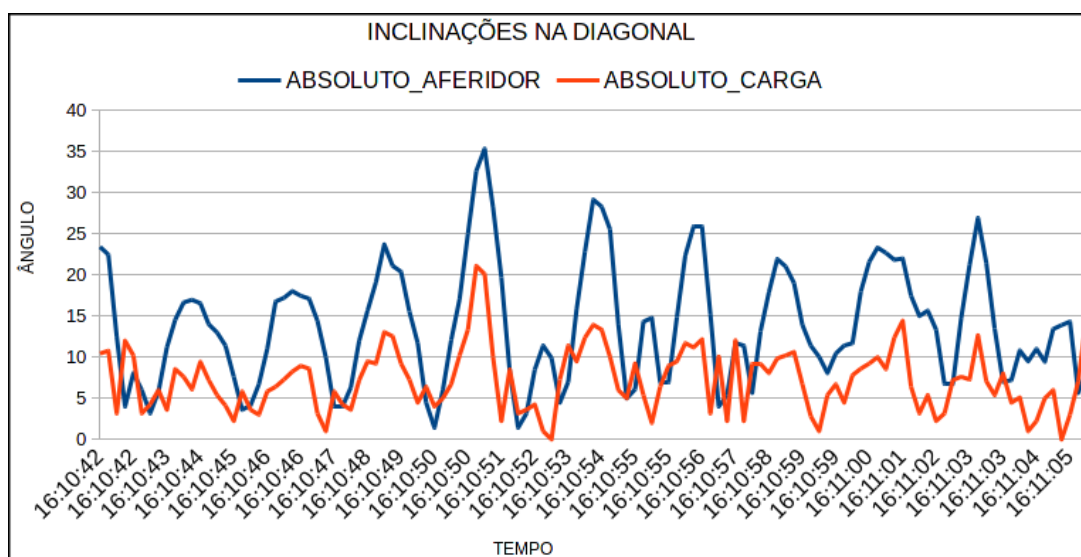
Figura 4.24 – Inclinações na diagonal



Fonte: elaborado pelo autor (2015)

Nota-se na **Figura 4.25** a existência de alguns valores divergentes, entretanto ainda é possível ver uma porcentagem de correção similar às anteriores em grande parte dos casos e uma variância de 52,5 e desvio padrão de 7,2 nos ângulos do aferidor, por outro lado a variância na carga é 15,2 e com desvio padrão 3,9. Entretanto a correção média de desnivelamento neste caso foi de 46,1%.

Figura 4.25 – Inclinações na diagonal com valores absolutos

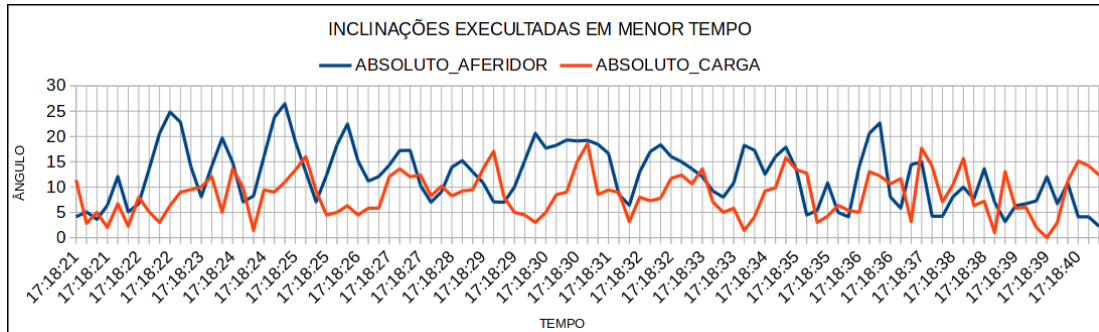


Fonte: elaborado pelo autor (2015)

Analisando outro cenário, no qual as inclinações têm o seu tempo total de execução reduzido, isto é, aumento da velocidade de execução do movimento, observa-se uma variância de 33,3 e um desvio padrão 5,8 nos ângulos do aferidor, já na carga a variância é 18,1 e o

desvio padrão 4,3, bastante semelhante ao desvio padrão do aferidor, representando somente uma correção média de desnivelamento de 26,3%, veja a **Figura 4.26**.

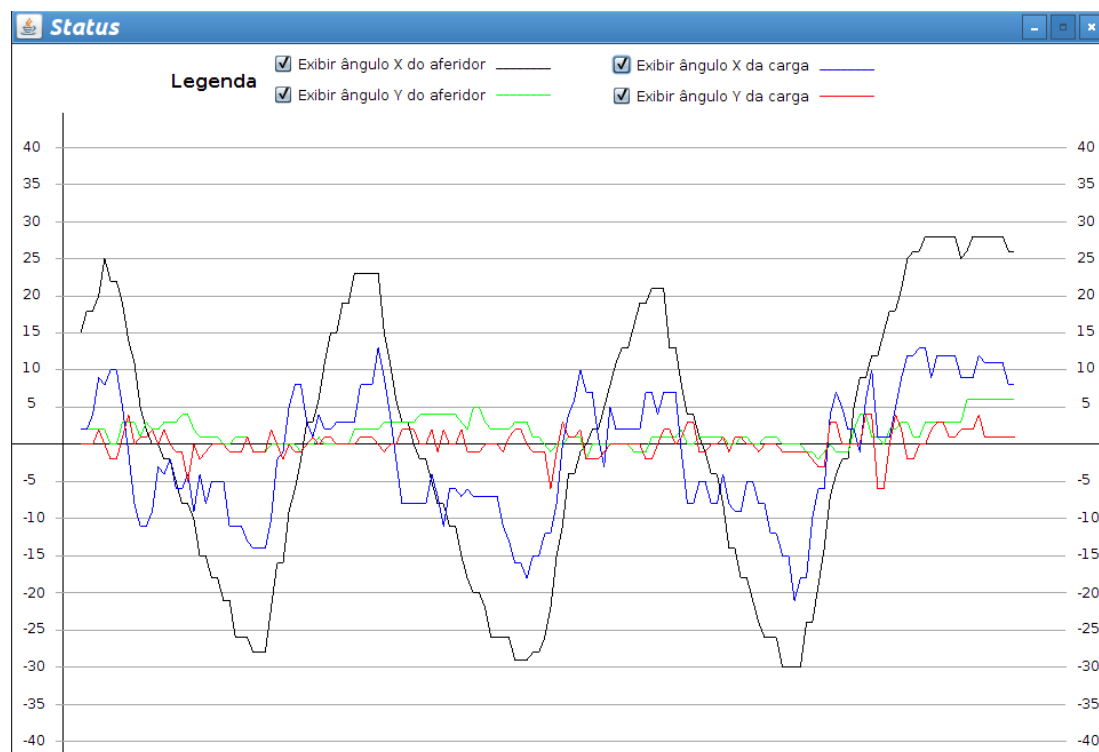
Figura 4.26 – Inclinações executadas em menor tempo



Fonte: elaborado pelo autor (2015)

Os resultados dos experimentos utilizando a funcionalidade de exibição de *status* presente no software de controle também se mostraram semelhantes aos anteriores, nos quais se percebe cerca de 50% a 60% de correção. A **Figura 4.27** mostra inclinações aplicadas no eixo x.

Figura 4.27 – Tela *status*: inclinações no eixo x

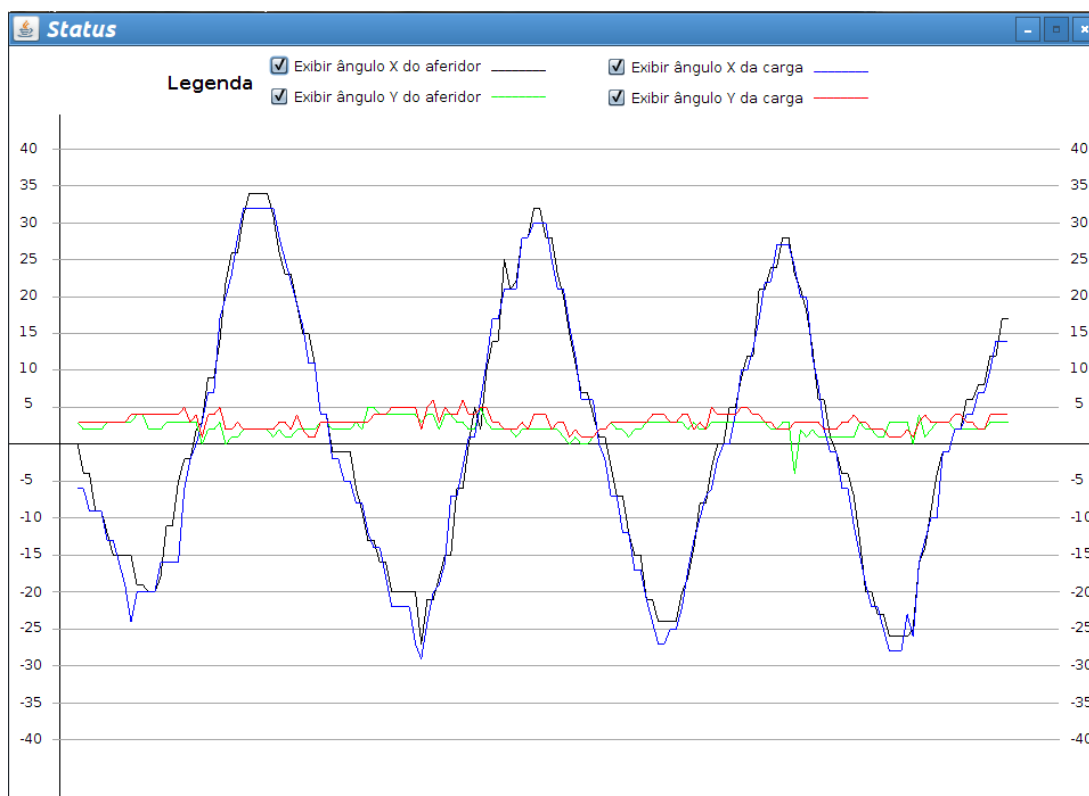


Fonte: elaborado pelo autor (2015)

Para efeito de comparação ainda realizou-se experimentos com os dois *smartphones* sem a plataforma, somente colocando um deles sobre o outro e realizando os movimentos.

Um dos resultados pode ser observado no gráfico da **Figura 4.28**. Os ângulos dos eixos X têm praticamente a mesma variação, não havendo nenhuma redução das inclinações.

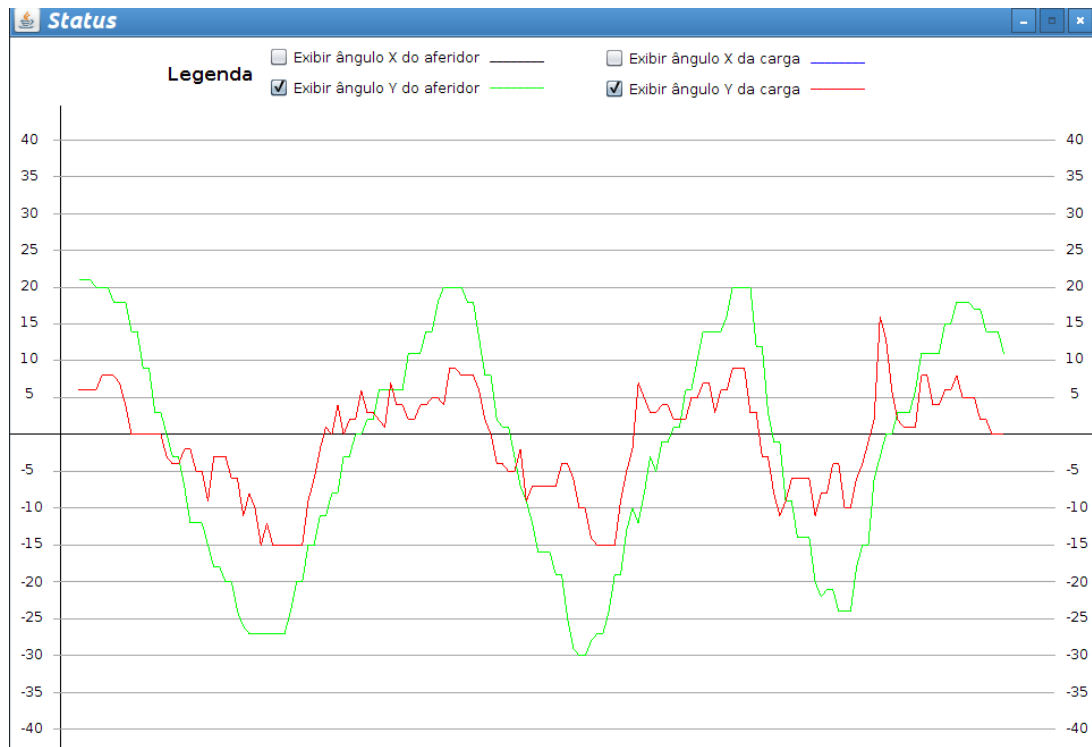
Figura 4.28 – Tela *status*: inclinações no eixo x sem a plataforma



Fonte: elaborado pelo autor (2015)

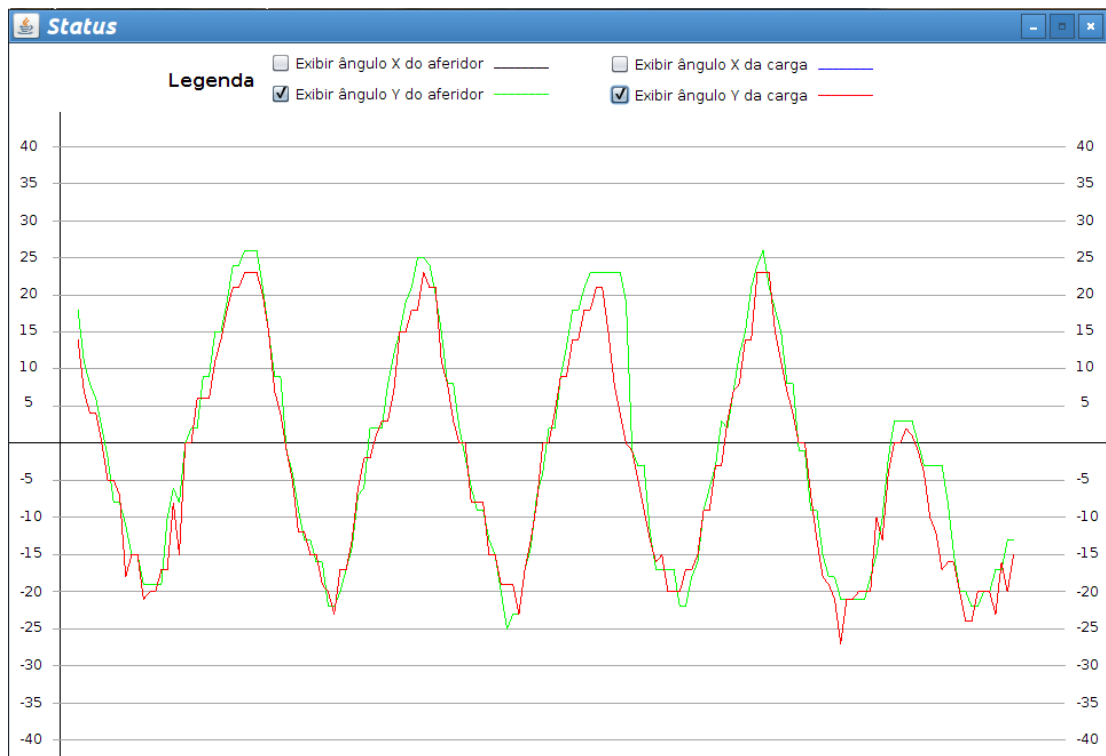
Analogamente ocorre com os eixos y, como demonstrado nas **Figuras 4.29 e 4.30**.

Figura 4.29 – Tela status: inclinações no eixo y



Fonte: elaborado pelo autor (2015)

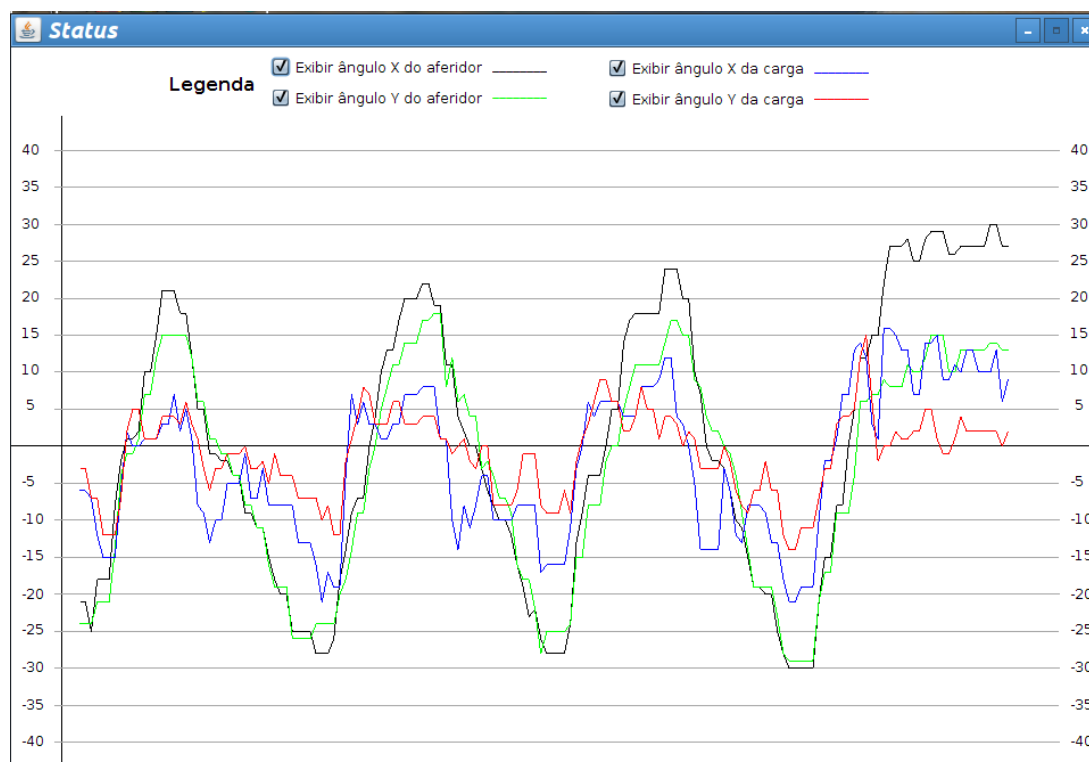
Figura 4.30 – Tela status: inclinações no eixo y sem a plataforma



Fonte: elaborado pelo autor (2015)

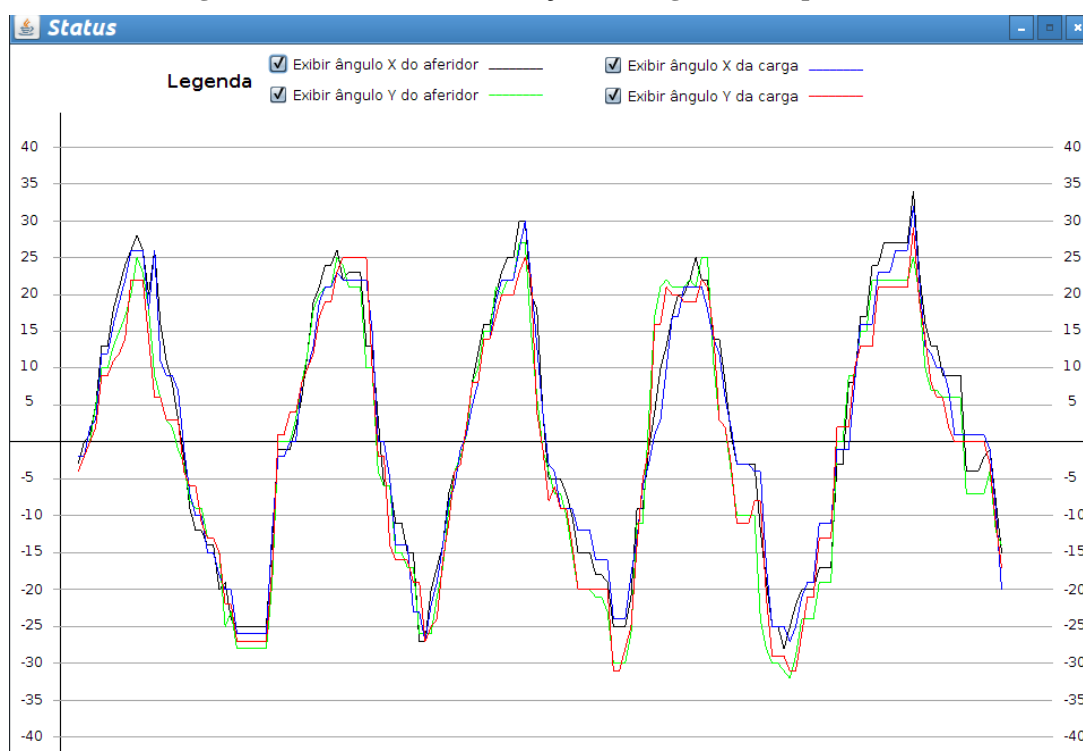
De mesmo modo os experimentos com inclinações diagonais, que afetam os eixos x e y, refletem os resultados anteriores, onde ocorre cerca de 50% a 60% de correção na maioria das inclinações, como observado na **Figura 4.31**. Já a **Figura 4.32** exemplifica a situação sem usar a plataforma.

Figura 4.31 – Tela status: inclinações na diagonal



Fonte: elaborado pelo autor (2015)

Figura 4.32 – Tela *status*: inclinações na diagonal sem a plataforma



Fonte: elaborado pelo autor (2015)

Em relação à comunicação entre o dispositivo de controle e o *smartphone* aferidor por meio da utilização do seu *hotspot wi-fi*, comprovou-se em testes que a transmissão dos dados é eficiente até aproximadamente 35 m ao ar livre, sendo importante ressaltar que o dispositivo de controle usado é um *notebook* que dispõe de uma placa *wi-fi* do padrão 802.11g. Visto que esse é um modelo de placa é um pouco antigo, isso pode ter reduzido a distância máxima para transmissão de dados pelo *smartphone*.

5 Conclusão

5.1 Considerações finais

Os resultados obtidos mostram uma correção média de desnivelamento acima de 50% na maioria das inclinações aplicadas em tempo real à plataforma e consequentemente ameniza os danos sofridos pela carga transportada, entretanto ainda há muitas limitações no

que se refere ao tempo gasto na correção dos desníveis, tanto pela velocidade dos atuadores quanto pela solução de hasteamento implementada.

No projeto foi considerado o transporte de cargas como a principal aplicabilidade, porém a ideia desenvolvida pode destinar-se a qualquer contexto que necessite um instrumento ou material nivelado, como por exemplo, balanças utilizadas por pesquisadores ou alunos para pesagens de materiais em estudo necessitam de um nivelamento perfeito para não comprometer os resultados do experimento, considerando que em muitos casos a matéria prima a ser estudada é colhida e pesada em ambientes externos com vegetação densa e terrenos irregulares, nos quais é muito complicado estabilizar a balança.

Em contextos como o citado o tempo de resposta ao desnivelamento não é essencialmente um problema, considerando a ausência de correções em tempo real, bastando então que haja uma melhora na forma de hasteamento móvel para alcançar uma maior estabilidade e precisão.

5.2 Contribuições deste trabalho

O trabalho aborda a interação entre Android e Arduino de uma forma diferenciada. Mostra que é possível obter nivelamento por meio do acelerômetro presente em *smartphones*. Comprovou-se que transmissão de dados entre Android e Arduino via USB é bastante estável e rápida.

5.3 Proposta para trabalhos futuros

- Desenvolver o sistema de locomoção e testá-lo em ambientes nos quais existam terrenos irregulares;
- Melhorar o sistema de hasteamento móvel vertical para que se obtenha um nivelamento com maior precisão;
- Desenvolver um sistema de estabilidade que leve em consideração a aceleração de deslocamentos verticais e horizontais;

- Melhorar o aplicativo aferidor para que se adapte às variações de dados do acelerômetro de diferentes modelos de *smartphones*;
- Acrescentar configurações para especificar o peso e/ou tipo do material transportado e assim adequar os movimentos da plataforma para casos específicos;
- Implementar aplicativos de controle destinados às diferentes plataformas móveis como Android e iOS ou uma interface Web unificada para diferentes dispositivos;

REFERÊNCIAS BIBLIOGRÁFICAS

ABLESON, F. **Introdução ao Desenvolvimento do Android**, 2009. Disponível em: <<http://www.ibm.com/developerworks/br/library/os-android-devel/index.html>>. Acesso em: 02 out. 2015.

ANDROID. Disponível em: <<http://android.com/>>. Acesso em: 29 set. 2015.

ANDROID DEVELOPER. Disponível em: <<http://developer.android.com>>. Acesso em: 29 set. 2015.

ARDUINO. Disponível em: <<http://arduino.cc/>>. Acesso em: 29 set. 2015.

ARDUINO UNO. **Arduino UNO & Genuino UNO**. Disponível em: <<https://www.arduino.cc/en/Main/ArduinoBoardUno>>. Acesso em: 29 set. 2015.

ASIMOV, Isaac. **Eu, Robot**. USA: Gnome Press, 1950.

AZEVEDO, S.; AGLAÉ, A; PITTA, R. **Minicurso: Introdução a Robótica Educacional**. Disponível em: <<http://www.sbpcnet.org.br/livro/62ra/minicursos/MC%20Samuel%20Azevedo.pdf>>. Acesso em: 22 mai. 2015.

BASCONCELLO FILHO, D. O. **Curso de Arduino Aula 2 - O Hardware do Arduino**. 2012a. Disponível em: <http://www.robotizando.com.br/curso_arduino_hardware_pg1.php>. Acesso em: 29 set. 2015.

BASCONCELLO FILHO, D. O. **O que afinal é Arduino?**, 2012b. Disponível em: <http://www.robotizando.com.br/curso_arduino_o_que_e_arduino_pg1.php>. Acesso em: 29 set. 2015.

BENEDICT. **Student's 3D printed balancing device is a whole new ball game**. Disponível em: <<http://www.3ders.org/articles/20151213-students-3d-printed-balancing-device-is-a-whole-new-ball-game.html>>. Acesso em: 01 dez. 2015.

BIG DOG. Disponível em: <http://www.bostondynamics.com/robot_bigdog.html>. Acesso em: 22 mai. 2015.

BOSTON DYNAMICS. Disponível em: <<http://www.bostondynamics.com>>. Acesso em: 22 mai. 2015.

CERBO, Manuel. **Android USB Host + Arduino: How to communicate without rooting your Android Tablet or Phone**, 2012. Disponível em: <<http://android.serverbox.ch/?p=549>>. Acesso em: 02 out. 2015.

DARPA. *Defense Advanced Research Projects Agency*. **About DARPA**. Disponível em: <<http://www.darpa.mil/about-us/about-darpa>>. Acesso em: 25 mai. 2015.

FERNANDES, A. M. da R.; LAURINDO, R. D. Publicação de artigos científicos. **Sistema de Controle Baseado em Telefonia Celular**. Disponível em: <<http://www.aedb.br/seget/artigos11/13214145.pdf>>. Acesso em: 27 mai. 2015.

GUIMARÃES, G. **A história do sistema operacional Android**, 2013. Disponível em: <http://www.dsc.ufcg.edu.br/~pet/jornal/agosto2013/materias/historia_da_computacao.html>. Acesso em: 02 out. 2015.

GONÇALVES, A. J. R. **Aplicativo em Android para controle de unidades robóticas móveis com Arduino**, 2013. Disponível em: <<http://www.ufpi.br/subsiteFiles/picos/arquivos/files/FINAL.pdf>>. Acesso em: 25 mai. 2015.

LABDEGARAGEM. **O que é arduino?**, 2015. Disponível em: <<http://arduino.labdegaragem.com/>>. Acesso em: 29 set. 2015.

LABDEGARAGEM. **Como utilizar uma protoboard?**, 2011. Disponível em: <http://www.labdegaragem.com.br/wiki/index.php?title=Como_utilizar_uma_protoboard>. Acesso em: 29 set. 2015.

LECHETA, R. R. **Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. São Paulo: Novatec, 2009.

MONTEIRO, G. B. **Google Android crie aplicações para celulares e tablets**. São Paulo: Casa do Código, 2012.

NASCIMENTO, Alexandre. **O que é acelerômetro? Como funciona?**, 2012. Disponível em: <<http://www.appleboy.com.br/blog/o-que-e-acelerometro-como-funciona>>. Acessado em: 01 jun. 2015.

PRACIANO, Elias. **Conheça os sensores do seu smartphone ou tablete**, 2015. Disponível em: <<http://elias.praciano.com/2015/02/conheca-os-sensores-do-seu-smartphone-ou-tablet/>>. Acesso em: 01 jun. 2015.

RUIC, Gabriela. **A história dos Robôs em imagens**, 2012. Disponível em: <<http://exame.abril.com.br/tecnologia/album-de-fotos/a-historia-dos-robos-em-imagens#1>>. Acessado em: 01 jun. 2015.

SANTOS, V. M. F. **Robótica Industrial**, 2003-2004. Disponível em: <www.ece.ufrgs.br/~rventura/RoboticaIndustrial.pdf>. Acesso em: 02 jul. 2015.

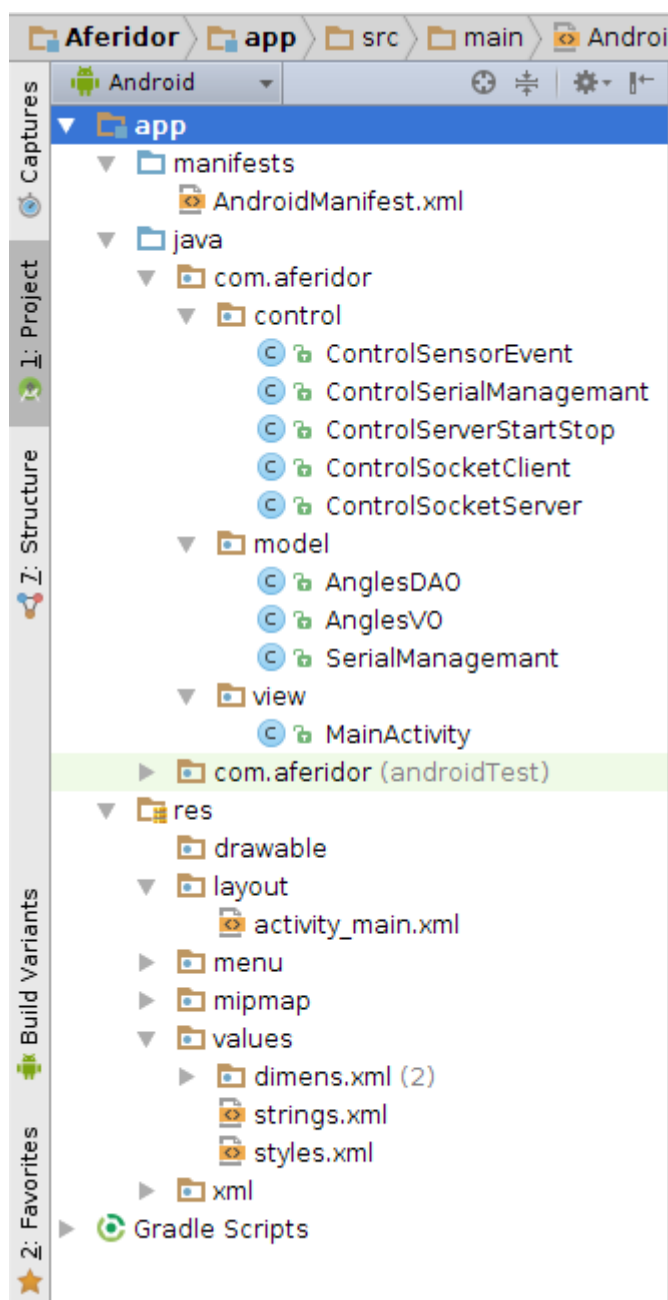
SOARES, Karla. **O que é um Arduino e o que pode ser feito com ele?**, 2013. Disponível em: <<http://www.techtudo.com.br/noticias/noticia/2013/10/o-que-e-um-arduino-e-o-que-pode-ser-feito-com-ele.html>>. Acesso em: 29 set. 2015.

SOUZA, Fahel. **O que é Protoboard e Stripboard?**, 2013. Disponível em: <<http://segredoeletronico.blogspot.com.br/2013/10/o-que-e-protoboard-e-stripboard.html>>. Acesso em: 29 set. 2015.

APÊNDICE A – Código fonte do aplicativo Aferidor

Este aplicativo foi desenvolvido utilizando o Android Studio e segue sua estrutura padrão de organização dos arquivos do código fonte, como mostrado na **Figura 5.1**.

Figura 5.1 – Organização de arquivos do código fonte do Aferidor



Fonte: elaborado pelo autor (2015)

Os arquivos criados e/ou editados estão listados a seguir.

Classe *MainActivity.java*

```

1  package com.aferidor.view;
2  import android.app.Activity;
3  import android.hardware.Sensor;
4  import android.hardware.SensorManager;
5  import android.os.Bundle;
6  import android.view.Menu;
7  import android.view.MenuItem;
8  import android.widget.Button;
9  import android.widget.TextView;
10 import com.aferidor.control.ControlSensorEvent;
11 import com.aferidor.control.ControlSerialManagemant;
12 import com.aferidor.R;
13 import com.aferidor.control.ControlServerStartStop;
14 import java.net.InetAddress;
15 import java.net.NetworkInterface;
16 import java.net.SocketException;
17 import java.util Enumeration;
18 public class MainActivity extends Activity {
19     private ControlSerialManagemant controlSerialManagemant;
20     private ControlSensorEvent controlSensorEvent;
21     private SensorManager sensorManager;
22     private TextView textView_serverStatus, textView_serverIP,
23     textView_angleXValue, textView_angleYValue;
24     private Button button_serverStartStop;
25     private int direction;
26     private boolean dataSave, sendStatus;
27     private String ipSendStatus;
28     @Override
29     protected void onCreate(Bundle savedInstanceState) {
30         super.onCreate(savedInstanceState);
31         setContentView(R.layout.activity_main);
32         this.direction = 11;
33         this.dataSave = false;
34         this.sendStatus = false;
35         this.ipSendStatus = null;
36         this.sensorManager = (SensorManager)

```

```

37 getSystemService(SENSOR_SERVICE);
38     this.controlSerialManagemant = new
39 ControlSerialManagemant(this);
40     this.controlSensorEvent = new ControlSensorEvent(this);
41     this.textView_serverStatus = (TextView)
42 findViewById(R.id.textView_serverStatus);
43     this.textView_serverIP = (TextView)
44 findViewById(R.id.textView_severIP);
45     this.textView_serverIP.setText("IP: " + getIp());
46     this.textView_angleXValue = (TextView)
47 findViewById(R.id.textView_angleXValue);
48     this.textView_angleYValue = (TextView)
49 findViewById(R.id.textView_angleYValue);
50     this.button_serverStartStop = (Button)
51 findViewById(R.id.button_serverStartStop);
52     this.button_serverStartStop.setOnClickListener(new
53 ControlServerStartStop(this));
54 }
55 @Override
56 protected void onResume() {
57     super.onResume();
58     this.controlSerialManagemant.openUSBConnection();
59     this.controlSerialManagemant.startUSBConnection();
60 }
61 @Override
62 protected void onPause() {
63     super.onPause();
64     this.controlSerialManagemant.closeUSBConnection();
65 }
66 @Override
67 public boolean onCreateOptionsMenu(Menu menu) {
68     // Inflate the menu; this adds items to the action bar if it
69 is present.
70     getMenuInflater().inflate(R.menu.menu_main, menu);
71     return true;
72 }
73 @Override
74 public boolean onOptionsItemSelected(MenuItem item) {
75     // Handle action bar item clicks here. The action bar will
76 // automatically handle clicks on the Home/Up button, so
77 long

```

```

78      // as you specify a parent activity in AndroidManifest.xml.
79      int id = item.getItemId();
80      //noinspection SimplifiableIfStatement
81      if (id == R.id.action_settings) {
82          return true;
83      }
84      return super.onOptionsItemSelected(item);
85  }
86  public TextView getTextView_serverStatus() {
87      return this.textView_serverStatus;
88  }
89  public TextView getTextView_angleXValue() {
90      return textView_angleXValue;
91  }
92  public int getDirection() {
93      return direction;
94  }
95  public void setDirection(int direction) {
96      this.direction = direction;
97  }
98  public void setDataSave(boolean dataSave) {
99      this.dataSave = dataSave;
100  }
101  public boolean isDataSave() {
102      return dataSave;
103  }
104  public boolean isSendStatus() {
105      return sendStatus;
106  }
107  public void setSendStatus(boolean sendStatus) {
108      this.sendStatus = sendStatus;
109  }
110  public TextView getTextView_angleYValue() {
111      return textView_angleYValue;
112  }
113  public Button getButton_serverStartStop() {
114      return button_serverStartStop;
115  }
116  public String getIpSendStatus() {
117      return ipSendStatus;
118  }

```



```

119     public void setIpSendStatus(String ipSendStatus) {
120         this.ipSendStatus = ipSendStatus;
121     }
122     public void registerSensorManager() {
123         sensorManager.registerListener(controlSensorEvent,
124         sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
125         SensorManager.SENSOR_DELAY_FASTEST);
126     }
127     public void unregisterSensorManager() {
128         sensorManager.unregisterListener(controlSensorEvent);
129     }
130     public String getIp() {
131         String ipAddress = null;
132         Enumeration<NetworkInterface> net = null;
133         try {
134             net = NetworkInterface.getNetworkInterfaces();
135         } catch (SocketException e) {
136             throw new RuntimeException(e);
137         }
138         while (net.hasMoreElements()) {
139             NetworkInterface element = net.nextElement();
140             Enumeration<InetAddress> addresses =
141             element.getInetAddresses();
142             while (addresses.hasMoreElements()) {
143                 InetAddress ip = addresses.nextElement();
144                 if (ip.isSiteLocalAddress()) {
145                     ipAddress = ip.getHostAddress();
146                 }
147             }
148         }
149         return ipAddress;
150     }
151 }
152

```

Classe ControlSensorEvent.java

```

1 package com.aferidor.control;
2 import android.hardware.Sensor;
3 import android.hardware.SensorEvent;
4 import android.hardware.SensorEventListener;

```

```

5  import android.os.AsyncTask;
6  import android.widget.Toast;
7  import com.aferidor.model.AnglesDAO;
8  import com.aferidor.model.AnglesVO;
9  import com.aferidor.model.SerialManagemant;
10 import com.aferidor.view.MainActivity;
11 import java.text.SimpleDateFormat;
12 import java.util.ArrayList;
13 import java.util.Calendar;
14 /**
15  * Created by leonardo on 15/05/15.
16  */
17 public class ControlSensorEvent implements SensorEventListener {
18     private MainActivity main_Activity;
19     private float interval, x_old, y_old, minimumOscillation;
20     private float[] values;
21     private String fileName;
22     private SimpleDateFormat dateFormat = new
23 SimpleDateFormat("yyyy-MM-dd__HH-mm-ss");
24     private int maxValues, cont, direction_old;
25     public ControlSensorEvent(MainActivity mainActivity) {
26         this.main_Activity = mainActivity;
27         this.fileName = "Dados__" +
28 this.dateFormat.format(Calendar.getInstance().getTime()); //nome do
29 arquivo salvo
30         this.maxValues = 5;
31         this.interval = 2.65f;
32         this.cont = 0;
33         this.minimumOscillation = 0.25f;
34         this.x_old = this.y_old = 0f;
35         this.direction_old = this.main_Activity.getDirection();
36
37 SerialManagemant.getUniqueSerialManagemant(this.main_Activity);
38     }
39     @Override
40     public void onSensorChanged(SensorEvent event) {
41         this.cont++;
42         this.values = event.values;
43         if (this.cont == this.maxValues) { // Entra no if pra enviar
44 ao Arduino, salvar em arquivo local e enviar status
45             try {

```

```

46         if (Math.abs(this.values[0] - this.x_old) >
47 this.minimumOscillation ||
48             Math.abs(this.values[1] - this.y_old) >
49 this.minimumOscillation) {
50             int current_angle_x =
51 mapValueToAngle(this.values[0], -this.interval, this.interval, 0,
52 180);
53
54 SerialManagemant.getUniqueSerialManagemant(main_Activity).sendToArdu
55 ino(current_angle_x + "a");
56             int current_angle_y =
57 mapValueToAngle(this.values[1], -this.interval, this.interval, 0,
58 180);
59
60 SerialManagemant.getUniqueSerialManagemant(main_Activity).sendToArdu
61 ino(current_angle_y + "b");
62
63 this.main_Activity.getTextView_angleYValue().setText("ÂnguloY: " +
64 current_angle_y);
65
66 this.main_Activity.getTextView_angleXValue().setText("ÂnguloX: " +
67 current_angle_x);
68             this.x_old = this.values[0];
69             this.y_old = this.values[1];
70         } else {
71             this.values[0] = this.x_old;
72             this.values[1] = this.y_old;
73         }
74         float x = mapValueToAngle(this.values[0], -10f, 10f,
75 -90, 90), y = mapValueToAngle(this.values[1], -10f, 10f, -90, 90);
76         if (this.main_Activity.isDataSave()) {
77             try {
78                 Object[] params = {
79                     "write", //tipo de operação
80                     new AnglesVO(x, y), //dados que serão
81 salvos
82                     this.fileName //nome do arquivo salvo
83                 };
84                 AnglesDAO anglesDAO = new AnglesDAO();
85
86 anglesDAO.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,

```

```

87 params);// usando o executeOnExecutor para rodar multiplas
88 AsyncTasks
89         } catch (Exception e) {
90             Toast.makeText(this.main_Activity, "Erro ao
91 salvar dados no arquivo CSV\n" + e.getMessage(),
92 Toast.LENGTH_LONG).show();
93
94 this.main_Activity.unregisterSensorManager();
95             e.printStackTrace();
96         }
97     }
98     if (this.main_Activity.isSendStatus()) {
99         try {
100             ArrayList<Float> arrayListAngles = new
101 ArrayList<>();
102             arrayListAngles.add(x);
103             arrayListAngles.add(y);
104             Object[] params = {
105                 this.main_Activity.getIpSendStatus(),
106 //IP do servidor para envio de status
107                 8888, //porta usada
108                 arrayListAngles //dados que serão
109 enviados
110             };
111             ControlSocketClient controlSocketClient = new
112 ControlSocketClient();
113
114 controlSocketClient.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR
115 , params);// usando o executeOnExecutor para rodar multiplas
116 AsyncTask
117         } catch (Exception e) {
118             Toast.makeText(this.main_Activity, "Erro ao
119 enviar status\n" + e.getMessage(), Toast.LENGTH_LONG).show();
120
121 this.main_Activity.unregisterSensorManager();
122             e.printStackTrace();
123         }
124     }
125     this.cont = 0;
126 } catch (Exception e) {
127     e.printStackTrace();

```

```

128         Toast.makeText(this.main_Activity, "Erro ao enviar
129 dado para Arduino\n" + e.getMessage(), Toast.LENGTH_LONG).show();
130         this.main_Activity.unregisterSensorManager();
131     }
132 }
133 if (this.direction_old != this.main_Activity.getDirection())
134 {
135     try {
136
137 SerialManagemant.getUniqueSerialManagemant(main_Activity).sendToArdu
138 ino(this.main_Activity.getDirection() + "c");
139         this.direction_old =
140 this.main_Activity.getDirection();
141     } catch (Exception e) {
142         e.printStackTrace();
143         Toast.makeText(this.main_Activity, "Erro ao enviar
144 dado para Arduino\n" + e.getMessage(), Toast.LENGTH_LONG).show();
145         this.main_Activity.unregisterSensorManager();
146     }
147 }
148 }
149 @Override
150 public void onAccuracyChanged(Sensor sensor, int accuracy) {
151 }
152 private int mapValueToAngle(float x, float input_min, float
153 input_max, int output_min, int output_max) {
154     int angle = (int) ((x - input_min) * (output_max -
155 output_min) / (input_max - input_min) + output_min);
156     return angle;
157 }
158 }
159

```

Classe ControlSerialManagemant.java

```

1 package com.aferidor.control;
2 import android.app.Activity;
3 import android.widget.Toast;
4 import com.aferidor.model.SerialManagemant;
5 /**
6  * Created by leonardo on 03/09/15.

```

```

7      */
8      public class ControlSerialManagemant {
9          private Activity activity;
10         public ControlSerialManagemant(Activity activity) {
11             this.activity = activity;
12         }
13         SerialManagemant.getUniqueSerialManagemant(this.activity.getApplicat
14         ionContext());
15     }
16     public void openUSBConnection () {
17         try {
18
19         SerialManagemant.getUniqueSerialManagemant(this.activity.getApplicat
20         ionContext()).openUSBConnection();
21         } catch (Exception e) {
22             Toast.makeText(this.activity.getApplicationContext(),
23             "Erro ao abrir conexão USB", Toast.LENGTH_LONG);
24             e.printStackTrace();
25         }
26     }
27     public void startUSBConnection () {
28         try {
29
30         SerialManagemant.getUniqueSerialManagemant(this.activity.getApplicat
31         ionContext()).startUSBConnection();
32         } catch (Exception e) {
33             Toast.makeText(this.activity.getApplicationContext(),
34             "Erro ao iniciar conexão USB", Toast.LENGTH_LONG);
35             e.printStackTrace();
36         }
37     }
38     public void closeUSBConnection () {
39         try {
40
41         SerialManagemant.getUniqueSerialManagemant(this.activity.getApplicat
42         ionContext()).closeUSBConnection();
43         } catch (Exception e) {
44             Toast.makeText(this.activity.getApplicationContext(),
45             "Erro ao fechar conexão USB", Toast.LENGTH_LONG);
46             e.printStackTrace();
47         }

```

48	}
49	}
50	

Classe *ControlServerStartStop.java*

```

1  package com.aferidor.control;
2  import android.view.View;
3  import android.widget.Toast;
4  import com.aferidor.view.MainActivity;
5  /**
6   * Created by leonardo on 02/09/15.
7   */
8  public class ControlServerStartStop implements View.OnClickListener
9  {
10     private MainActivity mainActivity;
11     public ControlServerStartStop(MainActivity mainActivity) {
12         this.mainActivity = mainActivity;
13     }
14     @Override
15     public void onClick(View v) {
16         if
17 (this.mainActivity.getTextView_serverStatus().getText().toString().e
18 quals("Servidor Parado")) {
19             try {
20
21 ControlSocketServer.getUNIQ_CONTROL_SERVER(this.mainActivity).startS
22 erver();
23 //             Toast.makeText(v.getContext(), "Iniciando servidor",
24 Toast.LENGTH_LONG).show();
25             } catch (Exception e) {
26                 Toast.makeText(v.getContext(), "Erro ao iniciar
27 servidor\n" + e.getMessage(), Toast.LENGTH_LONG).show();
28                 e.printStackTrace();
29             }
30         } else {
31             try {
32
33 ControlSocketServer.getUNIQ_CONTROL_SERVER(this.mainActivity).stopSe
34 rver();
35 //             Toast.makeText(v.getContext(), "Iniciando servidor",

```

```

36 Toast.LENGTH_LONG).show();
37         } catch (Exception e) {
38             Toast.makeText(v.getContext(), "Erro ao parar
39 servidor\n" + e.getMessage(), Toast.LENGTH_LONG).show();
40             e.printStackTrace();
41         } catch (Throwable throwable) {
42             Toast.makeText(v.getContext(), "Erro ao parar
43 servidor\n" + throwable.getMessage(), Toast.LENGTH_LONG).show();
44             throwable.printStackTrace();
45         }
46     }
47 }
48 }
49

```

Classe *ControlSocketClient.java*

```

1  package com.aferidor.control;
2  import android.os.AsyncTask;
3  import java.io.IOException;
4  import java.io.ObjectInputStream;
5  import java.io.ObjectOutputStream;
6  import java.net.Socket;
7  /**
8   * Created by leonardo on 05/06/15.
9   */
10 public class ControlSocketClient extends AsyncTask {
11     @Override
12     protected String doInBackground(Object[] params) {
13         Socket socket = null;
14         ObjectOutputStream objectOutputStream = null;
15         ObjectInputStream objectInputStream = null;
16         String resposta = null;
17         try {
18             socket = new Socket((String) params[0], (int)
19 params[1]);
20             objectOutputStream = new
21 ObjectOutputStream(socket.getOutputStream());
22             objectInputStream = new
23 ObjectInputStream(socket.getInputStream());
24             objectOutputStream.writeObject(params[2]);

```



```

25         } catch (Exception e) {
26             e.printStackTrace();
27         } finally {
28             try {
29                 if (socket != null)
30                     socket.close();
31                 if (objectInputStream != null)
32                     objectInputStream.close();
33                 if (objectOutputStream != null) {
34                     objectOutputStream.flush();
35                     objectOutputStream.close();
36                 }
37             } catch (IOException e) {
38                 e.printStackTrace();
39             }
40         }
41         return resposta;
42     }
43 }
44

```

Classe *ControlSocketServer.java*

```

1  package com.aferidor.control;
2  import android.graphics.Color;
3  import android.os.AsyncTask;
4  import com.aferidor.view.MainActivity;
5  import java.io.ObjectInputStream;
6  import java.io.ObjectOutputStream;
7  import java.net.ServerSocket;
8  import java.net.Socket;
9  import java.util.ArrayList;
10 /**
11  * Created by leonardo on 05/06/15.
12  */
13 public class ControlSocketServer extends AsyncTask {
14     private MainActivity mainActivity;
15     private boolean running;
16     protected static ControlSocketServer UNIQ_CONTROL_SERVER = null;
17     public ControlSocketServer(MainActivity mainActivity) {
18         this.mainActivity = mainActivity;

```

```

19         this.running = true;
20     }
21     public static ControlSocketServer
22     getUNIQ_CONTROL_SERVER(MainActivity mainActivity) {
23         if (UNIQ_CONTROL_SERVER == null)
24             UNIQ_CONTROL_SERVER = new
25             ControlSocketServer(mainActivity);
26         return UNIQ_CONTROL_SERVER;
27     }
28     @Override
29     protected String doInBackground(Object[] params) {
30         String resposta = null;
31         try {
32             ServerSocket serverSocket;
33             Socket socket;
34             ObjectOutputStream objectOutputStream = null;
35             ObjectInputStream objectInputStream = null;
36             ArrayList<Object> option;
37             while (this.running) {
38                 serverSocket = new ServerSocket((Integer)
39 params[0]);
40                 socket = serverSocket.accept();
41                 if (this.running) {
42                     objectOutputStream = new
43 ObjectOutputStream(socket.getOutputStream());
44                     objectInputStream = new
45 ObjectInputStream(socket.getInputStream());
46                     option = (ArrayList<Object>)
47 objectInputStream.readObject();
48                     switch ( (int)option.get(0) ) {
49                         case 1: //registrar sensores
50
51 this.mainActivity.registerSensorManager();
52                         break;
53                         case 2: //desregistrar sensores
54
55 this.mainActivity.unregisterSensorManager();
56                         break;
57                         case 3: //salvar dados em arquivo CSV no
58 celular local
59
60 this.mainActivity.setDataSave(true);

```

```

60         break;
61         case 4: //desativar o salvamento de arquivo
62 CSV
63             this.mainActivity.setDataSave(false);
64             break;
65         case 5: //enviar angulos em tempo real para
66 o dispositivo de controle
67
68 this.mainActivity.setIpSendStatus((String) option.get(1));
69             this.mainActivity.setSendStatus(true);
70             break;
71         case 6: //desativar o envio de angulos
72             this.mainActivity.setSendStatus(false);
73             break;
74         default:
75             this.mainActivity.setDirection(
76 (int)option.get(0) );
77             break;
78     }
79     objectOutputStream.writeObject(option);
80 }
81 objectInputStream.close();
82 objectOutputStream.flush();
83 objectOutputStream.close();
84 socket.close();
85 serverSocket.close();
86 }
87 } catch (Exception e) {
88     e.printStackTrace();
89 }
90 return resposta;
91 }
92 protected void startServer() throws Exception {
93     Object[] params = new Object[1];
94     params[0] = 6789;
95     this.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,
96 params); // usando o executeOnExecutor para rodar multiplas AsyncTask
97
98 this.mainActivity.getTextView_serverStatus().setText("Servidor
99 Iniciado");
100     this.mainActivity.getButton_serverStartStop().setText("Parar

```

```

101  Servidor");
102
103  this.mainActivity.getButton_serverStartStop().setBackgroundColor(Color
104  or.parseColor("#83BE2136"));
105  }
106  protected void stopServer() throws Throwable {
107      this.running = false;
108      this.finalize();
109
110  this.mainActivity.getTextView_serverStatus().setText("Servidor
111  Parado");
112
113  this.mainActivity.getButton_serverStartStop().setText("Iniciar
114  Servidor");
115
116  this.mainActivity.getButton_serverStartStop().setBackgroundColor(Color
117  or.parseColor("#833544ed"));
118      System.exit(0);
119  }
120  }
121

```

Classe AnglesDAO.java

```

1  package com.aferidor.model;
2  import android.os.AsyncTask;
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileWriter;
7  import java.io.PrintStream;
8  import java.text.SimpleDateFormat;
9  import java.util.Calendar;
10  /**
11   * Created by leonardo on 12/11/15.
12   */
13  public class AnglesDAO extends AsyncTask{
14      private SimpleDateFormat dateFormat = new
15  SimpleDateFormat("HH:mm:ss");
16      public void saveCoordinate (AnglesVO anglesVO, String nameFile)
17  throws Exception{

```

```

18         String lineWrite;
19         BufferedWriter bufferedWriter = new BufferedWriter(new
20 FileWriter("/storage/emulated/0/" + nameFile + ".csv", true));
21         lineWrite =
22 dateFormat.format(Calendar.getInstance().getTime()) + ";" +
23 anglesVO.getX() + ";" + anglesVO.getY();
24         lineWrite = lineWrite.replace('.', ',');
25         bufferedWriter.write(lineWrite);
26         bufferedWriter.newLine();
27         bufferedWriter.flush();
28         bufferedWriter.close();
29     }
30     @Override
31     protected Object doInBackground(Object[] params) {
32         switch ( (String) params[0]) {
33             case "write":
34                 try {
35                     this.saveCoordinate( (AnglesVO) params[1],
36 (String) params[2] );
37                 } catch (Exception e) {
38                     e.printStackTrace();
39                 }
40                 try {
41                     e.printStackTrace(new PrintStream(new
42 File("/storage/emulated/0/erros_AnglesDAO.txt")));
43                 } catch (FileNotFoundException e1) {
44                     e1.printStackTrace();
45                 }
46                 break;
47             case "read":
48                 break;
49             case "alter":
50                 break;
51         }
52         return null;
53     }
54 }
55

```

```

1  package com.aferidor.model;
2  import java.io.Serializable;
3  /**
4   * Created by leonardo on 12/06/15.
5   */
6  public class AnglesVO implements Serializable {
7      private static final long serialVersionUID = 1L;
8      private float x, y;
9      public AnglesVO(float x, float y) {
10         this.x = x;
11         this.y = y;
12     }
13     public float getX() {
14         return x;
15     }
16     public float getY() {
17         return y;
18     }
19 }
20

```

Classe SerialManagemant.java

```

1  package com.aferidor.model;
2  import android.content.Context;
3  import android.hardware.usb.UsbManager;
4  import com.hoho.android.usbserial.driver.UsbSerialDriver;
5  import com.hoho.android.usbserial.driver.UsbSerialProber;
6  /**
7   * Created by leonardo on 02/09/15.
8   */
9  public class SerialManagemant {
10     private UsbManager usbManager;
11     private UsbSerialDriver usbSerialDriver;
12     private Context context;
13     private static SerialManagemant UNIQUE_SERIAL_MANAGEMENT = null;
14     private SerialManagemant (Context context) {
15         this.context = context;
16         this.usbManager = (UsbManager)
17 this.context.getSystemService(Context.USB_SERVICE);
18     }

```

```

19     public static SerialManagemant getUniqueSerialManagemant
20     (Context context){
21         if (UNIQUESERIAL_MANAGEMANT == null)
22             UNIQUESERIAL_MANAGEMANT = new SerialManagemant(context);
23         return UNIQUESERIAL_MANAGEMANT;
24     }
25     public void sendToArduino(String data) throws Exception {
26         byte[] dataToSend = data.getBytes();
27         //send the datas to the serial device
28         if (this.usbSerialDriver != null){
29             this.usbSerialDriver.write(dataToSend, 500);
30         }
31     }
32     public void readFromToArduino(String data) throws Exception {
33     }
34     public void openUSBConnection () throws Exception {
35         //get a USB to Serial device object
36         this.usbSerialDriver =
37         UsbSerialProber.acquire(this.usbManager);
38         //open the device
39         this.usbSerialDriver.open();
40     }
41     public void startUSBConnection () throws Exception {
42         //set the communication speed
43         //make sure this matches your device's setting!
44         this.usbSerialDriver.setBaudRate(9600);
45     }
46     public void closeUSBConnection () throws Exception {
47         //close the device
48         this.usbSerialDriver.close();
49         this.usbSerialDriver = null;
50     }
51 }
52

```

Arquivo *activity_main.xml*, no qual é configurado o *layout* e disposição dos elementos gráficos

```

1 <RelativeLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"

```

```

4  android:layout_width="match_parent"
5      android:layout_height="match_parent"
6  android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:paddingBottom="@dimen/activity_vertical_margin"
10 tools:context=".MainActivity">
11     <Button
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="@string/button_serverStartStop"
15         android:id="@+id/button_serverStartStop"
16         android:background="#833544ed"
17         style="@android:style/MediaButton"
18         android:layout_alignParentTop="true"
19         android:layout_marginTop="175dp"
20         android:layout_alignParentRight="true"
21         android:layout_alignParentEnd="true"
22         android:layout_alignParentLeft="true"
23         android:layout_alignParentStart="true" />
24     <TextView
25         android:layout_width="wrap_content"
26         android:layout_height="wrap_content"
27         android:textAppearance="?android:attr/textAppearanceLarge"
28         android:text="@string/textView_serverStatus"
29         android:id="@+id/textView_serverStatus"
30         android:layout_centerVertical="true"
31         android:layout_alignParentLeft="true"
32         android:layout_alignParentStart="true" />
33     <TextView
34         android:layout_width="wrap_content"
35         android:layout_height="wrap_content"
36         android:textAppearance="?android:attr/textAppearanceLarge"
37         android:text="@string/textView_serverIP"
38         android:id="@+id/textView_severIP"
39         android:layout_above="@+id/textView_angleXValue"
40         android:layout_alignParentLeft="true"
41         android:layout_alignParentStart="true" />
42     <TextView
43         android:layout_width="wrap_content"
44         android:layout_height="wrap_content"

```


45	<code>android:textAppearance="?android:attr/textAppearanceLarge"</code>
46	<code>android:text="ÂnguloX:"</code>
47	<code>android:id="@+id/textView_angleXValue"</code>
48	<code>android:layout_alignParentBottom="true"</code>
49	<code>android:layout_alignParentLeft="true"</code>
50	<code>android:layout_alignParentStart="true"</code>
51	<code>android:layout_marginBottom="42dp" /></code>
52	<code><TextView</code>
53	<code>android:layout_width="wrap_content"</code>
54	<code>android:layout_height="wrap_content"</code>
55	<code>android:textAppearance="?android:attr/textAppearanceLarge"</code>
56	<code>android:text="ÂnguloY:"</code>
57	<code>android:id="@+id/textView_angleYValue"</code>
58	<code>android:layout_alignParentBottom="true"</code>
59	<code>android:layout_alignParentLeft="true"</code>
60	<code>android:layout_alignParentStart="true" /></code>
61	<code></RelativeLayout></code>
62	

Arquivo *strings.xml* onde são armazenados os textos usados nos elementos gráficos

1	<code><resources></code>
2	<code><string name="app_name">Aferidor</string></code>
3	<code><string name="action_settings">Settings</string></code>
4	<code><string name="button_serverStartStop">Iniciar Servidor</string></code>
5	<code><string name="textView_serverStatus">Servidor Parado</string></code>
6	<code><string name="textView_serverIP">IP:</string></code>
7	<code></resources></code>
8	

Arquivo *device_filter.xml* que está relacionado à identificação do dispositivo conectado à porta USB do *smartphone*

1	<code><?xml version="1.0" encoding="utf-8"?></code>
2	<code><resources></code>
3	<code><!-- 0x0403 / 0x6001: FTDI FT232R UART --></code>
4	<code><usb-device vendor-id="1027" product-id="24577" /></code>
5	<code><!-- 0x2341 / Arduino --></code>
6	<code><usb-device vendor-id="9025" /></code>
7	<code><!-- 0x16C0 / 0x0483: Teensyduino --></code>
8	<code><usb-device vendor-id="5824" product-id="1155" /></code>

9	<code><!-- 0x10C4 / 0xEA60: CP210x UART Bridge --></code>
10	<code><usb-device vendor-id="4292" product-id="60000" /></code>
11	
12	<code></resources></code>
13	

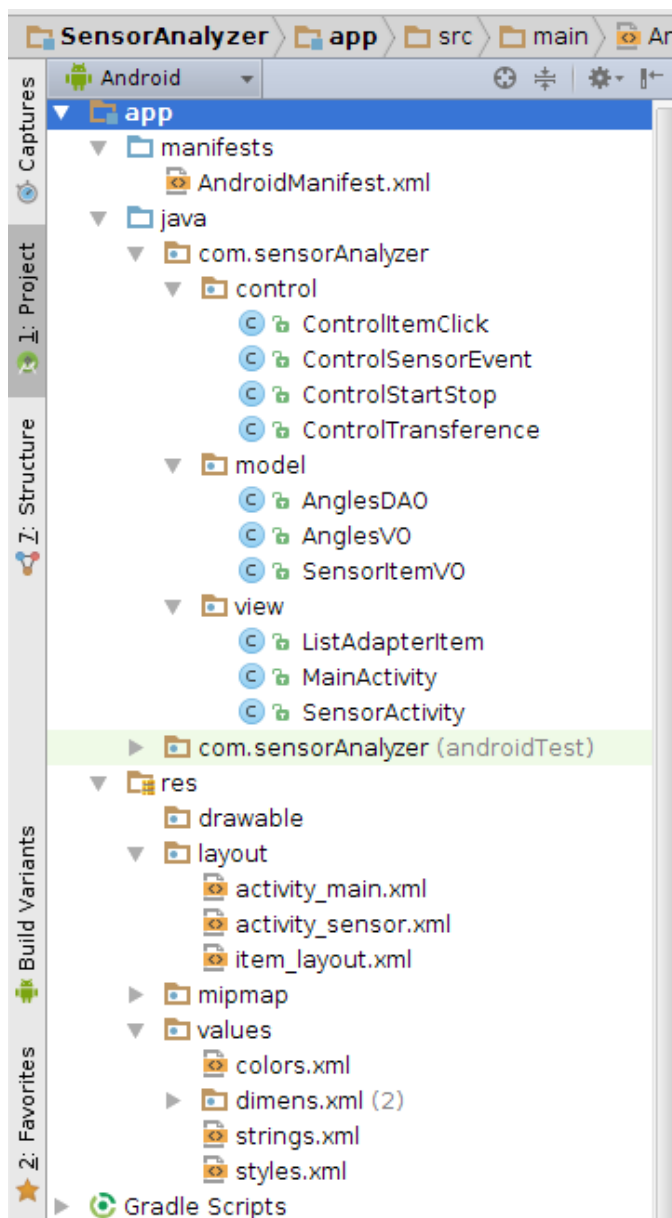
Arquivo *manifest.xml* onde é feita, entre outras, as configurações das permissões de acesso

1	<code><?xml version="1.0" encoding="utf-8"?></code>
2	<code><manifest xmlns:android="http://schemas.android.com/apk/res/android"</code>
3	<code>package="com.aferidor" ></code>
4	<code><uses-permission</code>
5	<code>android:name="android.permission.WRITE_EXTERNAL_STORAGE" /></code>
6	<code><uses-permission</code>
7	<code>android:name="android.permission.READ_EXTERNAL_STORAGE" /></code>
8	<code><uses-permission android:name="android.permission.INTERNET" /></code>
9	<code><uses-permission</code>
10	<code>android:name="android.permission.ACCESS_NETWORK_STATE" /></code>
11	<code><application</code>
12	<code>android:allowBackup="true"</code>
13	<code>android:icon="@mipmap/ic_launcher"</code>
14	<code>android:label="@string/app_name"</code>
15	<code>android:theme="@style/AppTheme" ></code>
16	<code><activity</code>
17	<code>android:name="com.aferidor.view.MainActivity"</code>
18	<code>android:label="@string/app_name" ></code>
19	<code><intent-filter></code>
20	<code><action android:name="android.intent.action.MAIN" /></code>
21	<code><action</code>
22	<code>android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" /></code>
23	<code><category</code>
24	<code>android:name="android.intent.category.LAUNCHER" /></code>
25	<code></intent-filter></code>
26	<code><meta-data</code>
27	
28	<code>android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"</code>
29	<code>android:resource="@xml/device_filter"/></code>
30	<code></activity></code>
31	<code></application></code>
32	<code></manifest></code>
33	

APÊNDICE B – Código fonte do aplicativo *sensor analyzer*

Este aplicativo foi desenvolvido utilizando o Android Studio e segue sua estrutura padrão de organização dos arquivos do código fonte, como mostrado na **Figura 5.2**.

Figura 5.2 – Organização de arquivos do código fonte do sensor analyzer



Fonte: elaborado pelo autor (2015)

Os arquivos criados e/ou editados estão listados a seguir.

Classe *MainActivity.java*

```

1  package com.sensorAnalyzer.view;
2  import android.hardware.Sensor;
3  import android.hardware.SensorManager;
4  import android.support.v7.app.AppCompatActivity;
5  import android.os.Bundle;
6  import android.widget.ListView;
7  import android.widget.TextView;
8  import com.sensorAnalyzer.R;
9  import com.sensorAnalyzer.model.SensorItemVO;
10 import java.util.Ar
11 rayList;
12 import java.util.List;
13 public class MainActivity extends AppCompatActivity {
14     private SensorManager sensor_manager;
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19         sensor_manager = (SensorManager)
20 getSystemService(SENSOR_SERVICE);
21         List<Sensor> deviceSensors =
22 sensor_manager.getSensorList(Sensor.TYPE_ALL);
23         ArrayList<SensorItemVO> listSensors = new
24 ArrayList<SensorItemVO>();
25         String numberOfSensores = "Total de sensores encontrados: "
26 + deviceSensors.size() + "\nInformações dos Sensores";
27         for (Sensor sensor : deviceSensors) {
28             SensorItemVO sensor_item = new
29 SensorItemVO(sensor.getName(), sensor.getType(), sensor.getVendor()
30 + "\n.....\n");
31             listSensors.add(sensor_item);
32         }
33         TextView textViewNumberOfSensors = (TextView)
34 findViewById(R.id.textViewNumberOfSensors);
35         textViewNumberOfSensors.setText(numberOfSensores);
36         ListAdapterItem listAdapterItem = new ListAdapterItem(this,
37 0, listSensors);
38         ListView listView = (ListView) findViewById(R.id.listView);
39         listView.setAdapter(listAdapterItem);
40     }
41 }

```

Classe *SensorActivity.java*

```

1  package com.sensorAnalyzer.view;
2  import android.graphics.Color;
3  import android.hardware.SensorManager;
4  import android.os.Bundle;
5  import android.support.v7.app.AppCompatActivity;
6  import android.widget.Button;
7  import android.widget.EditText;
8  import android.widget.Switch;
9  import android.widget.TextView;
10 import com.sensorAnalyzer.R;
11 import com.sensorAnalyzer.control.ControlSensorEvent;
12 import com.sensorAnalyzer.control.ControlStartStop;
13 public class SensorActivity extends AppCompatActivity {
14     private SensorManager sensorManager;
15     private ControlSensorEvent controlSensorEvent;
16     private TextView textView_x_axis, textView_y_axis,
17     textView_sensorName;
18     private Button button_startStop;
19     private EditText editText_nameFile, editText_IPToSendStatus;
20     private Switch switch_sendStatus, switch_saveDatas;
21     private int currentSensor;
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.activity_sensor);
26         this.sensorManager = (SensorManager)
27     getSystemService(SENSOR_SERVICE);
28         this.currentSensor =
29     getIntent().getExtras().getInt("sensorAtual");
30         this.textView_x_axis = (TextView) findViewById(R.id.X_value);
31         this.textView_y_axis = (TextView) findViewById(R.id.Y_value);
32         this.textView_sensorName = (TextView)
33     findViewById(R.id.sensor_name);
34
35     this.textView_sensorName.setText(this.sensorManager.getDefaultSensor(
36     this.currentSensor).getName());
37         this.editText_nameFile = (EditText)

```

```

38 findViewById(R.id.editText_nameFile);
39         this.editText_IPToSendStatus = (EditText)
40 findViewById(R.id.editText_IPToSendStatus);
41         this.button_startStop = (Button)
42 findViewById(R.id.button_startStop);
43         this.button_startStop.setOnClickListener(new
44 ControlStartStop(this));
45         this.switch_saveDatas = (Switch)
46 findViewById(R.id.switch_saveDatas);
47         this.switch_sendStatus = (Switch)
48 findViewById(R.id.switch_sendStatus);
49         this.controlSensorEvent = new ControlSensorEvent(this);
50     }
51     public TextView getTextView_x_axis() {
52         return textView_x_axis;
53     }
54     public TextView getTextView_y_axis() {
55         return textView_y_axis;
56     }
57     public TextView getTextView_sensorName() {
58         return textView_sensorName;
59     }
60     public Button getButton_startStop() {
61         return button_startStop;
62     }
63     public EditText getEditText_nameFile() {
64         return editText_nameFile;
65     }
66     public EditText getEditText_IPToSendStatus() {
67         return editText_IPToSendStatus;
68     }
69     public Switch getSwitch_sendStatus() {
70         return switch_sendStatus;
71     }
72     public Switch getSwitch_saveDatas() {
73         return switch_saveDatas;
74     }
75     public void registerSensorManager() {
76         sensorManager.registerListener(this.controlSensorEvent,
77 sensorManager.getDefaultSensor(this.currentSensor),
78 SensorManager.SENSOR_DELAY_FASTEST);

```

```

79         this.button_startStop.setText("Parar");
80
81     this.button_startStop.setBackgroundColor(Color.parseColor("#83BE2136"
82 ));
83     }
84     public void unregisterSensorManager() {
85         sensorManager.unregisterListener(this.controlSensorEvent);
86         this.button_startStop.setText("Iniciar");
87
88     this.button_startStop.setBackgroundColor(Color.parseColor("#833544ed"
89 ));
90     }
91 }
92

```

Classe ListAdapterItem .java

```

1  package com.sensorAnalyzer.view;
2  import android.content.Context;
3  import android.view.LayoutInflater;
4  import android.view.View;
5  import android.view.ViewGroup;
6  import android.widget.ArrayAdapter;
7  import android.widget.TextView;
8  import com.sensorAnalyzer.R;
9  import com.sensorAnalyzer.control.ControlItemClick;
10 import com.sensorAnalyzer.model.SensorItemVO;
11 import java.util.ArrayList;
12 /**
13  * Created by leonardo on 12/05/15.
14  */
15 public class ListAdapterItem extends ArrayAdapter<SensorItemVO> {
16     private Context context;
17     private int resource;
18     private ArrayList<SensorItemVO> list;
19     public ListAdapterItem(Context context, int resource,
20 ArrayList<SensorItemVO> list) {
21         super(context, resource, list);
22         this.context = context;
23         this.resource = resource;
24         this.list = list;

```



```

25     }
26     @Override
27     public View getView(int position, View convertView, ViewGroup
28 parent) {
29         SensorItemVO sensor_item = this.list.get(position);
30         convertView =
31 LayoutInflater.from(this.context).inflate(R.layout.item_layout, null);
32         TextView textViewName = (TextView)
33 convertView.findViewById(R.id.textViewSensorName)
34         ;
35         textViewName.setText(textViewName.getText() +
36 sensor_item.getName());
37         TextView textViewType = (TextView)
38 convertView.findViewById(R.id.textViewTypeSensor);
39         textViewType.setText(textViewType.getText() + "" +
40 sensor_item.getType());
41         TextView textViewManufacturer = (TextView)
42 convertView.findViewById(R.id.textViewSensorManufacturer);
43         textViewManufacturer.setText(textViewManufacturer.getText() +
44 sensor_item.getManufacturer());
45         convertView.setOnClickListener(new
46 ControlItemClick(sensor_item));
47         return convertView;
48     }
49 }
50

```

Classe *ControlItemClick.java*

```

1  package com.sensorAnalyzer.control;
2  import android.content.Intent;
3  import android.view.View;
4  import com.sensorAnalyzer.model.SensorItemVO;
5  import com.sensorAnalyzer.view.SensorActivity;
6  /**
7   * Created by leonardo on 13/05/15.
8   */
9  public class ControlItemClick implements View.OnClickListener {
10     private SensorItemVO sensorItemVO;
11     public ControlItemClick(SensorItemVO sensorItemVO) {
12         this.sensorItemVO = sensorItemVO;

```

```

13     }
14     /**
15      * Called when a view has been clicked.
16      *
17      * @param v The view that was clicked.
18      */
19     @Override
20     public void onClick(View v) {
21         //
22         MainActivity.setCurrentTypeSensor(this.sensorItemVO.getType());
23         Intent intent = new Intent(v.getContext(),
24             SensorActivity.class);
25         intent.putExtra("sensorAtual", this.sensorItemVO.getType());
26         v.getContext().startActivity(intent);
27     }
28 }
29

```

Classe ControlSensorEvent.java

```

1  package com.sensorAnalyzer.control;
2  import android.hardware.Sensor;
3  import android.hardware.SensorEvent;
4  import android.hardware.SensorEventListener;
5  import android.os.AsyncTask;
6  import android.widget.Toast;
7  import com.sensorAnalyzer.model.AnglesDAO;
8  import com.sensorAnalyzer.model.AnglesVO;
9  import com.sensorAnalyzer.view.SensorActivity;
10 import java.util.ArrayList;
11 /**
12  * Created by leonardo on 15/05/15.
13  */
14 public class ControlSensorEvent implements SensorEventListener {
15     private SensorActivity sensorActivity;
16     private float[] values;
17     private float x_old, y_old, minimumOscillation;
18     private int cont_toSave, maxValues_toSave;
19     public ControlSensorEvent(SensorActivity sensorActivity) {
20         this.sensorActivity = sensorActivity;
21         this.cont_toSave = 0;

```

```

22         this.minimumOscillation = 0.25f;
23         this.maxValues_toSave = 5;
24         this.x_old = this.y_old = 0f;
25     }
26     @Override
27     public void onSensorChanged(SensorEvent event) {
28         this.values = event.values;
29         this.cont_toSave++;
30         if (this.cont_toSave == this.maxValues_toSave) {
31             if (Math.abs(this.values[0] - this.x_old) >
32 this.minimumOscillation ||
33             Math.abs(this.values[1] - this.y_old) >
34 this.minimumOscillation) {
35                 this.x_old = this.values[0];
36                 this.y_old = this.values[1];
37             } else {
38                 this.values[0] = this.x_old;
39                 this.values[1] = this.y_old;
40             }
41             float x_angle = mapValueToAngle(this.values[0], -10f,
42 10f, -90, 90),
43             y_angle = mapValueToAngle(this.values[1], -10f,
44 10f, -90, 90);
45             if
46 (this.sensorActivity.getSwitch_saveDatas().isChecked()) {
47                 try {
48                     Object[] params = {
49                         "write",
50                         new AnglesVO(x_angle, y_angle),
51
52 this.sensorActivity.getEditText_nameFile().getText().toString()
53                     };
54                     AnglesDAO anglesDAO = new AnglesDAO();
55
56 anglesDAO.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,
57 params); // usando o executeOnExecutor para rodar multiplas AsyncTasks
58                 } catch (Exception e) {
59                     e.printStackTrace();
60                     Toast.makeText(this.sensorActivity, "Erro ao
61 salvar dados no arquivo CSV\n" + e.getMessage(),
62 Toast.LENGTH_LONG).show();

```

```

63         this.sensorActivity.unregisterSensorManager();
64     }
65 }
66 if
67 (this.sensorActivity.getSwitch_sendStatus().isChecked()) {
68     try {
69         ArrayList<Float> arrayListAngles = new
70 ArrayList<>();
71         arrayListAngles.add(x_angle);
72         arrayListAngles.add(y_angle);
73         Object[] params = {
74
75 this.sensorActivity.getEditText_IPToSendStatus().getText().toString()
76 , //IP do servidor de status
77         9999, //porta usada
78         arrayListAngles //dados que serão
79 enviados
80     };
81     ControlTransference controlTransference = new
82 ControlTransference();
83
84 controlTransference.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,
85 params); // usando o executeOnExecutor para rodar multiplas AsyncTask
86     } catch (Exception e) {
87         Toast.makeText(this.sensorActivity, "Erro ao
88 enviar status\n" + e.getMessage(), Toast.LENGTH_LONG).show();
89         this.sensorActivity.unregisterSensorManager();
90         e.printStackTrace();
91     }
92 }
93
94 this.sensorActivity.getTextView_sensorName().setText(event.sensor.get
95 Name());
96
97 this.sensorActivity.getTextView_x_axis().setText("'" +
98 x_angle);
99 this.sensorActivity.getTextView_y_axis().setText("'" +
100 y_angle);
101 this.cont_toSave = 0;
102 }
103 @Override

```

```

104     public void onAccuracyChanged(Sensor sensor, int accuracy) {
105     }
106     private int mapValueToAngle(float x, float input_min, float
107 input_max, int output_min, int output_max)
108     {
109         int angle = (int) ((x - input_min) * (output_max -
110 output_min) / (input_max - input_min) + output_min);
111         return angle;
112     }
113 }
114

```

Classe *ControlStartStop.java*

```

1  package com.sensorAnalyzer.control;
2  import android.view.View;
3  import com.sensorAnalyzer.view.SensorActivity;
4  /**
5   * Created by leonardo on 11/11/15.
6   */
7  public class ControlStartStop implements View.OnClickListener{
8      private SensorActivity sensorActivity;
9      public ControlStartStop(SensorActivity sensorActivity) {
10         this.sensorActivity = sensorActivity;
11     }
12     @Override
13     public void onClick(View v) {
14         if
15 (this.sensorActivity.getButton_startStop().getText().toString().equal
16 s("Parar"))
17             this.sensorActivity.unregisterSensorManager();
18         else
19             this.sensorActivity.registerSensorManager();
20     }
21 }
22

```

Classe *ControlTransference.java*

```

1  package com.sensorAnalyzer.control;
2  import android.os.AsyncTask;

```

```

3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.IOException;
6  import java.io.ObjectInputStream;
7  import java.io.ObjectOutputStream;
8  import java.io.PrintStream;
9  import java.net.Socket;
10 /**
11  * Created by leonardo on 05/06/15.
12  */
13 public class ControlTransference extends AsyncTask {
14     @Override
15     protected String doInBackground(Object[] params) {
16         Socket socket = null;
17         ObjectOutputStream objectOutputStream = null;
18         ObjectInputStream objectInputStream = null;
19         String resposta = null;
20         try {
21             socket = new Socket((String) params[0], (int)
22 params[1]);
23             objectOutputStream = new
24 ObjectOutputStream(socket.getOutputStream());
25             objectInputStream = new
26 ObjectInputStream(socket.getInputStream());
27             objectOutputStream.writeObject(params[2]);
28         } catch (Exception e) {
29             try {
30                 e.printStackTrace(new PrintStream(new
31 File("/storage/emulated/0/erros_ControlTransference.txt")));
32             } catch (FileNotFoundException e1) {
33                 e1.printStackTrace();
34             }
35         } finally {
36             try {
37                 if (socket != null)
38                     socket.close();
39                 if (objectInputStream != null)
40                     objectInputStream.close();
41                 if (objectOutputStream != null) {
42                     objectOutputStream.flush();
43                     objectOutputStream.close();

```

```

44         }
45     } catch (IOException e) {
46         e.printStackTrace();
47         try {
48             e.printStackTrace(new PrintStream(new
49 File("/storage/emulated/0/erros_ControlTransference.txt")));
50         } catch (FileNotFoundException e1) {
51             e1.printStackTrace();
52         }
53     }
54 }
55 return resposta;
56 }
57 }
58

```

Classe *SensorItemVO.java*

```

1  package com.sensorAnalyzer.model;
2
3  /**
4   * Created by leonardo on 11/05/15.
5   */
6  public class SensorItemVO {
7      private String name, manufacturer;
8      private int type;
9      public SensorItemVO(String name, int type, String manufacturer)
10     {
11         this.name = name;
12         this.type = type;
13         this.manufacturer = manufacturer;
14     }
15     public String getName() {
16         return this.name;
17     }
18     public int getType() {
19         return this.type;
20     }
21     public String getManufacturer() {
22         return this.manufacturer;
23     }
24 }

```

Classe *AnglesVO.java*

```

1  package com.sensorAnalyzer.model;
2  import java.io.Serializable;
3  /**
4   * Created by leonardo on 12/06/15.
5   */
6  public class AnglesVO implements Serializable {
7      private static final long serialVersionUID = 1L;
8      private float x, y;
9      public AnglesVO(float x, float y) {
10         this.x = x;
11         this.y = y;
12     }
13     public float getX() {
14         return x;
15     }
16     public float getY() {
17         return y;
18     }
19 }
20

```

Classe *AnglesDAO.java*

```

1  package com.sensorAnalyzer.model;
2  import android.os.AsyncTask;
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileWriter;
7  import java.io.PrintStream;
8  import java.text.SimpleDateFormat;
9  import java.util.Calendar;
10 /**
11  * Created by leonardo on 12/11/15.
12  */
13 public class AnglesDAO extends AsyncTask{
14     private SimpleDateFormat dateFormat = new

```



```

15 SimpleDateFormat("HH:mm:ss");
16     private void saveCoordinate (AnglesVO anglesVO, String nameFile)
17     throws Exception{
18         String lineWrite;
19         BufferedWriter bufferedWriter = new BufferedWriter(new
20 FileWriter("/storage/emulated/0/" + nameFile + ".csv", true));
21         lineWrite =
22 dateFormat.format(Calendar.getInstance().getTime()) + ";" +
23 anglesVO.getX() + ";" + anglesVO.getY();
24         lineWrite = lineWrite.replace('.', ',');
25         bufferedWriter.write(lineWrite);
26         bufferedWriter.newLine();
27         bufferedWriter.flush();
28         bufferedWriter.close();
29     }
30     @Override
31     protected Object doInBackground(Object[] params) {
32         switch ( (String) params[0]){
33             case "write":
34                 try {
35                     this.saveCoordinate( (AnglesVO) params[1],
36 (String) params[2] );
37                 } catch (Exception e) {
38                     e.printStackTrace();
39                     try {
40                         e.printStackTrace(new PrintStream(new
41 File("/storage/emulated/0/erros_AnglesDAO.txt")));
42                     } catch (FileNotFoundException e1) {
43                         e1.printStackTrace();
44                     }
45                 }
46                 break;
47             case "read":
48                 break;
49             case "alter":
50                 break;
51         }
52         return null;
53     }
54 }
55

```

Arquivo *activity_main.xml*, no qual é configurado o *layout* e disposição dos elementos gráficos da tela inicial

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4          xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6          android:layout_height="match_parent"
7      android:paddingLeft="@dimen/activity_horizontal_margin"
8          android:paddingRight="@dimen/activity_horizontal_margin"
9          android:paddingTop="@dimen/activity_vertical_margin"
10         android:paddingBottom="@dimen/activity_vertical_margin"
11     tools:context=".MainActivity">
12         <TextView
13             android:layout_width="wrap_content"
14             android:layout_height="wrap_content"
15             android:textAppearance="?android:attr/textAppearanceLarge"
16             android:text="Large Text"
17             android:id="@+id/textViewNumberOfSensors"
18             android:layout_alignParentTop="true"
19             android:layout_alignParentLeft="true"
20             android:layout_alignParentStart="true"
21             android:layout_alignRight="@+id/listView" />
22         <ListView
23             android:layout_width="wrap_content"
24             android:layout_height="wrap_content"
25             android:id="@+id/listView"
26             android:layout_alignParentLeft="true"
27             android:layout_alignParentStart="true"
28             android:layout_below="@+id/textViewNumberOfSensors"
29             android:background="#b2c4c4df" />
30     </RelativeLayout>
31

```

Arquivo *activity_sensor.xml*, no qual é configurado o *layout* e disposição dos elementos gráficos da tela de *status* do sensor

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout

```

```

3  xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:tools="http://schemas.android.com/tools"
5  android:layout_width="match_parent"
6      android:layout_height="match_parent"
7  android:paddingLeft="@dimen/activity_horizontal_margin"
8      android:paddingRight="@dimen/activity_horizontal_margin"
9      android:paddingTop="@dimen/activity_vertical_margin"
10     android:paddingBottom="@dimen/activity_vertical_margin"
11     tools:context="com.sensorAnalyzer.view.SensorActivity">
12     <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:textAppearance="?android:attr/textAppearanceLarge"
16         android:text="sensor_name"
17         android:id="@+id/sensor_name"
18         android:layout_alignParentTop="true"
19         android:layout_alignParentLeft="true"
20         android:layout_alignParentStart="true" />
21     <TextView
22         android:layout_width="wrap_content"
23         android:layout_height="wrap_content"
24         android:textAppearance="?android:attr/textAppearanceLarge"
25         android:text="Ângulo X:"
26         android:id="@+id/textView"
27         android:layout_marginTop="50dp"
28         android:layout_alignParentTop="true"
29         android:layout_alignParentLeft="true"
30         android:layout_alignParentStart="true" />
31     <TextView
32         android:layout_width="wrap_content"
33         android:layout_height="wrap_content"
34         android:textAppearance="?android:attr/textAppearanceLarge"
35         android:text="Ângulo Y:"
36         android:id="@+id/textView2"
37         android:layout_below="@+id/textView"
38         android:layout_alignParentLeft="true"
39         android:layout_alignParentStart="true"
40         android:layout_marginTop="23dp" />
41     <TextView
42         android:layout_width="wrap_content"
43         android:layout_height="wrap_content"

```

```

44         android:textAppearance="?android:attr/textAppearanceLarge"
45         android:text=" PARADO"
46         android:id="@+id/X_value"
47         android:layout_alignTop="@+id/textView"
48         android:layout_toRightOf="@+id/textView"
49         android:layout_toEndOf="@+id/textView" />
50     <TextView
51         android:layout_width="wrap_content"
52         android:layout_height="wrap_content"
53         android:textAppearance="?android:attr/textAppearanceLarge"
54         android:text=" PARADO"
55         android:id="@+id/Y_value"
56         android:layout_alignBottom="@+id/textView2"
57         android:layout_alignLeft="@+id/X_value"
58         android:layout_alignStart="@+id/X_value" />
59     <TextView
60         android:layout_width="wrap_content"
61         android:layout_height="wrap_content"
62         android:textAppearance="?android:attr/textAppearanceLarge"
63         android:text="@string/nameFile"
64         android:id="@+id/textView_fileName"
65         android:layout_above="@+id/editText_nameFile"
66         android:layout_alignParentLeft="true"
67         android:layout_alignParentStart="true" />
68     <EditText
69         android:layout_width="wrap_content"
70         android:layout_height="wrap_content"
71         android:id="@+id/editText_nameFile"
72         android:text="@string/datas"
73         android:layout_above="@+id/button_startStop"
74         android:layout_alignParentLeft="true"
75         android:layout_alignParentStart="true"
76         android:layout_alignRight="@+id/button_startStop"
77         android:layout_alignEnd="@+id/button_startStop" />
78     <Button
79         android:layout_width="wrap_content"
80         android:layout_height="wrap_content"
81         android:text="@string/button_saveStart"
82         android:id="@+id/button_startStop"
83         android:layout_alignParentBottom="true"
84         android:layout_alignParentLeft="true"

```

```

85         android:layout_alignParentStart="true"
86         android:layout_alignParentRight="true"
87         android:layout_alignParentEnd="true"
88         android:background="#833544ed" />
89     <TextView
90         android:layout_width="wrap_content"
91         android:layout_height="wrap_content"
92         android:textAppearance="?android:attr/textAppearanceLarge"
93         android:text="@string/IP_toSend"
94         android:id="@+id/textView_IPToSendStatus"
95         android:layout_centerVertical="true"
96         android:layout_alignParentLeft="true"
97         android:layout_alignParentStart="true" />
98     <EditText
99         android:layout_width="wrap_content"
100        android:layout_height="wrap_content"
101        android:id="@+id/editText_IPToSendStatus"
102        android:text="@string/IP_toSendStatus"
103        android:layout_below="@+id/textView_IPToSendStatus"
104        android:layout_alignParentLeft="true"
105        android:layout_alignParentStart="true"
106        android:layout_alignRight="@+id/editText_nameFile"
107        android:layout_alignEnd="@+id/editText_nameFile" />
108    <Switch
109        android:layout_width="wrap_content"
110        android:layout_height="wrap_content"
111        android:text="@string/send_status"
112        android:id="@+id/switch_sendStatus"
113        android:checked="true"
114        android:textAppearance="?android:attr/textAppearanceLarge"
115        android:layout_above="@+id/textView_IPToSendStatus"
116        android:layout_alignParentLeft="true"
117        android:layout_alignParentStart="true" />
118    <Switch
119        android:layout_width="wrap_content"
120        android:layout_height="wrap_content"
121        android:text="@string/saveDatas"
122        android:id="@+id/switch_saveDatas"
123        android:layout_above="@+id/textView_fileName"
124        android:layout_alignParentLeft="true"
125        android:layout_alignParentStart="true"

```

126	<code>android:checked="true"</code>
127	<code>android:textAppearance="?android:attr/textAppearanceLarge"</code>
128	<code>/></code>
129	<code></RelativeLayout></code>
130	

Arquivo *item_layout.xml*, no qual é configurado o *layout* e disposição dos elementos gráficos para cada item da lista de sensores exibida na tela inicial

1	<code><?xml version="1.0" encoding="utf-8" ?></code>
2	<code><LinearLayout</code>
3	<code>xmlns:android="http://schemas.android.com/apk/res/android"</code>
4	<code>android:orientation="vertical"</code>
5	<code>android:layout_width="match_parent"</code>
6	<code>android:layout_height="match_parent"></code>
7	<code><TextView</code>
8	<code>android:layout_width="wrap_content"</code>
9	<code>android:layout_height="wrap_content"</code>
10	<code>android:textAppearance="?android:attr/textAppearanceLarge"</code>
11	<code>android:text="@string/sensorName"</code>
12	<code>android:id="@+id/textViewSensorName"</code>
13	<code>android:layout_alignParentTop="true"</code>
14	<code>android:layout_alignParentLeft="true"</code>
15	<code>android:layout_alignParentStart="true" /></code>
16	<code><TextView</code>
17	<code>android:layout_width="wrap_content"</code>
18	<code>android:layout_height="wrap_content"</code>
19	<code>android:textAppearance="?android:attr/textAppearanceLarge"</code>
20	<code>android:text="@string/sensorType"</code>
21	<code>android:id="@+id/textViewTypeSensor"</code>
22	<code>android:layout_below="@+id/textViewSensorName"</code>
23	<code>android:layout_alignParentLeft="true"</code>
24	<code>android:layout_alignParentStart="true" /></code>
25	<code><TextView</code>
26	<code>android:layout_width="wrap_content"</code>
27	<code>android:layout_height="wrap_content"</code>
28	<code>android:textAppearance="?android:attr/textAppearanceLarge"</code>
29	<code>android:text="@string/sensorManufacturer"</code>
30	<code>android:id="@+id/textViewSensorManufacturer"</code>
31	<code>android:layout_below="@+id/textViewTypeSensor"</code>
32	<code>android:layout_alignParentLeft="true"</code>

33	<code>android:layout_alignParentStart="true" /></code>
34	<code></LinearLayout></code>
35	

Arquivo *strings.xml* onde são armazenados os textos usados nos elementos gráficos

1	<code><resources></code>
2	<code> <string name="app_name">Sensor Analyzer</string></code>
3	<code> <string name="saveDatas">Salvar Dados</string></code>
4	<code> <string name="nameFile">Digite nome do Arquivo</string></code>
5	<code> <string name="datas">Dados</string></code>
6	<code> <string name="button_saveStart">Iniciar</string></code>
7	<code> <string name="sensorName">Nome:</string></code>
8	<code> <string name="sensorType">Tipo:</string></code>
9	<code> <string name="sensorManufacturer">Fabricante:</string></code>
10	<code> <string name="send_status">Enviar Status</string></code>
11	<code> <string name="IP_toSend">IP para enviar Status</string></code>
12	<code> <string name="IP_toSendStatus">192.168.43.47</string></code>
13	<code></resources></code>
14	

Arquivo *manifest.xml* onde é feita, entre outras, as configurações das permissões de acesso

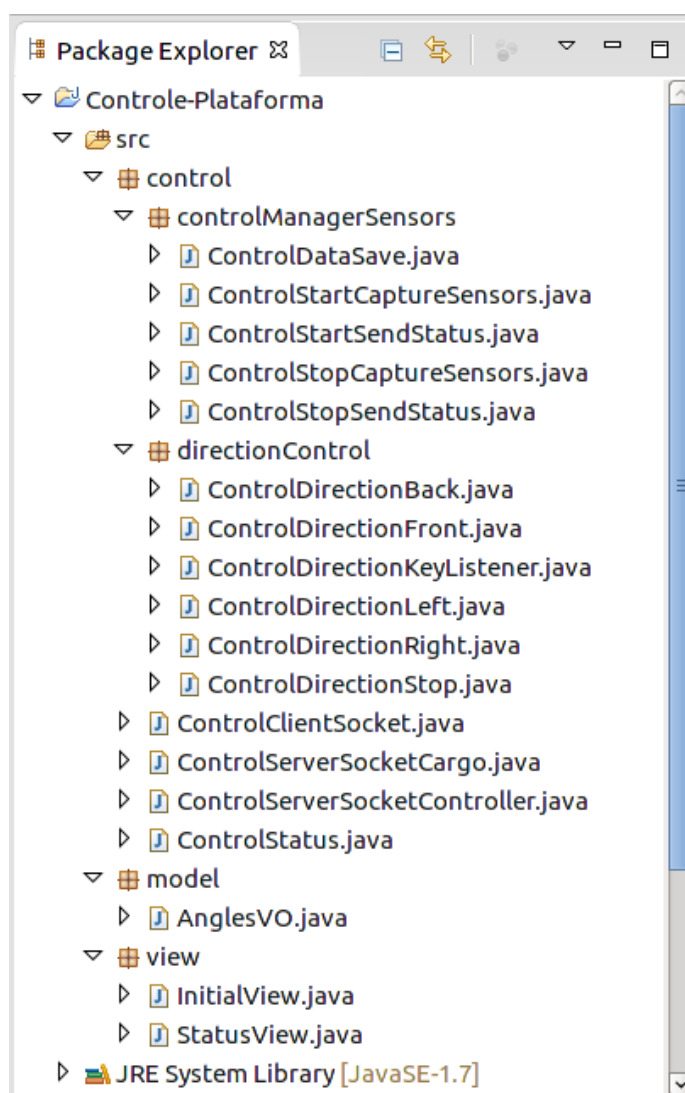
1	<code><?xml version="1.0" encoding="utf-8"?></code>
2	<code><manifest xmlns:android="http://schemas.android.com/apk/res/android"</code>
3	<code> package="com.sensorAnalyzer" ></code>
4	<code> <uses-permission</code>
5	<code> android:name="android.permission.WRITE_EXTERNAL_STORAGE" /></code>
6	<code> <uses-permission</code>
7	<code> android:name="android.permission.READ_EXTERNAL_STORAGE" /></code>
8	<code> <uses-permission android:name="android.permission.INTERNET" /></code>
9	<code> <uses-permission</code>
10	<code> android:name="android.permission.ACCESS_NETWORK_STATE" /></code>
11	<code> <application</code>
12	<code> android:allowBackup="true"</code>
13	<code> android:icon="@mipmap/ic_launcher"</code>
14	<code> android:label="@string/app_name"</code>
15	<code> android:supportRtl="true"</code>
16	<code> android:theme="@style/AppTheme" ></code>
17	<code> <activity android:name=".view.MainActivity" ></code>
18	<code> <intent-filter></code>

```
19         <action android:name="android.intent.action.MAIN" />
20         <category
21 android:name="android.intent.category.LAUNCHER" />
22     </intent-filter>
23 </activity>
24 <activity android:name=".view.SensorActivity" >
25 </activity>
26 </application>
27 </manifest>
28
```


APÊNDICE C – Código fonte do software de controle

Este aplicativo foi desenvolvido utilizando o IDE Eclipse e a estrutura de organização dos arquivos do código fonte é mostrada na **Figura 5.3**.

Figura 5.3 – Organização de arquivos do código fonte do software de controle



Fonte: elaborado pelo autor (2015)

Os arquivos criados e/ou editados estão listados a seguir.

Classe *InitialView.java*

```

1  package view;
2
3  import java.awt.Color;
4  import java.awt.Font;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.net.InetAddress;
8  import java.net.NetworkInterface;
9  import java.net.SocketException;
10 import java.util.Enumuration;
11
12 import javax.swing.JButton;
13 import javax.swing.JCheckBox;
14 import javax.swing.JFrame;
15 import javax.swing.JLabel;
16 import javax.swing.JTextField;
17
18 import control.controlManagerSensors.ControlDataSave;
19 import control.controlManagerSensors.ControlStartCaptureSensors;
20 import control.controlManagerSensors.ControlStartSendStatus;
21 import control.controlManagerSensors.ControlStopCaptureSensors;
22 import control.controlManagerSensors.ControlStopSendStatus;
23 import control.directionControl.ControlDirectionBack;
24 import control.directionControl.ControlDirectionFront;
25 import control.directionControl.ControlDirectionKeyListener;
26 import control.directionControl.ControlDirectionLeft;
27 import control.directionControl.ControlDirectionRight;
28 import control.directionControl.ControlDirectionStop;
29
30 public class InitialView extends JFrame {
31
32     private static final long serialVersionUID = 1L;
33
34     private JTextField textField_IP;
35     private JCheckBox chckBoxDataSave;
36     private StatusView statusView;
37     private JLabel lbl_IP_local;
38
39     public InitialView() {
40         getContentPane().setBackground(new Color(240, 248, 255));
41

```

```

42         try {
43
44             javax.swing.UIManager.setLookAndFeel("com.sun.java.swing.plaf.n
45 imbus.NimbusLookAndFeel");
46         } catch (Exception e) {
47             e.printStackTrace();
48         }
49
50         setSize(500, 569);
51         setTitle("Controle Plataforma");
52         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
53         setResizable(false);
54         setLocation(770, 100);
55         getContentPane().setLayout(null);
56
57         JLabel lbl_IP_smartphone = new JLabel("IP do
58 Smartphone");
59         lbl_IP_smartphone.setFont(new Font("Dialog", Font.BOLD,
60 14));
61         lbl_IP_smartphone.setBounds(36, 15, 140, 17);
62         getContentPane().add(lbl_IP_smartphone);
63
64         textField_IP = new JTextField();
65         textField_IP.setFont(new Font("Dialog", Font.PLAIN, 14));
66         textField_IP.setText("192.168.43.1");
67         textField_IP.setBounds(181, 12, 125, 24);
68         getContentPane().add(textField_IP);
69         textField_IP.setColumns(10);
70
71         JLabel lblIP = new JLabel("IP local para status");
72         lblIP.setFont(new Font("Dialog", Font.BOLD, 14));
73         lblIP.setBounds(27, 44, 149, 17);
74         getContentPane().add(lblIP);
75
76         lbl_IP_local = new JLabel(getIp());
77         lbl_IP_local.setFont(new Font("Dialog", Font.BOLD, 13));
78         lbl_IP_local.setBounds(181, 45, 125, 15);
79         getContentPane().add(lbl_IP_local);
80
81         JButton btnStartCapture = new JButton("Iniciar captura de
82 sensores");

```

```

83         btnStartCapture.setBackground(new Color(70, 130, 180));
84         btnStartCapture.setBounds(37, 126, 228, 35);
85         btnStartCapture.addActionListener(new
86 ControlStartCaptureSensors(this));
87         getContentPane().add(btnStartCapture);
88
89         JButton btnStopCapture = new JButton("Parar captura de
90 sensores");
91         btnStopCapture.setBackground(new Color(205, 92, 92));
92         btnStopCapture.setBounds(37, 173, 228, 35);
93         btnStopCapture.addActionListener(new
94 ControlStopCaptureSensors(this));
95         getContentPane().add(btnStopCapture);
96
97         JButton btnShowStatus = new JButton("Exibir Status");
98         btnShowStatus.setBackground(new Color(70, 130, 180));
99         btnShowStatus.setBounds(295, 126, 170, 35);
100        btnShowStatus.addActionListener(new
101 ControlStartSendStatus(this));
102        getContentPane().add(btnShowStatus);
103
104        JButton btnNewButton = new JButton("Fechar Status");
105        btnNewButton.setBackground(new Color(205, 92, 92));
106        btnNewButton.setBounds(295, 173, 170, 35);
107        btnNewButton.addActionListener(new
108 ControlStopSendStatus(this));
109        getContentPane().add(btnNewButton);
110
111        final JButton btnFront = new JButton("Frente");
112        btnFront.setBounds(189, 286, 117, 45);
113        btnFront.addActionListener(new
114 ControlDirectionFront(this));
115        getContentPane().add(btnFront);
116
117        final JButton btnRight = new JButton("Direita");
118        btnRight.setBounds(326, 361, 117, 45);
119        btnRight.addActionListener(new
120 ControlDirectionRight(this));
121        getContentPane().add(btnRight);
122
123        final JButton btnBack = new JButton("Atr\u00E9s");

```

```

124         btnBack.setBounds(189, 436, 117, 45);
125         btnBack.addActionListener(new
126 ControlDirectionBack(this));
127         getContentPane().add(btnBack);
128
129         final JButton btnLeft = new JButton("Esquerda");
130         btnLeft.setBounds(52, 361, 117, 45);
131         btnLeft.addActionListener(new
132 ControlDirectionLeft(this));
133         getContentPane().add(btnLeft);
134
135         final JButton btnStop = new JButton("Parar");
136         btnStop.setBounds(189, 361, 117, 45);
137         btnStop.addActionListener(new
138 ControlDirectionStop(this));
139         getContentPane().add(btnStop);
140
141         chckBoxDataSave = new JCheckBox("Salvar Dados em Arquivo
142 no Smartphone");
143         chckBoxDataSave.setBounds(27, 98, 313, 23);
144         chckBoxDataSave.addActionListener(new
145 ControlDataSave(this));
146         getContentPane().add(chckBoxDataSave);
147
148         final JCheckBox chckbxUseKeysDirections = new
149 JCheckBox("Usar teclas direcionais do teclado");
150         chckbxUseKeysDirections.setBounds(141, 256, 266, 23);
151         getContentPane().add(chckbxUseKeysDirections);
152
153         chckbxUseKeysDirections.addKeyListener(new
154 ControlDirectionKeyListener(this));
155         chckbxUseKeysDirections.addActionListener(new
156 ActionListener() {
157
158             @Override
159             public void actionPerformed(ActionEvent e) {
160                 if (chckbxUseKeysDirections.isSelected()) {
161                     btnFront.setEnabled(false);
162                     btnRight.setEnabled(false);
163                     btnBack.setEnabled(false);
164                     btnLeft.setEnabled(false);

```

```

165         btnStop.setEnabled(false);
166
167         } else {
168             btnFront.setEnabled(true);
169             btnRight.setEnabled(true);
170             btnBack.setEnabled(true);
171             btnLeft.setEnabled(true);
172             btnStop.setEnabled(true);
173         }
174
175     }
176
177     });
178
179     setVisible(true);
180
181 }
182
183 public StatusView getStatusView() {
184     return statusView;
185 }
186
187 public void setStatusView(StatusView statusView) {
188     this.statusView = statusView;
189 }
190
191 public JTextField getTextField_IP() {
192     return textField_IP;
193 }
194
195 public JCheckBox getChckBoxDataSave() {
196     return chckBoxDataSave;
197 }
198
199 public JLabel getLbl_IP_local() {
200     return lbl_IP_local;
201 }
202
203 public String getIp() {
204     String ipAddress = null;
205     Enumeration<NetworkInterface> net = null;
206     try {

```

```

206         net = NetworkInterface.getNetworkInterfaces();
207     } catch (SocketException e) {
208         throw new RuntimeException(e);
209     }
210
211     while (net.hasMoreElements()) {
212         NetworkInterface element = net.nextElement();
213         Enumeration<InetAddress> addresses =
214 element.getInetAddresses();
215         while (addresses.hasMoreElements()) {
216             InetAddress ip = addresses.nextElement();
217
218             if (ip.isSiteLocalAddress()) {
219                 ipAddress = ip.getHostAddress();
220             }
221         }
222     }
223     return ipAddress;
224 }
225
226 public static void main(String[] args) {
227     new InitialView();
228 }
229 }
230

```

Classe *StatusView.java*

```

1  package view;
2
3  import java.awt.Color;
4  import java.awt.Font;
5  import java.util.ArrayList;
6
7  import javax.swing.JCheckBox;
8  import javax.swing.JFrame;
9  import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 import com.jogamp.opengl.GLCapabilities;
13 import com.jogamp.opengl.GLProfile;

```

```

14 import com.jogamp.opengl.awt.GLCanvas;
15 import com.jogamp.opengl.util.FPSAnimator;
16
17 import control.ControlServerSocketCargo;
18 import control.ControlServerSocketController;
19 import control.ControlStatus;
20
21 public class StatusView extends JFrame {
22
23     private static final long serialVersionUID = 1L;
24     public ArrayList<Float> angles_smart_controller,
25 angles_smart_cargo;
26     private JCheckBox chckbx_showAngleYCargo;
27     private JCheckBox chckbx_showAngleXCargo;
28     private JCheckBox chckbx_showAngleXController;
29     private JCheckBox chckbx_showAngleYController;
30
31     public StatusView() {
32         setTitle("Status Plataforma");
33         setSize(960, 700);
34         setLocation(0, 0);
35         setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
36         setResizable(false);
37         getContentPane().setBackground(Color.WHITE);
38         getContentPane().setLayout(null);
39
40         this.angles_smart_cargo = new ArrayList<Float>();
41         this.angles_smart_cargo.add(0f);
42         this.angles_smart_cargo.add(0f);
43
44         this.angles_smart_controller = new ArrayList<Float>();
45         this.angles_smart_controller.add(0f);
46         this.angles_smart_controller.add(0f);
47
48         int x_labelsLeft = 9;
49         int x_labelsRight = 931;
50
51         // JLabels left
52         JLabel label_40 = new JLabel("40");
53         label_40.setBounds(x_labelsLeft, 82, 16, 15);
54         getContentPane().add(label_40);

```



```
55
56         JLabel label_35 = new JLabel("35");
57         label_35.setBounds(x_labelsLeft, 114, 16, 15);
58         getContentPane().add(label_35);
59
60         JLabel label_30 = new JLabel("30");
61         label_30.setBounds(x_labelsLeft, 147, 16, 15);
62         getContentPane().add(label_30);
63
64         JLabel label_25 = new JLabel("25");
65         label_25.setBounds(x_labelsLeft, 179, 16, 15);
66         getContentPane().add(label_25);
67
68         JLabel label_20 = new JLabel("20");
69         label_20.setBounds(x_labelsLeft, 211, 16, 15);
70         getContentPane().add(label_20);
71
72         JLabel label_15 = new JLabel("15");
73         label_15.setBounds(x_labelsLeft, 243, 16, 15);
74         getContentPane().add(label_15);
75
76         JLabel label_10 = new JLabel("10");
77         label_10.setBounds(x_labelsLeft, 275, 16, 15);
78         getContentPane().add(label_10);
79
80         JLabel label_5 = new JLabel(" 5");
81         label_5.setBounds(x_labelsLeft, 306, 16, 15);
82         getContentPane().add(label_5);
83
84         // JLabel label_0 = new JLabel(" 0");
85         // label_0.setBounds(x_labelsLeft, 341, 16, 15);
86         // getContentPane().add(label_0);
87
88         JLabel label_5n = new JLabel("-5");
89         label_5n.setBounds(x_labelsLeft, 373, 16, 15);
90         getContentPane().add(label_5n);
91
92         JLabel label_10n = new JLabel("-10");
93         label_10n.setBounds(x_labelsLeft, 406, 21, 15);
94         getContentPane().add(label_10n);
95
```

```

96      JLabel label_15n = new JLabel("-15");
97      label_15n.setBounds(x_labelsLeft, 438, 21, 15);
98      getContentPane().add(label_15n);
99
100     JLabel label_20n = new JLabel("-20");
101     label_20n.setBounds(x_labelsLeft, 472, 21, 15);
102     getContentPane().add(label_20n);
103
104     JLabel label_25n = new JLabel("-25");
105     label_25n.setBounds(x_labelsLeft, 503, 21, 15);
106     getContentPane().add(label_25n);
107
108     JLabel label_30n = new JLabel("-30");
109     label_30n.setBounds(x_labelsLeft, 536, 21, 15);
110     getContentPane().add(label_30n);
111
112     JLabel label_35n = new JLabel("-35");
113     label_35n.setBounds(x_labelsLeft, 569, 21, 15);
114     getContentPane().add(label_35n);
115
116     JLabel label_40n = new JLabel("-40");
117     label_40n.setBounds(x_labelsLeft, 601, 21, 15);
118     getContentPane().add(label_40n);
119
120     //      JLabels right
121     JLabel label_40_right = new JLabel("40");
122     label_40_right.setBounds(x_labelsRight, 82, 16, 15);
123     getContentPane().add(label_40_right);
124
125     JLabel label_35_right = new JLabel("35");
126     label_35_right.setBounds(x_labelsRight, 114, 16, 15);
127     getContentPane().add(label_35_right);
128
129     JLabel label_30_right = new JLabel("30");
130     label_30_right.setBounds(x_labelsRight, 147, 16, 15);
131     getContentPane().add(label_30_right);
132
133     JLabel label_25_right = new JLabel("25");
134     label_25_right.setBounds(x_labelsRight, 179, 16, 15);
135     getContentPane().add(label_25_right);
136

```

```

137 JLabel label_20_right = new JLabel("20");
138 label_20_right.setBounds(x_labelsRight, 211, 16, 15);
139 getContentPane().add(label_20_right);
140
141 JLabel label_15_right = new JLabel("15");
142 label_15_right.setBounds(x_labelsRight, 243, 16, 15);
143 getContentPane().add(label_15_right);
144
145 JLabel label_10_right = new JLabel("10");
146 label_10_right.setBounds(x_labelsRight, 275, 16, 15);
147 getContentPane().add(label_10_right);
148
149 JLabel label_5_right = new JLabel(" 5");
150 label_5_right.setBounds(x_labelsRight, 306, 16, 15);
151 getContentPane().add(label_5_right);
152
153 // JLabel label_0_right = new JLabel(" 0");
154 // label_0_right.setBounds(x_labelsRight, 341, 16, 15);
155 // getContentPane().add(label_0_right);
156
157 JLabel label_5n_right = new JLabel("-5");
158 label_5n_right.setBounds(x_labelsRight, 373, 16, 15);
159 getContentPane().add(label_5n_right);
160
161 JLabel label_10n_right = new JLabel("-10");
162 label_10n_right.setBounds(x_labelsRight, 406, 21, 15);
163 getContentPane().add(label_10n_right);
164
165 JLabel label_15n_right = new JLabel("-15");
166 label_15n_right.setBounds(x_labelsRight, 438, 21, 15);
167 getContentPane().add(label_15n_right);
168
169 JLabel label_20n_right = new JLabel("-20");
170 label_20n_right.setBounds(x_labelsRight, 472, 21, 15);
171 getContentPane().add(label_20n_right);
172
173 JLabel label_25n_right = new JLabel("-25");
174 label_25n_right.setBounds(x_labelsRight, 503, 21, 15);
175 getContentPane().add(label_25n_right);
176
177 JLabel label_30n_right = new JLabel("-30");

```

```

178 label_30n_right.setBounds(x_labelsRight, 536, 21, 15);
179 getContentPane().add(label_30n_right);
180
181 JLabel label_35n_right = new JLabel("-35");
182 label_35n_right.setBounds(x_labelsRight, 569, 21, 15);
183 getContentPane().add(label_35n_right);
184
185 JLabel label_40n_right = new JLabel("-40");
186 label_40n_right.setBounds(x_labelsRight, 601, 21, 15);
187 getContentPane().add(label_40n_right);
188
189 JLabel lblNewLabel = new JLabel("Legenda");
190 lblNewLabel.setFont(new Font("Dialog", Font.BOLD, 16));
191 lblNewLabel.setBounds(139, 23, 75, 19);
192 getContentPane().add(lblNewLabel);
193
194 chckbx_showAngleXController = new JCheckBox("Exibir
195 \u00E2ngulo X do aferidor");
196 chckbx_showAngleXController.setSelected(true);
197 chckbx_showAngleXController.setBounds(228, 8, 186, 18);
198 getContentPane().add(chckbx_showAngleXController);
199
200 chckbx_showAngleYController = new JCheckBox("Exibir
201 \u00E2ngulo Y do aferidor");
202 chckbx_showAngleYController.setSelected(true);
203 chckbx_showAngleYController.setBounds(228, 35, 186,
204 18);
205 getContentPane().add(chckbx_showAngleYController);
206
207 JLabel lblNewLabel_1 = new JLabel("_____");
208 lblNewLabel_1.setBounds(423, 7, 48, 15);
209 getContentPane().add(lblNewLabel_1);
210
211 JLabel label = new JLabel("_____");
212 label.setForeground(Color.GREEN);
213 label.setBounds(423, 30, 48, 15);
214 getContentPane().add(label);
215
216 chckbx_showAngleXCargo = new JCheckBox("Exibir
217 \u00E2ngulo X da carga");
218 chckbx_showAngleXCargo.setSelected(true);

```

```

219         chckbx_showAngleXCargo.setBounds(524, 9, 173, 18);
220         getContentPane().add(chckbx_showAngleXCargo);
221
222         chckbx_showAngleYCargo = new JCheckBox("Exibir
223 \u00E2ngulo Y da carga");
224         chckbx_showAngleYCargo.setSelected(true);
225         chckbx_showAngleYCargo.setBounds(524, 36, 173, 18);
226         getContentPane().add(chckbx_showAngleYCargo);
227
228         JLabel label_1 = new JLabel("_____");
229         label_1.setForeground(Color.BLUE);
230         label_1.setBounds(706, 8, 48, 15);
231         getContentPane().add(label_1);
232
233         JLabel label_2 = new JLabel("_____");
234         label_2.setForeground(Color.RED);
235         label_2.setBounds(706, 31, 48, 15);
236         getContentPane().add(label_2);
237
238         JPanel panel = new JPanel();
239         panel.setBackground(Color.WHITE);
240         panel.setBounds(6, 0, 946, 60);
241         getContentPane().add(panel);
242
243         final GLProfile profile = GLProfile.get(GLProfile.GL2);
244         GLCapabilities capabilities = new
245 GLCapabilities(profile);
246         // The canvas
247         final GLCanvas glcanvas = new GLCanvas(capabilities);
248         ControlStatus controlEventCube = new
249 ControlStatus(this);
250         glcanvas.addGLEventListener(controlEventCube);
251         glcanvas.setSize(960, 700);
252         glcanvas.setLocation(0, 0);
253         getContentPane().add(glcanvas);
254
255         setVisible(true);
256
257         new ControlServerSocketController(this);
258         new ControlServerSocketCargo(this);
259

```

```

260         final FPSAnimator animator = new FPSAnimator(glcanvas,
261 16, true);
262         animator.start();
263
264     }
265
266     public JCheckBox getChckbx_showAngleYCargo() {
267         return chckbx_showAngleYCargo;
268     }
269
270     public JCheckBox getChckbx_showAngleXCargo() {
271         return chckbx_showAngleXCargo;
272     }
273
274     public JCheckBox getChckbx_showAngleXController() {
275         return chckbx_showAngleXController;
276     }
277
278     public JCheckBox getChckbx_showAngleYController() {
279         return chckbx_showAngleYController;
280     }
281
282 }
283

```

Classe *ControlStatus.java*

```

1  package control;
2
3  import java.util.ArrayList;
4
5  import model.AnglesVO;
6  import view.StatusView;
7
8  import com.jogamp.opengl.GL2;
9  import com.jogamp.opengl.GLAutoDrawable;
10 import com.jogamp.opengl.GLEventListener;
11 import com.jogamp.opengl.glu.GLU;
12
13 public class ControlStatus implements GLEventListener {
14

```

```

15 // public static DisplayMode dm, dm_old;
16 private GLU glu = new GLU();
17 private float x_interval;
18 private StatusView statusView;
19 private ArrayList<AnglesVO>
20 anglesSmartController,
21 anglesSmartCargo;
22 int maxNamberAngles, drawingStartPoint;
23
24 public ControlStatus(StatusView statusView) {
25     super();
26     this.statusView = statusView;
27     this.anglesSmartController = new ArrayList<AnglesVO>();
28     this.anglesSmartController.add(new AnglesVO(0f, 0f));
29
30     this.anglesSmartCargo = new ArrayList<AnglesVO>();
31     this.anglesSmartCargo.add(new AnglesVO(0f, 0f));
32
33     this.x_interval = 0.8f;
34     this.maxNamberAngles = 158;
35     this.drawingStartPoint = 128;
36
37 }
38
39 @Override
40 public void display(GLAutoDrawable drawable) {
41
42     if ( !(this.statusView.angles_smart_controller.get(0) ==
43 0 && this.statusView.angles_smart_controller.get(1) == 0) ) {
44
45         if (this.anglesSmartController.size() ==
46 this.maxNamberAngles)
47             this.anglesSmartController.remove(0);
48
49             this.anglesSmartController.add(new AnglesVO(
50 this.statusView.angles_smart_controller.get(0),
51 this.statusView.angles_smart_controller.get(1)));
52         }
53
54         if ( !(this.statusView.angles_smart_cargo.get(0) == 0 &&
55 this.statusView.angles_smart_cargo.get(1) == 0) ) {

```

```

56
57         if (this.anglesSmartCargo.size() ==
58 this.maxNamberAngles)
59             this.anglesSmartCargo.remove(0);
60
61             this.anglesSmartCargo.add(new AnglesVO(
62 this.statusView.angles_smart_cargo.get(0),
63 this.statusView.angles_smart_cargo.get(1)));
64
65         }
66
67         final GL2 gl = drawable.getGL().getGL2();
68         gl.glClear(GL2.GL_COLOR_BUFFER_BIT |
69 GL2.GL_DEPTH_BUFFER_BIT);
70
71         gl.glLoadIdentity();
72         gl.glTranslatef(-67f, 0f, -130f);
73
74         // float scale = 3f;
75         // gl.glScalef(scale, scale, scale);
76
77         float axis_size_x = 160f, axis_size_y = 48f;
78
79         // Eixo X - Preto
80         gl.glBegin(GL2.GL_LINES);
81         gl.glColor3f(0f, 0f, 0f);
82         gl.glVertex2f(-axis_size_x, 0f);
83         gl.glVertex2f(axis_size_x, 0f);
84         gl.glEnd();
85
86         // Eixo Y - Preto
87         gl.glBegin(GL2.GL_LINES);
88         gl.glColor3f(0f, 0f, 0f);
89         gl.glVertex2f(0f, -axis_size_y);
90         gl.glVertex2f(0f, axis_size_y);
91         gl.glEnd();
92
93         // Desenhar linhas paralelas ao eixo X
94         gl.glBegin(GL2.GL_LINES);
95         gl.glColor3f(0.66f, 0.66f, 0.66f);
96         for (float i = -40f; i <= 40; i += 5) {

```



```

97         if (i != 0) { // para não pintar o eixo X
98             gl.glVertex2f(-1f, i);
99             gl.glVertex2f(135f, i);
100
101         }
102
103     }
104     gl.glEnd();
105
106     float x_desloc; //deslocamento no eixo X
107     int array_position; //percorre o arrayList
108
109     if
110     (this.statusView.getChckbx_showAngleXController().isSelected()) {
111         // Desenhar eixo X do smartphone controle
112         gl.glBegin(GL2.GL_LINE_STRIP);
113         gl.glColor3f(0f, 0f, 0f);
114         for (x_desloc = this.drawingStartPoint,
115 array_position = this.anglesSmartController.size() - 1;
116 array_position >=0; x_desloc -= this.x_interval, array_position--)
117             gl.glVertex2f(x_desloc,
118 this.anglesSmartController.get(array_position).getXangle());
119
120         gl.glEnd();
121
122     }
123
124     if
125     (this.statusView.getChckbx_showAngleYController().isSelected()) {
126         // Desenhar eixo Y do smartphone controle
127         gl.glBegin(GL2.GL_LINE_STRIP);
128         gl.glColor3f(0f, 1f, 0f);
129         for (x_desloc = this.drawingStartPoint,
130 array_position = this.anglesSmartController.size() - 1;
131 array_position >=0; x_desloc -= this.x_interval, array_position--)
132             gl.glVertex2f(x_desloc,
133 this.anglesSmartController.get(array_position).getYangle());
134
135         gl.glEnd();
136
137     }

```

```

138
139         if
140     (this.statusView.getChckbx_showAngleXCargo().isSelected()) {
141         // Desenhar eixo X do smartphone carga
142         gl.glBegin(GL2.GL_LINE_STRIP);
143         gl.glColor3f(0f, 0f, 1f);
144
145         for (x_desloc = this.drawingStartPoint,
146 array_position = this.anglesSmartCargo.size() - 1; array_position
147 >=0; x_desloc -= this.x_interval, array_position--)
148             gl.glVertex2f(x_desloc,
149 this.anglesSmartCargo.get(array_position).getXangle());
150
151         gl.glEnd();
152
153     }
154
155     if
156     (this.statusView.getChckbx_showAngleYCargo().isSelected()) {
157         // Desenhar eixo Y do smartphone carga
158         gl.glBegin(GL2.GL_LINE_STRIP);
159         gl.glColor3f(1f, 0f, 0f);
160         for (x_desloc = this.drawingStartPoint,
161 array_position = this.anglesSmartCargo.size() - 1; array_position
162 >=0; x_desloc -= this.x_interval, array_position--)
163             gl.glVertex2f(x_desloc,
164 this.anglesSmartCargo.get(array_position).getYangle());
165
166         gl.glEnd();
167
168     }
169
170
171     gl.glFlush();
172
173 }
174
175 @Override
176 public void dispose(GLAutoDrawable drawable) {
177     // System.out.println("dispose");
178 }

```

```

179
180     @Override
181     public void init(GLAutoDrawable drawable) {
182         // System.out.println("init");
183         final GL2 gl = drawable.getGL().getGL2();
184
185         gl.glShadeModel(GL2.GL_SMOOTH); // define o tipo de
186         sombra
187         gl.glClearColor(1f, 1f, 1f, 1f); // define a cor do fundo
188         gl.glClearDepth(1.0f); // nao sei
189         gl.glEnable(GL2.GL_DEPTH_TEST); // nao sei
190         gl.glDepthFunc(GL2.GL_LEQUAL); // nao sei
191         gl.glHint(GL2.GL_PERSPECTIVE_CORRECTION_HINT,
192         GL2.GL_NICEST);
193     }
194
195     @Override
196     public void reshape(GLAutoDrawable drawable, int x, int y, int
197     width,
198         int height) {
199
200         final GL2 gl = drawable.getGL().getGL2();
201
202         if (height <= 0)
203             height = 1;
204
205         final float h = (float) width / (float) height;
206
207         gl.glViewport(0, 0, width, height); // define a area q
208         sera usada para o
209
210         //
211         desenho
212         gl.glMatrixMode(GL2.GL_PROJECTION); // define o modo de
213         matriz utilizada
214         gl.glLoadIdentity();
215         glu.gluPerspective(45.0f, h, 1.0f, 200.0f); // define a
216         perspectiva, p1:
217
218         // angulo de visao; p2:
219
220         // aspecto; p3: visao mais

```

```

220
221         // perto; p4: visao mais
222
223         // longe
224         gl.glMatrixMode(GL2.GL_MODELVIEW);
225         gl.glLoadIdentity();
226     }
227
228 }
229

```

Classe *ControlClientSocket.java*

```

1  package control;
2  import java.io.IOException;
3  import java.io.ObjectInputStream;
4  import java.io.ObjectOutputStream;
5  import java.net.Socket;
6  import java.util.ArrayList;
7
8  import javax.swing.JOptionPane;
9
10 import view.InitialView;
11
12
13 public class ControlClientSocket {
14
15     private InitialView initialView;
16
17     public ControlClientSocket(InitialView initialView) {
18         super();
19         this.initialView = initialView;
20     }
21
22     public void sendToServer(ArrayList<Object> options) {
23
24         Socket socket = null;
25         ObjectOutputStream objectOutputStream = null;
26         ObjectInputStream objectInputStream = null;
27
28         try {

```

```

29
30         socket = new
31 Socket(this.initialView.getTextField_IP().getText(), 6789);
32
33         objectOutputStream = new
34 ObjectOutputStream(socket.getOutputStream());
35         objectInputStream = new
36 ObjectInputStream(socket.getInputStream());
37
38         System.out.println("Enviando: " + options);
39         objectOutputStream.writeObject(options);
40
41         @SuppressWarnings("unchecked")
42         ArrayList<Object> resposta = (ArrayList<Object>)
43 objectInputStream.readObject();
44
45         System.out.println("Resposta: " + resposta);
46
47     } catch (Exception e) {
48
49         JOptionPane.showMessageDialog(this.initialView, "Erro na
50 conexão!!\n"+e.getMessage(), "Erro!", JOptionPane.ERROR_MESSAGE);
51
52         e.printStackTrace();
53
54     } finally {
55
56         try {
57
58             if (objectInputStream != null)
59                 objectInputStream.close();
60
61             if (objectOutputStream != null) {
62                 objectOutputStream.flush();
63                 objectOutputStream.close();
64             }
65
66             if (socket != null)
67                 socket.close();
68
69         } catch (IOException e) {

```

```

70         JOptionPane.showMessageDialog(this.initialView, "Erro
71 ao fechar socket!!\n"+e.getMessage(), "Erro!",
72 JOptionPane.ERROR_MESSAGE);
73         e.printStackTrace();
74     }
75 }
76 }
77 }
78

```

Classe *ControlServerSocketCargo.java*

```

1  package control;
2  import java.io.ObjectInputStream;
3  import java.io.ObjectOutputStream;
4  import java.net.Socket;
5  import java.util.ArrayList;
6
7  import view.StatusView;
8
9  public class ControlServerSocketCargo implements Runnable{
10
11      private StatusView statusView;
12      private Thread thread;
13
14      public ControlServerSocketCargo(StatusView statusView) {
15          super();
16          this.statusView = statusView;
17          this.thread = new Thread(this);
18          this.thread.start();
19
20      }
21
22      @Override
23      public void run() {
24
25          try {
26              java.net.ServerSocket serverSocket;
27              Socket socket;
28
29              while (this.statusView.isVisible()) {

```

```

30         serverSocket = new
31 java.net.ServerSocket(9999);
32
33         socket = serverSocket.accept();
34
35         ObjectOutputStream outputStream = new
36 ObjectOutputStream(socket.getOutputStream());
37         ObjectInputStream inputStream = new
38 ObjectInputStream(socket.getInputStream());
39
40         @SuppressWarnings("unchecked")
41         ArrayList<Float> arrayListAngles =
42 (ArrayList<Float>) inputStream.readObject();
43         this.statusView.angles_smart_cargo =
44 arrayListAngles;
45
46         outputStream.flush();
47         inputStream.close();
48         outputStream.close();
49         socket.close();
50         serverSocket.close();
51
52     }
53
54     this.thread.interrupt();
55
56     } catch (Exception e) {
57         e.printStackTrace();
58
59     }
60
61     }
62
63 }
64

```

Classe *ControlServerSocketController.java*

```

1 package control;
2 import java.io.ObjectInputStream;
3 import java.io.ObjectOutputStream;

```

```

4  import java.net.Socket;
5  import java.util.ArrayList;
6
7  import view.StatusView;
8
9  public class ControlServerSocketController implements Runnable{
10
11      private StatusView statusView;
12      private Thread thread;
13
14      public ControlServerSocketController(StatusView statusView) {
15          super();
16          this.statusView = statusView;
17          this.thread = new Thread(this);
18          this.thread.start();
19
20      }
21
22      @Override
23      public void run() {
24
25          try {
26              java.net.ServerSocket serverSocket;
27              Socket socket;
28
29              while (this.statusView.isVisible()) {
30                  serverSocket = new
31 java.net.ServerSocket(8888);
32
33                  socket = serverSocket.accept();
34
35                  ObjectOutputStream objectOutputStream = new
36 ObjectOutputStream(socket.getOutputStream());
37                  ObjectInputStream objectInputStream = new
38 ObjectInputStream(socket.getInputStream());
39
40                  @SuppressWarnings("unchecked")
41                  ArrayList<Float> arrayListAngles =
42 (ArrayList<Float>) objectInputStream.readObject();
43                  this.statusView.angles_smart_controller =
44 arrayListAngles;

```



```

45
46         objectInputStream.close();
47
48         objectOutputStream.flush();
49         objectOutputStream.close();
50
51         socket.close();
52         serverSocket.close();
53
54     }
55
56     this.thread.interrupt();
57
58     } catch (Exception e) {
59         e.printStackTrace();
60
61     }
62
63 }
64
65 }
66

```

Classe *ControlDataSave.java*

```

1  package control.controlManagerSensors;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10 public class ControlDataSave implements ActionListener{
11
12     private InitialView initialView;
13
14     public ControlDataSave(InitialView initialView) {
15         super();
16         this.initialView = initialView;

```

```

17     }
18
19     @Override
20     public void actionPerformed(ActionEvent e) {
21
22         ArrayList<Object> option = new ArrayList<Object>();
23
24         if (this.initialView.getChckBoxDataSave().isSelected())
25             option.add(3);
26
27         else
28             option.add(4);
29
30         new
31         ControlClientSocket(this.initialView).sendToServer(option);
32
33     }
34
35 }
36

```

Classe *ControlStartCaptureSensors.java*

```

1  package control.controlManagerSensors;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10 public class ControlStartCaptureSensors implements ActionListener{
11
12     private InitialView initialView;
13
14     public ControlStartCaptureSensors(InitialView initialView) {
15         super();
16         this.initialView = initialView;
17     }
18

```

```

19      @Override
20      public void actionPerformed(ActionEvent e) {
21          ArrayList<Object> option = new ArrayList<Object>();
22          option.add(1);
23
24          new
25      ControlClientSocket(this.initialView).sendToServer(option);
26
27
28      }
29
30  }
31

```

Classe *ControlStartSendStatus.java*

```

1  package control.controlManagerSensors;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9  import view.StatusView;
10
11  public class ControlStartSendStatus implements ActionListener{
12
13      private InitialView initialView;
14
15      public ControlStartSendStatus(InitialView initialView) {
16          super();
17          this.initialView = initialView;
18      }
19
20      @Override
21      public void actionPerformed(ActionEvent e) {
22          ArrayList<Object> option = new ArrayList<Object>();
23          option.add(5);
24          option.add(this.initialView.getLbl_IP_local().getText());
25

```

```

26         new
27 ControlClientSocket (this.initialView).sendToServer(option);
28         this.initialView.setStatusView(new StatusView());
29
30     }
31
32 }
33
34

```

Classe *ControlStopCaptureSensors.java*

```

1  package control.controlManagerSensors;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10 public class ControlStopCaptureSensors implements ActionListener{
11
12     private InitialView initialView;
13
14     public ControlStopCaptureSensors(InitialView initialView) {
15         super();
16         this.initialView = initialView;
17     }
18
19     @Override
20     public void actionPerformed(ActionEvent e) {
21         ArrayList<Object> option = new ArrayList<Object>();
22         option.add(2);
23
24         new
25 ControlClientSocket (this.initialView).sendToServer(option);
26
27     }
28
29 }

```

30

Classe *ControlStopSendStatus.java*

```

1  package control.controlManagerSensors;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10 public class ControlStopSendStatus implements ActionListener{
11
12     private InitialView initialView;
13
14     public ControlStopSendStatus(InitialView initialView) {
15         super();
16         this.initialView = initialView;
17     }
18
19     @Override
20     public void actionPerformed(ActionEvent e) {
21
22         this.initialView.getStatusView().setVisible(false);
23         this.initialView.getStatusView().dispose();
24
25         ArrayList<Object> option = new ArrayList<Object>();
26         option.add(6);
27
28         new
29 ControlClientSocket(this.initialView).sendToServer(option);
30
31     }
32
33 }
34

```

Classe *ControlDirectionBack.java*

```

1  package control.directionControl;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10 public class ControlDirectionBack implements ActionListener{
11
12     private InitialView initialView;
13
14     public ControlDirectionBack(InitialView initialView) {
15         super();
16         this.initialView = initialView;
17     }
18
19     @Override
20     public void actionPerformed(ActionEvent e) {
21         ArrayList<Object> option = new ArrayList<Object>();
22         option.add(9);
23
24         new
25         ControlClientSocket(this.initialView).sendToServer(option);
26
27     }
28
29 }
30

```

Classe *ControlDirectionFront.java*

```

1  package control.directionControl;
2
3
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.util.ArrayList;
7
8  import control.ControlClientSocket;

```

```

9  import view.InitialView;
10
11  public class ControlDirectionFront implements ActionListener{
12
13      private InitialView initialView;
14
15      public ControlDirectionFront(InitialView initialView) {
16          super();
17          this.initialView = initialView;
18      }
19
20      @Override
21      public void actionPerformed(ActionEvent e) {
22          ArrayList<Object> option = new ArrayList<Object>();
23          option.add(7);
24
25          new
26  ControlClientSocket(this.initialView).sendToServer(option);
27
28      }
29
30  }
31

```

Classe *ControlDirectionKeyListener.java*

```

1  package control.directionControl;
2
3  import java.awt.event.KeyEvent;
4  import java.awt.event.KeyListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10  public class ControlDirectionKeyListener implements KeyListener{
11
12      private InitialView initialView;
13
14      public ControlDirectionKeyListener(InitialView initialView) {
15          super();

```

```

16         this.initialView = initialView;
17     }
18
19     @Override
20     public void keyTyped(KeyEvent e) {}
21
22     @Override
23     public void keyPressed(KeyEvent e) {
24         // TODO Auto-generated method stub
25
26         if (e.getKeyCode() == KeyEvent.VK_UP) {
27             // System.out.println("up");
28             ArrayList<Object> option = new ArrayList<Object>();
29             option.add(7);
30             new
31 ControlClientSocket(this.initialView).sendToServer(option);
32
33         } else if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
34             // System.out.println("left");
35             ArrayList<Object> option = new ArrayList<Object>();
36             option.add(8);
37             new
38 ControlClientSocket(this.initialView).sendToServer(option);
39
40         } else if (e.getKeyCode() == KeyEvent.VK_DOWN) {
41             // System.out.println("down");
42             ArrayList<Object> option = new ArrayList<Object>();
43             option.add(9);
44             new
45 ControlClientSocket(this.initialView).sendToServer(option);
46
47         } else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
48             // System.out.println("right");
49             ArrayList<Object> option = new ArrayList<Object>();
50             option.add(10);
51             new
52 ControlClientSocket(this.initialView).sendToServer(option);
53
54         } else if (e.getKeyCode() == KeyEvent.VK_ENTER) {
55             // System.out.println("enter");
56             ArrayList<Object> option = new ArrayList<Object>();

```



```

57         option.add(11);
58         new
59         ControlClientSocket(this.initialView).sendToServer(option);
60
61     }
62
63 }
64
65 @Override
66 public void keyReleased(KeyEvent e) {}
67
68 }
69

```

Classe *ControlDirectionLeft.java*

```

1  package control.directionControl;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10 public class ControlDirectionLeft implements ActionListener{
11
12     private InitialView initialView;
13
14     public ControlDirectionLeft(InitialView initialView) {
15         super();
16         this.initialView = initialView;
17     }
18
19     @Override
20     public void actionPerformed(ActionEvent e) {
21         ArrayList<Object> option = new ArrayList<Object>();
22         option.add(10);
23         new
24         ControlClientSocket(this.initialView).sendToServer(option);
25

```

26	}
27	
28	}
29	

Classe *ControlDirectionRight.java*

1	package control.directionControl;
2	
3	import java.awt.event.ActionEvent;
4	import java.awt.event.ActionListener;
5	import java.util.ArrayList;
6	
7	import control.ControlClientSocket;
8	import view.InitialView;
9	
10	public class ControlDirectionRight implements ActionListener{
11	
12	private InitialView initialView;
13	
14	public ControlDirectionRight(InitialView initialView) {
15	super ();
16	this .initialView = initialView;
17	}
18	
19	@Override
20	public void actionPerformed(ActionEvent e) {
21	ArrayList<Object> option = new ArrayList<Object>();
22	option.add(8);
23	new
24	ControlClientSocket(this .initialView).sendToServer(option);
25	
26	}
27	
28	}
29	

Classe *ControlDirectionStop.java*

1	package control.directionControl;
2	
3	import java.awt.event.ActionEvent;

```

4  import java.awt.event.ActionListener;
5  import java.util.ArrayList;
6
7  import control.ControlClientSocket;
8  import view.InitialView;
9
10 public class ControlDirectionStop implements ActionListener{
11
12     private InitialView initialView;
13
14     public ControlDirectionStop(InitialView initialView) {
15         super();
16         this.initialView = initialView;
17     }
18
19     @Override
20     public void actionPerformed(ActionEvent e) {
21         ArrayList<Object> option = new ArrayList<Object>();
22         option.add(11);
23         new
24         ControlClientSocket(this.initialView).sendToServer(option);
25
26     }
27
28 }
29

```

Classe *AnglesVO.java*

```

1  package model;
2
3  /**
4   * Created by leonardo on 12/06/15.
5   */
6  public class AnglesVO {
7
8     private float x_angle, y_angle;
9
10     public AnglesVO(float x_angle, float y_angle) {
11         this.x_angle = x_angle;
12         this.y_angle = y_angle;

```

```
13     }  
14  
15     public float getXangle() {  
16         return x_angle;  
17     }  
18  
19     public float getYangle() {  
20         return y_angle;  
21     }  
22  
23 }  
24
```

APÊNDICE D – Código fonte do *firmware* do Arduino

Firmware do Arduino

```

1  #include <Servo.h>
2
3  Servo servoMotorX, servoMotorY;
4  String value; //variável que receberá os valores da porta serial
5  char character;
6
7  void setup() {
8      Serial.begin(9600);
9
10     servoMotorX.attach(8);
11     servoMotorY.attach(9);
12
13     servoMotorX.write(90);
14     servoMotorY.write(90);
15
16     String value = "";
17
18 }
19
20 void loop() {
21     //verifica se existe algum dado na porta serial
22     if (Serial.available()) {
23         character = Serial.read();
24
25         if (character == 'a') {
26             servoMotorX.write( value.toInt() );
27             value = "";
28
29         } else if (character == 'b') {
30             servoMotorY.write( value.toInt() );
31             value = "";
32
33         } else if (character == 'c') {
34             //         executa do código de controle de locomoção

```

```
35
36     switch( value.toInt() ){
37         case 7:
38             //         inserir código referente à locomoção para frente
39
40             break;
41
42         case 8:
43             //         inserir código referente à locomoção para direita
44
45             break;
46
47         case 9:
48             //         inserir código referente à locomoção para trás
49
50             break;
51
52         case 10:
53             //         inserir código referente à locomoção para esquerda
54
55             break;
56
57         case 11:
58             //         inserir código de parar locomoção
59
60             break;
61     }
62
63     value = "";
64
65     } else {
66         value.concat(character);
67
68     }
69 }
70 }
71
```