Universidade Federal de São Carlos - UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Engenharia Mecânica – DEMec
Curso de Engenharia Mecânica

Trabalho de Conclusão de Curso

IMPLEMENTAÇÃO DE UMA ARQUITETURA EXPERIMENTAL DE AVIÔNICA MODULAR INTEGRADA DISTRIBUíDA

Gustavo Teruo Bernardino Tamanaka

Orientador:
Prof. Dr. Rafael Vidal Aroca

São Carlos - SP- 2017

Universidade Federal de São Carlos - UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Engenharia Mecânica – DEMec
Curso de Engenharia Mecânica

Trabalho de Conclusão de Curso

IMPLEMENTAÇÃO DE UMA ARQUITETURA EXPERIMENTAL DE AVIÔNICA MODULAR INTEGRADA DISTRIBUíDA

Trabalho de Conclusão de Curso apresentado à Universidade Federal de São Carlos, como parte dos requisitos para obtenção do título de bacharel em Engenharia Mecânica.

Prof. Dr. Rafael Vidal Aroca

São Carlos - SP- 2017

Dedico este trabalho a minha família que sempre me apoiou em todos os meus momentos e aqueles que vem depois de mim, deixem que o conhecimento transforme a suas vidas.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a minha família, meus pais Antônia Vera Lucia Bernardino Tamanaka e Toshimitsi Tamanaka, minha irmã Natália Mayumi Bernardino Tamanaka e minha tia Maria Bonfim de Souza, sem o apoio deles estar na graduação nunca seria possível. Agradeço a Universiade Federal de São Carlos e ao Departamento de Engenharia Mecânica pelos ano de ensino gratuido de qualidade e pelas políticas de permanência estundantil, sem elas teria continuado na graduação. Agradeço ao professor Rafael Vidal Aroca, orietador neste trabalho, pela infindáveis ajudas, tanto nesse trabalho quando em outro projetos, seus anos de amizade e os constantes ensinamentos foram muitos importantes para as minha decisões profissionais.

Finalmente, não menos importante, gostaria de agradecer a todos os meus amigos que zelo com carinho, aos fiéis e de longa data amigos do CTI, aos amigos da MEC011 que estiveram em muitas noites viradas fazendo projeto, aos amigos do intercâmbio companheiros na saudade e aos meu amigos da "Cobertura"que alegram todos os meu dias com agradavéis conversas no corredor.

“Conhecereis a verdade, e a verdade vos libertará."
(João 8,32)

RESUMO

Este trabalho discute o aspectos teóricos e experimentais para a implementação de aviônica Modular Integrada Distribuída. A tecnologia aviônica distribuída é, ainda, um assunto recente, mas de grande importância para a indústria aeronáutica, portanto este trabalho tenta acrescentar informação e promover a discussão sobre o tema. Para tanto, este trabalho buscou alinhar teoria e prática ao desenvolver uma bancada experimental de forma simplificada. Para o desenvolvimento da bancada foram utilizados circuitos de prototipagem rápida a fim de simular os equivalentes comerciais. O desenvolvimento da bancada ocorreu como esperado, resultando em um sistema capaz de receber adequadamente a leitura a captura de sensores e a movimentação de atuadores. Desta forma conclui-se que os objetivos propostos foram alcançados.

Palavras-chave: DIMA. Aviônica. RTOS.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

IMA	*Integrated Modular Avionics*
RTOS	*Real Time Operation System*
DIMA	*Distributed Integrated Modular Avionics*
ARINC	*Aeronautical Radio, Incorporated*
AFDX	*Avionics Full Duplex Ethernet*
CPM	*Computer Power Management*
IMU	*Inertail Measurement Unit*
UDP	*User Datagram Protocol*
TCP	*Transfer Control Protocol*
API	*Application Programming Interface*
RFC	*Request for Comments*
FAA	*Federal Aviation Administration*
BB	*Beagle Bone*

SUMÁRIO

1 . INTRODUÇÃO

Os avanços nas tecnologias criadas pela humanidade sempre visaram aumentar a segurança, comodidade e diminuir os custos. A aviação segue pelos mesmos caminhos, onde o aumento da automação e a busca constante por avanços tecnológicos tem mudado os paradigmas dos equipamentos de aviônica. O conceito de aviônica é a concatenação de aviação e eletrônica, ou seja, a eletrônica especificamente presente em aeronaves, apoiada em sensores, atuadores e controladores necessários para realizar as mais diversas atividades de um avião que outrora foram feitos por equipamentos estritamente mecânicos. Dessa forma, hoje em dia, a maior parte das aeronaves tanto comerciais quanto militares fazem uso de diversos sistemas de aviônica.

Dois conceitos de aviônica importantes e que hoje existem em várias aeronaves comerciais e militares são o *fly-by-wire* e o *glass cockpit*. Ambos conceitos são exemplos de melhorias para atender requisitos de segurança e disponibilidades de sistemas de uma aeronave que levaram à diminuição de peso, uma característica sempre almejada durante a concepção de um projeto na área da aviação.

Fly-by-wire é a filosofia de projeto de controle de voo, onde toda a parte mecânica que antes era responsável pelo controle é substituída por uma rede de comunicação redundante e todos os comandos são enviados via sinais digitais distribuídos para cada um dos atuadores de um sistema. Esta tecnologia vem sendo explorada desde 1970, sendo o Concorde a primeira aeronave comercial a possuir tal sistema. A AIRBUS e a Boeing consagram o uso desse tipo de controle quando implementaram suas próprias versões no A320 e no Boeing 777 (LEITE, 2014).

Glass cockpit é composto por um ou vários mostradores digitais que substituem os antigos mostradores tradicionais analógicos (referidos como *Steam cockpit*). O *Glass cockpit* visa facilitar a navegação e operação da aeronave ao mostrar apenas as informações necessárias para o piloto, diminuindo a quantidade de informação que é mostrada nas cabines *Steam cockpit*.

Apesar da clara vantagem da substituição de todos os equipamentos mecânicos por equipamentos eletrônicos, o peso das aeronaves ainda continuaram a ser um requisito de projeto, visto que apesar da diminuição de peso para um sistema específico, o peso aumentou devido ao aumento de sistemas nas aeronaves modernas, além do fato dos comprimentos dos cabos poderem chegar a quilômetros (ANNIGHÖFER; THIELECKE, 2012).

Entre as primeiras arquiteturas de aviônica criadas, uma das principais arquiteturas é a do tipo Federada. Na arquitetura federada, cada sistema de entrada e saída (I/O) é um sistema independente, possuindo recursos computacionais e de coleta de dados independentes, portanto um sistema contido em si próprio. Tipicamente as unidades

que compõem uma rede federada são modelo *Line Replaceable Units* (LRU) ou *Line Replaceable Module* (LRM), ou seja, para substituir uma unidade é necessário substituir todo um sistema, não havendo modularidade entre as entidades (WATKINS; WALTER, 2007). Inevitavelmente, o desejo por otimizar o peso de um projeto levou ao surgimento das arquiteturas de aviônica modular integrada (IMA).

O conceito de *Integrated Modular Avionics* (IMA) surgiu em meados dos anos 1980, sendo primeiro introduzido no Boeing 777, tornando-se rapidamente difundido entre aplicações tanto militares quando comerciais, entre elas o Lockheed C130 AMP, o Airbus A380 e o Boeing 787 (MOIGNE; PASQUIER; CALVEZ, 2004). Um dos principais impulsionadores dessa tecnologia foi o *Airlines Electronic Engineering Commite* (AEEC).

A natureza modular da arquitetura IMA permite a diminuição dos custos de aquisição, devido aos componentes que podem se produzidos em escalas muito maiores, que levam a custo de fabricação muito menores. A intercambiabilidade dos módulos também permite alterações nos sistemas da aeronave sem que uma grande parte seja comprometida. Esse dois fatores, por consequência, levam à aeronaves com custo menores (PRISAZNUK, 1992).

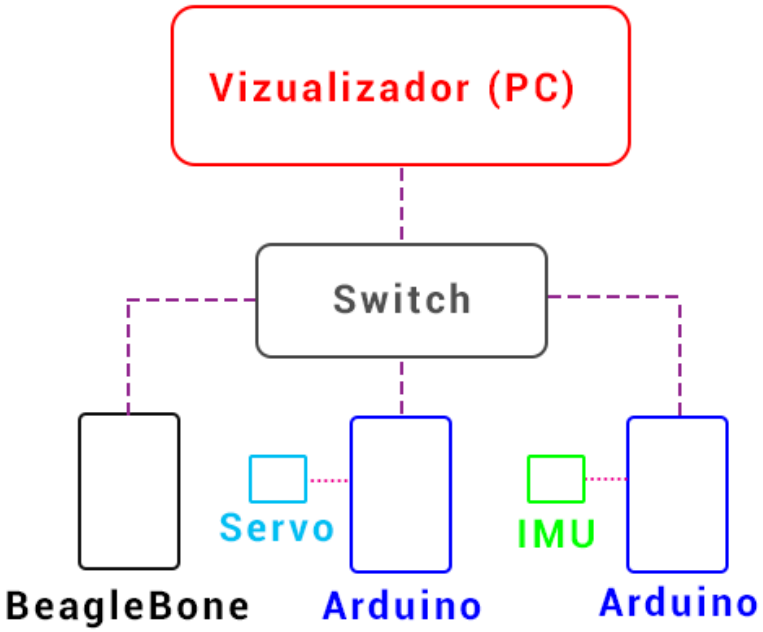
O passo seguinte para a arquitetura IMA, ainda em desenvolvimento, segunda geração IMA (IMA2G), comumente referidas como redes IMA distribuídas (*Distributed Integrated Modular Avionics* – DIMA). As arquiteturas DIMA se apoiam no paradigma de projeto de novos dispositivos ligados em rede e distribuídos por toda aeronave, obtendo melhores latências e menos cabeamento necessário para interligar diferente sistemas, além de maior flexibilidade de projeto.

Existem algumas normas que especificam as diretrizes para concepção e implementação desse tipo de tecnologia em aeronaves, entre elas pode-se citar a ARINC 653 e a DO-178 A prática e teoria sobre os desenvolvimento utilizando destas normas tornaram alguns conceitos mandatórios como por exemplo os sistemas operacionais de tempo real (RTOS), ferramentas especificadas nas normas para garantir robustez e confiabilidade aos sistemas.

Considerando a importância das arquiteturas IMA e DIMA, e a falta de grupos especializados nesta área no Brasil, o presente trabalho busca contribuir no estudo e desenvolvimento destas tecnologias no Brasil. Dessa forma, apresenta-se o conceito, implementação, construção e validação experimental de uma rede DIMA completa, incluindo sensores, atuadores, sistemas de controle e sistema visualização de dados de sensores presentes na rede. Para tanto, o trabalho faz uso de um sistema operacional de tempo que real que implementa as especificações exigidas pela norma ARINC 653. O trabalho também se apoia no uso de placas micro-controladas para simular alguns dos sistemas de uma aeronave: atuadores e sensores.

A Figura 1 demonstra de forma esquemática a topografia da rede que foi desenvolvida nesse trabalho. Na arquitetura proposta, e implementada, um *Switch* Ethernet convencional faz papel de *Switch* AFDX (*Avionics Full Duplex Switch*), conectando uma placa BeagleBone, que funciona com um RTOS compatível com ARINC 653, e faz o papel de módulo de Computação e Processamento da (CPM) rede DIMA. Além disso, uma ou mais placas Arduino, todas com conexão Ethernet, se conectam à rede Ethernet através do *Switch*, disponibilizando sensores e/ou atuadores como serviços em rede. No caso deste trabalho, um Arduino disponibiliza dados de *Roll*, *Pitch* e *Yaw* através de uma unidade de medidas inerciais (IMU), enquanto outro Arduino permite o controle do ângulo do eixo de um Servo motor ligado ao referido Arduino. Finalmente, um PC também é conectado à rede, sendo este capaz de visualizar dados de forma gráfica e com atualização automática sobre leituras dos sensores em rede.

Figura 1 – Diagrama de blocos da rede proposta



Fonte: do Autor

Os resultados obtidos neste trabalho são interessantes para o entendimento da aplicabilidade do desenvolvimento de DIMA através da abordagem por prototipação utilizando componentes de baixo custo, com funções similares aos reais de uma aeronave. O conceito demonstrado pode ser replicado em aeronaves reais usando sistemas e equipamentos certificados, o que poderá prover todas a vantagens de uma rede DIMA.

Para contextualizar o assunto e embasar o trabalho aqui desenvolvido, este documento apresenta uma breve explicação dos conceitos e história das redes de aviônica

modular, passando pelo entendimento de sistemas de tempo real, pelas diretrizes para esse tipo de sistema segundo a ARINC 653 e finalmente mostrando a aplicação desenvolvida e seus resultados.

1.1 Objetivos

O principal objetivo deste trabalho, é a construção de uma bancada experimental para aviônica usando arquitetura DIMA, permitindo realizar estudos, testes e experimentos neste ambiente.

1.2 Objetivos Específicos

Para alcançar o objetivo principal, as seguinte metas precisam ser alcançadas. São elas:

- Estudo de arquiteturas de aviônica;
- Projeto e construção de uma bancada de testes para redes DIMA;
- Estudo de Sistemas Operacionais de Tempo real (RTOS);
- Estudo de comunicação em rede AFDX, Ethernet e dos protocolos IP, TCP e UDP;
- Configuração e instalação de RTOS para a aplicação;
- Desenvolvimento de nós (RDCs) com sensores e atuadores ligados em rede Ethernet;
- Desenvolvimento de nó CPM para gerenciamento da rede;
- Implementação de software e integração dos nós;
- Testes, documentação e validação da proposta.

1.3 Organização

O presente trabalho é organizado respeitando a ordem lógica e cronológica da execução de cada atividade necessária para se alcançar o objetivo final, tentando ao máximo tornar claro os passos necessários para que os mesmo tipos de resultado possam ser realizado por outros que não o autor, para tanto várias explicações com respeito a utilitários utilizados estão presentes. Este trabalho contém as seguintes partes: Introdução (Capítulo 1), Revisão Bibliográfica (Capítulo 2), Fundamentação Teórica (Capítulo 3),Materiais e Métodos (Capítulo 4), Desenvolvimento (Capítulo 5), Resultados (Capítulo 6) e Conclusões (Capítulo 7).

2 . REVISÃO BIBLIOGRÁFICA

Os sistemas de aviônica e de aviônica distribuída têm sido temas de constantes pesquisas e desenvolvimentos, tanto na área acadêmica quanto na indústria. Neste capítulo, alguns trabalhos relacionados com o tema do presente texto são apresentados e discutidos.

Sabe-se que na maior parte das arquiteturas de aviônica moderna, há diversas interconexões para troca de dados entre dispositivos. Normalmente utiliza-se tecnologias de comunicação em rede projetadas para sistemas de aviônica, ou tecnologias já bem estabelecidas que são adaptadas e melhoradas para aplicações de aviônica. Um exemplo de padrão que vem se tornando a base de diversos sistemas, são as redes baseadas no padrão Ethernet (TANENBAUM, 2002). O próprio padrão *Avionics Full Duplex Switched Ethernet* (AFDX) é baseado em Ethernet.

Dessa forma, Warden (2017) discute como a utilização de redes Ethernet para sistemas determinísticos se tornou uma opção para aeronaves, especialmete pelo advento da ARINC 664 parte 7 e dos esforços de órgãos de outras áreas que trabalham com equivalentes redes determinísticas, como o IEEE e a SAE. Entretanto, existem diversos desafios a serem superados com respeito à sua utilização. Nota-se ainda que um padrão cada vez mais presente é o AFDX, que é uma implementação da norma ARINC 664 - parte 7.

Segundo o Warden (2017), a maior inovação por parte da ARINC 664 - parte 7 foi a adoção de *"Virtual Links"*. Na operação dos *"Virtual Links"* todos os caminhos entre um *"source"* e múltiplos destinos são identificados, e então estes caminhos recebem restrições de banda em cada caminho de forma a reduzir a probabilidade de se perder pacotes, conferindo maior confiabilidade à rede (WARDEN, 2017).

Entretanto, Warden (2017) ainda aponta que a falta de um controle do fluxo da rede ainda é um problema, pois se um nós envia mais pacotes do que outro nó pode receber, os pacotes serão perdidos e nenhum dos nós saberá que perdeu pacotes. Outro problema apontado é a falta de equipamentos adequados para realizar medições determinísticas adequadas, sendo muito mais comum a utilização de soluções desenvolvidas para cada rede em seu uso particular.

Já partindo para a discussão de aviônica integrada distribuída, Šegvić et al. (2016) descrevem um sistema de controle de voo totalmente distribuído, numa abordagem diferente dos atuais sistemas de controle de voo empregados em redes de arquitetura federada e IMA. A motivação dos autores se baseia na ideia de retirar a certificação da rede IMA e transferir para subsistemas separados, além de promover a redução de custos, transferindo as funções de controle de voo diretamente a nível do atuador e não mais no nível da aplicação.

Os autores ainda apontam que a transferência das funções de controle de voo a nível

de atuadores aumentaria o número de dispositivos embarcados, entretanto seria possível substituir satisfatoriamente os computadores de alto custo e potência já empregados que são necessários para controlar as aplicações. Adicionalmente, todos os pequenos módulos embarcados seriam do mesmo tipo, mais fáceis de serem manufaturados e substituídos, levando a custo mais reduzido, visto que atualmente os sistema de aviônica custam em torno de 40% os custo de uma aeronave comercial (ŠEGVIĆ; NIKOLIĆ; IVANJKO, 2016).

Considerando a maturidade e amplo uso da abordagem IMA, tipicamente se utiliza a norma DO-297 para guiar o desenvolvimento e certificação de sistemas com arquitetura IMA. A DO-297 subdivide um sistema em 4 níveis de certificação:

- *Module Acceptable*,
- *Application Acceptable*,
- *System-Level Acceptable* e
- *Aircraft-Level Acceptable*.

Module Acceptable corresponde à validação dos componentes chamado de módulos, podendo ser tanto Hardware quanto Software, ou combinação de ambos;

Application Acceptable é a validação referente à função de um módulo;

System-Level Acceptable avalia o sistema que fornece poder computacional para as aplicações, bem como os serviços suportes para essas aplicações;

Aircraft-Level Acceptable valida a aeronave como um todo.

Contudo, a norma DO-297 não foi estabelecida para sistemas de aviônica distribuída. essa forma, Wolfig e Jakovljevic (2007) mostram em seu trabalho, pontos guias com relação ao DO-297 para a certificação de sistemas integrados, tais guias permitem o desenvolvimento e certificações de funções de uma aeronave independentemente como a certificação da arquitetura de integração para as funções.

Os autores argumentam que a utilização de certificação independente para cada módulo de uma rede modular permite poupar recursos e esforços de validação. Isto seria válido caso ocorra a reutilização de um componente já certificado em outro projeto, portanto o mesmo não precisa ser certificado, ou caso uma arquitetura já em uso precisar apenas de uma alteração, apenas o novo módulo precisaria ser certificado.

Zheng Li, Qiao Li e Huagang Xiong (2012) apresentam uma proposta para a inserção do conceito de *Cloud Computaning* na aviação, objetivando aumentar o nível de virtualização dos equipamentos, possibilitando a utilização de recurso entre plataformas, sendo essas plataformas os outras aeronaves em voo, recursos em terra, entre outros.

Os autores apontam que o conceito de *Cloud Computaning* se baseia em três camadas: Infraestrutura como serviço (IaaS), Plataforma como serviço (PaaS) e Software como serviço (SaaS). No conceito da internet comum, a IaaS é responsável por prover armazenamento e poder computacional como serviço através da rede, a PaaS provém execução de ambientes de linguagem de programação e serviço como servidores e banco de dados, finalmente, o SaaS oferece aplicações como serviço, portanto o usuário não precisa possuir a aplicação na sua máquina local.

Com relação ao *Cloud Computaning* aplicado à aviação, o autor introduz o conceito de *Avionic Cloud*, que é já explorado por outro autores da área, entretanto tais pesquisas utilizam um ambiente de computação em nuvem que é apenas social, apenas simples trocas de informações, e não leva em consideração cenários de de interação baseadas em tempo real. Na proposta de Zheng Li, Qiao Li e Huagang Xiong (2012) o sistema seria baseado em três camadas, similares as presentes na computação em nuvem convencional, que seriam camada de Infraestrutura, camada de função e camada de aplicação.

Na camada de infraestrutura estariam virtualizados não somente os elementos de computação, mas também os elementos da aviônica, como seus sensores, componentes de navegação e comunicação, componentes de medidas meteorológicas, radares. Na camada de função os serviços virtualizados alcançariam funções específicas, como navegação precisa, informações em tempo real sobre regiões para evitar colisões, sobre obstáculos, entre outros, permitindo que a cada acesso a esse serviço pela aeronave melhore o seu desempenho em voo.

Finalmente, a camada de aplicação permitiria o compartilhamento de recurso completos de aviônica, como medidas globais meteorológicas, planejamento de rotas, voo de baixa altitude e altas velocidade, até mesmo sistema de entretenimento. Tudo isso permitira que aeronaves *low-end* tivessem acesso a recursos mais avançados, sem a necessidade de possuir o tipo de equipamento equivalente.

Toda a complexidade dos sistemas IMA e suas próximas gerações demandam ferramentas de análise e validação de forma confiável e produtiva. Assim, Efkemann e Peleska (2011), relatam sobre o desenvolvimento da projeto SCARLETT, que visa o desenvolvimento de ferramentas para testes automatizados de redes DIMA. SCARLETT é o acrônimo para *SCAlable ReconfigurabLe elEctronics plaTforms and Tools*, um projeto conjunto de diversas entidade Europeias visando melhorar o desempenho de redes ao aplicar o conceito de *Distributed Modular Electronics* (DME).

Os ensaios para análises de redes DIMA são em diferentes níveis:

Bare Module Test;

Configured Module Tests; e

Functional Test.

O *Bare Module Test* analisa o sistema operacional a nível da API, testando o comportamento da API, bem como sua robustez quando a tentativas de violação de regras. O *Configured Module Tests* verifica configurações de módulos com relação a uma aeronave real, como por exemplo se o I/O funciona como especificado. Finalmente, o *Functional Test* testa todo o conjunto, ou seja a API e seus módulos de configuração (EFKEMANN; PELESKAR, 2011).

Estes autores apresentam testes automatizados baseados em linguagem programada, tendo as ferramentas *IMA Test Modelling Language variant* B e C para realizar, respectivamente, *Bare Module Test* e *Configured Module Tests* de forma autônoma e com resultados adequados. Os autores afirmam que o uso destas ferramentas diminui de 40% a 80% os esforços necessários para realizar estes testes se os mesmo fossem feitos utilizando o método convencional (EFKEMANN; PELESKAR, 2011).

Este capítulo apresentou uma breve revisão de trabalhos relacionados na área de aviônica distribuída integrada, demonstrando o interesse, demanda e esforços na área. No próximo capítulo são diascutidos os aspectos teóricos que embasam o desenvolvimento e implementação da arquitetura proposta neste trabalho.

3 . FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo aprofundar a discussão sobre os conceitos mais relevantes apresentados no Capítulo 1. A importância das arquiteturas IMA e DIMA também é discutida. Além disso, é explicada a importância dos RTOS no escopo do desenvolvimento da rede ARINC 653, bem como os principais pontos da ARINC 653.

3.1 Sistemas IMA

Como citado no Capítulo 1, os esforços para o desenvolvimento da arquitetura IMA datam de meados de 1980, com o objetivo de viabilizar a fabricação de equipamentos de aviônica menores, mais leves e com melhor custo. O rápido avanço dos *softwares* ajudou a impulsionar a capacidade de gerar este tipo de equipamento. Os conceitos da IMA se apoiam na capacidade advinda dos softwares de usar altos níveis de processamento em um módulo e que pode alocar processamento para aplicações independentes. Um dos grandes impulsionadores do padrão IMA é o *Airlines Electronic Engineering Committee* (AEEC) (PRISAZNUK, 1992).

O AEEC é um organização de padronização internacional fundada em 1949, responsável pelo desenvolvimento padrões e normas para equipamentos eletrônicos voltados para o transporte aéreo (aviônica). O principal objetivo da AEEC é promover a diminuição do custo do equipamento de eletrônica voltada para aeronáutica através da maior competição entre os fabricantes desse tipo de equipamento e melhores padronizações de produtos (PRISAZNUK, 1992).

Os benefícios da abordagem IMA se estendem aos 4 grandes grupos de interesse ao desenvolvimento das técnicas da industria aeronáutica. As empresas de linhas aéreas se beneficiam do aumento da performance no nível operacional, aumento da carga de voo, além de melhoras nas rotinas de manutenção. As fabricantes de aviões se beneficiam das diminuições dos custo de fabricação, mais desempenho e menores custos de certificação. As fabricantes de equipamentos eletrônicos para aviação têm como vantagem melhores oportunidades de atingir maiores fatias do mercado, maior tempo de mercado para um determinado componente e a possibilidade de flexibilizar suas fabricação para atender pedidos específicos de clientes. Finalmente, as autoridade certificadoras podem criar melhores padronizações para qualificar componentes independentemente da aeronave (PRISAZNUK, 1992).

3.2 Sistemas DIMA

A primeira geração de sistemas to tipo de IMA são uma tecnologias bastante consolidas e amplamente aplicadas na indústria aeronáutica. Porém, apesar de todos os

benefícios proporcionados por esse tipo de implementação de sistema, a mesma ainda não é capaz de utilizar de toda capacidade de compartilhamento de recurso entre os módulos, o que pode levar a um sobrepeso, e por muitas vezes a um excesso de cabos. A fim de possibilitar melhor distribuição de recursos computacionais garantindo menores comprimentos de cabos e melhores tempos de repostas, surge a segunda geração de sistemas IMA (IMA2G) ou sistemas IMA distribuídos (*Distributed Integrated Modular Avionics – DIMA*) (ANNIGHÖFER; THIELECKE, 2012), que naturalmente demandam comunicação em rede e o uso de Sistemas Operacionais de Tempo Real (RTOS) em conjunto.

3.3 Real Time Operation System

O *Real Time Operation System* (RTOS), ou Sistema Operacional de Tempo Real, em português, é considerado pela ARINC 653 como componente essencial para a consolidação e uniformização do padrão IMA. Nos primeiros sistemas, cada sistema de tempo real era desenvolvido para uma aplicação especifica, o que tornava o processo extremamente repetitivo. A necessidade da existência de uma padronização se mostrou favorável pela industria que aceitou o padrão definido pela ARINC 653 (PRISAZNUK, 2007).

O modelo de comportamento de um sistema RTOS em geral é baseado no agendamento de tarefas, prioridade da tarefas e preemptividade das tarefas. O agendamento permite ao sistema operacional decidir qual tarefa deve ser executada entre as tarefas disponíveis, baseando-se na prioridade da mesma ou no prazo para execução.

A preemptividade garante ao sistema a capacidade de suspender uma tarefa em execução sem a dependência da mesma, ou seja, não é necessário esperar que a tarefa termine sua execução para que a mesma possa ser suspensa.

Considerando que o RTOS pode garantir a de forma robusta o isolamento total entre tarefas, sendo executadas em um mesmo computador, no tempo e na memória, torna-se então seguro deixar de usar uma série de computadores dedicados, um para cada aplicação, e unificar diversas tarefas para serem processadas por um único computador. O RTOS, então, faz o papel de gerante, garantindo que todos programas funcionem de forma segura e isolada, como se estivessem em máquinas separadas. Na abordagem aeronáutica, existem normas que definem requisitos e características dos RTOS usados em aviação, tais como a DO-178.

3.4 Avionics Full Duplex Switched Ethernet

Especificado pela ARINC 664, o *Avionics Full Duplex Switched Ethernet* (AFDX) é um padrão para o desenvolvimento de uma rede de comunicação de dados. A utilização de uma tecnologia padroniza garante ao projeto vários benefícios, dentre eles a velocidade de desenvolvimento e um custo reduzido em relação ao desenvolvimento de uma rede

proprietária, bem como garante a certeza de utilizar um equipamento certificado e que foi testado adequadamente e se encontra nos padrões comerciais. Apesar da semelhança com o padrão Ethernet comum, o AFDX é uma extensão do padrão, apresentando comportamento determinístico (SYSGO, 2007).

O padrão tradicional para redes Ethernet utiliza o sistema *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD) (TANENBAUM, 2002) para acesso ao meio. Isto significa, que o acesso ao barramento (rede) é múltiplo, sendo que qualquer nó pode iniciar a transmissão no momento que desejar, podendo, eventualmente, causar uma colisão. A colisão é detectada, e os nós retransmitem. Esta abordagem gera, inevitavelmente colisões, e sabe-se que quanto maior o uso da rede, maior o número de colisões, prejudicando o desempenho da rede, e dificultando que se entregue pacotes de rede em tempos previsíveis. Esse tipo de comportamento leva muitas vezes a comportamentos de latência na rede, devido a necessidade de retransmissão, o que seria inaceitável para uma aplicação crítica como as aplicações em redes para aeronaves.

Para resolver esse tipo de problema, o AFDX é construído utilizando uma redundância física de componentes, onde dois cabos Ethernet transmitem simultaneamente o mesmo pacote então dois controladores Ethernet fazendo uso de algoritmo apropriado decidem por enviar um dos dois pacote gêmeos para o nó de recebimento (SYSGO, 2007).

3.5 Protocolo IP

O Internet Protocol (IP) é definido pela Request for Comments 791 (RFC 791), sendo complementado pelas RFC 1349, 2474 e 6864. O protocolo de internet permite e se limita à transmissão de blocos de informação entre máquinas *hosts* identificados com endereços de protocolo de internet, comumente referidos como apenas IP (POSTEL, 1981).

Diversos outros protocolos são operam utilizando o IP, adicionando outras funcionalidades como controle de fluxo, garantia de entrega.

3.6 Protocolo UDP

O User Datagram Protocol (UDP) está definido na Request for Comments 768 (RFC 768) e foi desenvolvido por David P. R Reed. Assim como o Transmission Control Protocol (TCP), o protocolo UDP utiliza o protocolo IP como base de funcionamento. O Protocolo UDP não é orientado a transações e, portanto, não garante a entrega dos pacotes e não garante ordem de entrega (POSTEL, 1980).

No datagrama UDP uma mensagem vinda de um processo tem anexada a origem e o endereços de destino, inclusive com a porta. A mensagem então é encapsulada utilizando o protocolo IP e a camada de rede então transmite a mensagem ao destinatário (POSTEL, 1980).

O protocolo UDP apesar de não ser confiável para aplicações em geral, se mostra muito útil em aplicações de tempo real, nas quais a perda de pacotes é preferível em relação ao atraso na rede devido a constante retransmissão e checam de entrega que ocorrem no protocolo TCP (KUROSE; ROSS., 2013).

4 . MATERIAIS E MÉTODOS

4.1 Hardware

Essa seção descreve os materiais físicos utilizados no desenvolvimento do presente projeto, contemplam essa seção os seguintes itens: Beagle Bone Black e Arduino.

4.1.1 Beagle Bone Black

A *Beagle Bone* (BB) (Figura 2) é um projeto *open-source* de um computador de baixo custo, de placa única e com tamanhos diminutos, tendo aproximadamente as mesma dimensões de um cartão de crédito. O projeto possui diversas variantes pela sua natureza *open-source*, todos em geral nomeados por cores, em especial este trabalho citará os modelos *Black* e *Blue*. Tanto o modelo *Black* quanto o *Blue* são voltados para desenvolvedores de todos os níveis. Os dois modelos possuem como principais configurações 512Mb de memória RAM integrada e processador ARM Cortex-M3. Em particular o modelo *Blue* é voltado para desenvolvimento de robôs por possuir algumas especificações extras que permitem controlar nativamente, sem a necessidade de placa auxiliares, atuadores e controladores. Ambos o modelos são compatíveis com o RTOS AIRXKY devido a arquitetura ARM de seus processadores (BEAGLEBONE, 2017).

Figura 2 – Beagle Bone Black

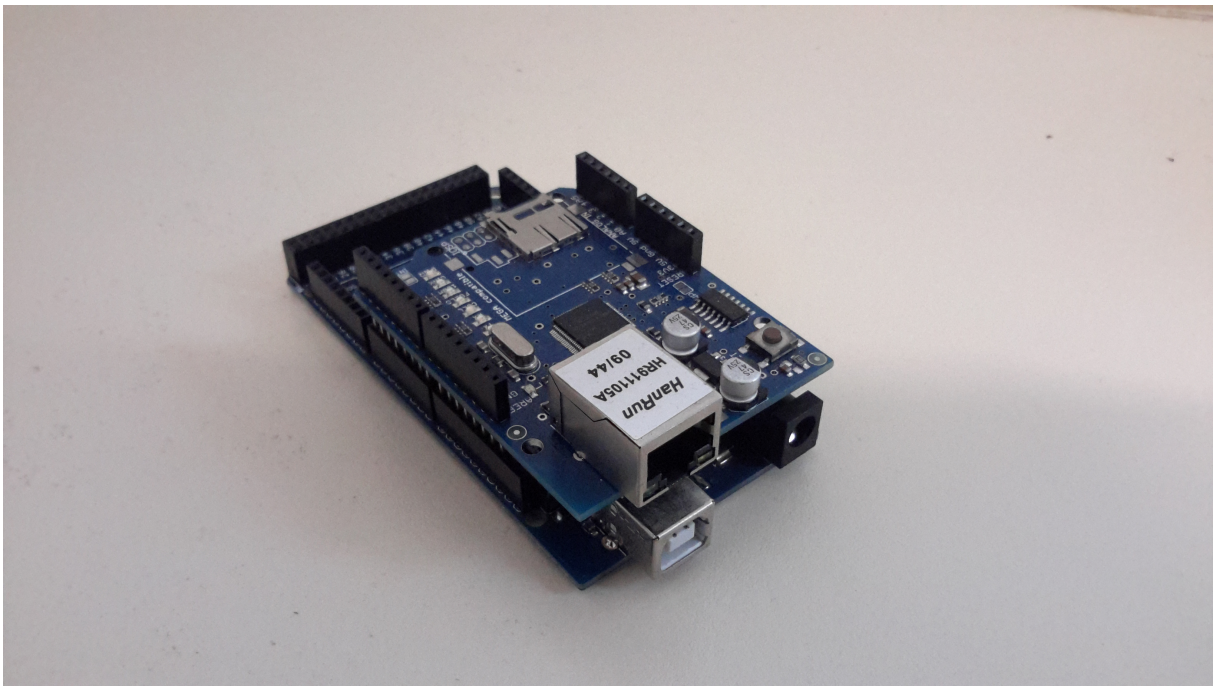
Fonte: do Autor

No presente projeto a Beagle Bone Black tem por função representar o computador responsável por dar suporte a rede DIMA realizando o processamento dos dados enviados na rede, ou seja, o módulo CPM. No decorrer do projeto, a Beagle Bone foi carregada com cada uma das aplicações desenvolvidas para testar cada característica necessária para o funcionamento do da arquitetura proposta.

4.1.2 Arduino

O Arduino (Figura 3), similarmente a Beagle Bone, é um projeto *open source* de uma plataforma de prototipagem eletrônica rápida de baixo custo. Desenvolvido plea *Ivrea Iteraction Design Institute*, o Arduino utiliza de um processador Atmega para processar aplicações desenvolvidas em C++, tento a capacidade de receber diversos periféricos como sensores, LEDs e atuadores. Uma das vantagens do Arduino é sua modularidade, sendo possível acoplar diversos *shields* (placas auxiliares) que estendem as capacidades operacionais da plataforma, permitindo, por exemplo, a utilização de comunicação Ethernet.

Figura 3 – Arduino com *shield* Ethernet



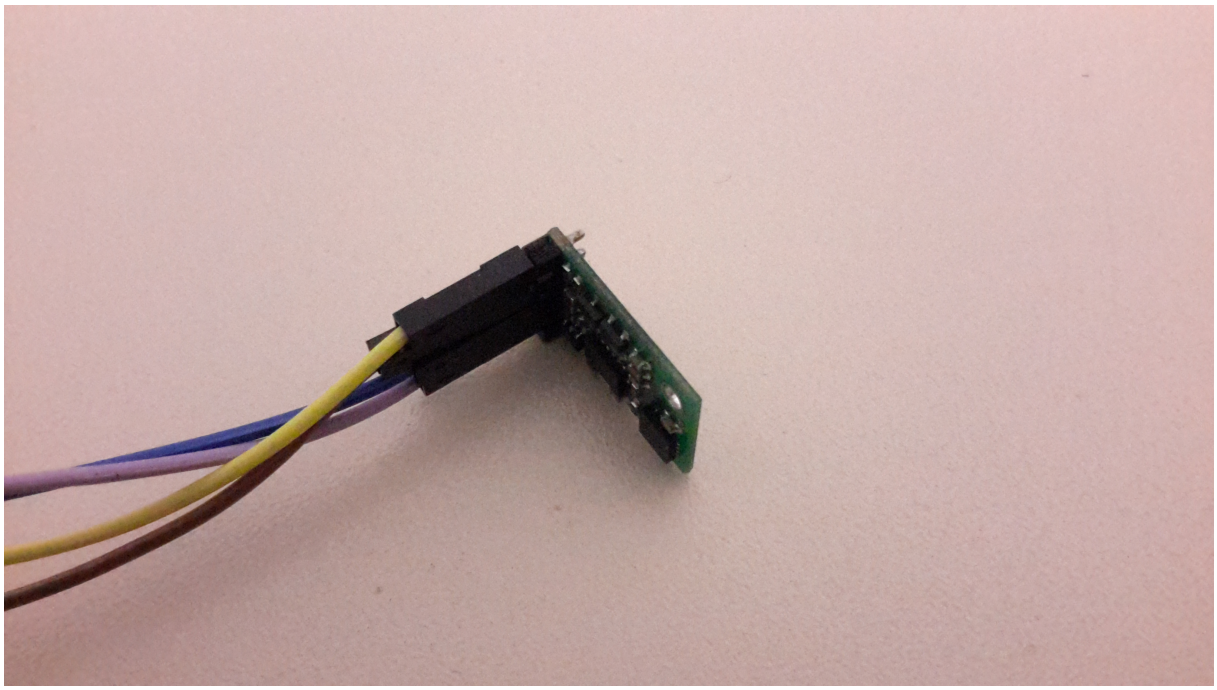
Fonte: do Autor

No presente trabalho, o Arduino foi utilizado para simular os *payloads* de um sistema DIMA. Para que o mesmo fosse integrado a rede foi necessário utilizar o *shield* Ethernet que possui um conector RJ45 fêmea.

4.2 Unidade de Medida Inercial

A unidade de medida inercial (Figura 4), do inglês *inertail measurement unit* (IMU), é um dispositivo capaz de medir a orientação, força e campo magnético ao qual se encontra sobre efeito. O principio funcional da IMU se baseia no uso de acelerômetros, giroscópios e magnetrômetros integrados através de filtros e algoritmos. O uso da IMU é amplamente divulgado na indústria aeroespacial. O presente trabalho faz uso de uma IMU

Figura 4 – Unidade de Medida Inercial modelo IMU01b



Fonte: do Autor

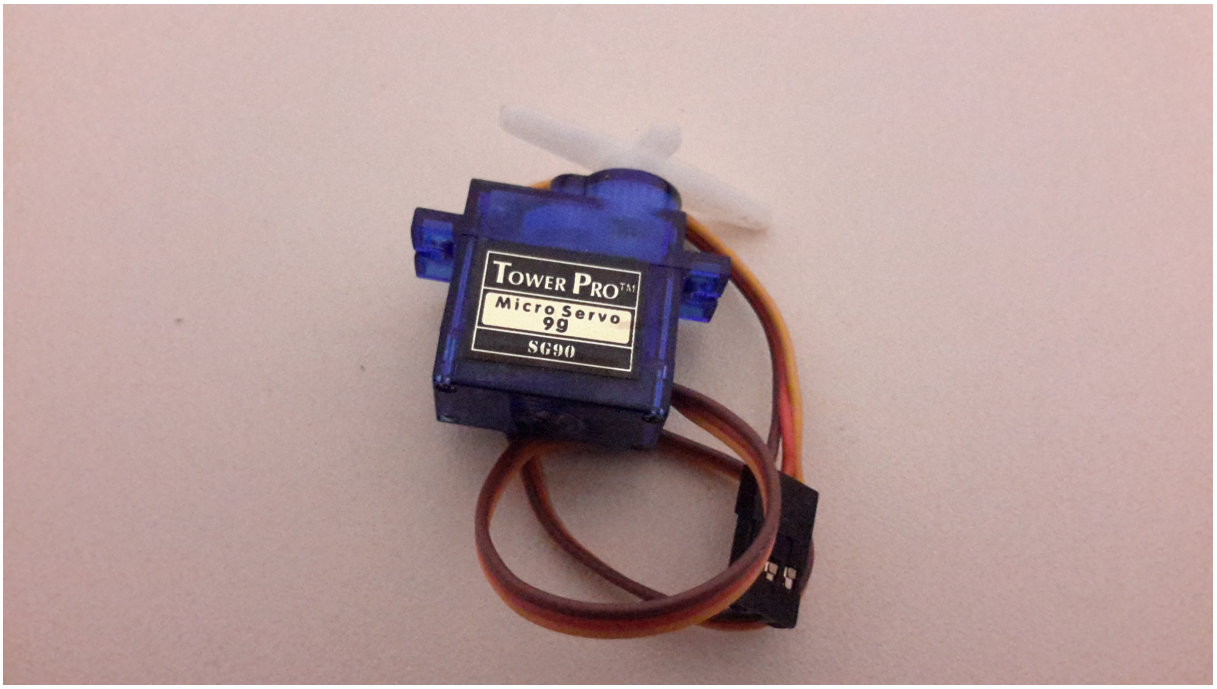
conectada a um Arduino para medir alguns do possíveis movimentos presentes em uma aeronave e que pode ser lidos por sensores, no caso os movimentos de arfagem e rolagem, bem como a direção cardinais. Os dados da IMU são lidos pelo Arduino e disponibilizados via rede através do *shield* Ethernet e de um código desenvolvido para isto utilizando o protocolo UDP (POSTEL, 1980).

4.3 Servo

Como representação de um *payload*, um Arduino conectado com um servo motor (Figura 5) foi adicionado a rede. Servo motores são acionadores comuns para diverso mecanismos em que deseja-se testar um bom controle de precisão. Seu método construtivo permite a construção de um componente que é uma junção entre um redutor e um motor,

e com os devidos conhecimentos das relações de transmissões é possível determinar a posição final do deixo de saída.

Figura 5 – Micro Servo utilizado no projeto



Fonte: do Autor

4.4 Unidade Visualização

Foi utilizado um computador com o sistema operacional Linux como unidade visualizadora. A unidade visualizadora tem por objetivo permitir a observação do que ocorre na rede em operação, bem como rodar a aplicação de horizonte artificial utilizada para melhor monitorar dos dados gerados pelo uso da IMU. O maior objetivo de utilizar uma unidade visualizadora é simular o conceito de *glass cockpit*.

4.5 Materiais Auxiliares

Também foram utilizados nesse trabalho um conversor serial para USB a fim de possibilitar o acesso ao console da placa BB. Outro elemento importante foi um roteador que realizou o papel que deveria ser ocupado por um switch AFDX, esse elemento é responsável por organizar o tráfego de mensagens na rede.

4.6 Software

Essa seção descreve os programas e aplicações utilizados no desenvolvimento do presente projeto, contemplam essa seção os seguintes itens: Minicom, Das U-Boot e AIRXKY.

4.6.1 Minicom

O Minicom é uma aplicação que permite estabelecer uma comunicação serial baseada em texto desenvolvido pela TELIX, porém é *open source*. A aplicação é compatível com diversas distribuições que utilizam UNIX e possui diversas configurações para múltiplos propósitos.

No presente trabalho o Minicom serviu para verificação e observação da inicialização do Das U-Boot, da transferência de arquivos entre a máquina Linux e a Beagle Bone, e finalmente, da atividades da aplicação XKY.

4.6.2 Das U-Boot

O *Das U-Boot*, também referido comumente apenas por *U-Boot*, é um *bootloader* de primeiro e segundo estágio amplamente utilizado em sistemas embarcados.

O projeto inicial foi concebido pela denx engenharia de software sob a uma licença GNU General Public License Version 2 (GNU GPLv2) sendo, portanto, um software livre. Devido a natureza da licença e ao engajamento da comunidade *Open Source*, o U-Boot ganhou várias variações de distribuição, cada qual com suas peculiaridades e diferentes abstrações.

Umas das grandes vantagens do U-Boot é o suporte à entrada USB, o que possibilita a leitura dos pinos seriais e a capacidade de fornecer comandos via teclado utilizando um teclado. Outra vantagem com respeito ao U-Boot é sua flexibilidade, diferentemente dos *bootloaders* de computadores domésticos, o U-Boot confere ao usuário a capacidade de configurar diferentes fatores durante a inicialização do sistema, como por exemplo alocação de memória. Finalmente, o U-Boot oferece suporte a diferentes tipos de memória como SD *card*, HD SATA, entre outros, sendo portanto adequado para a implementação do presente trabalho. Para este projeto, em especial, ele permite realizar o *boot* a partir de uma imagem carregada via rede a partir de um *host*.

4.6.3 AIRXKY

Desenvolvido pela *GMV Inovating Solution*, o AIRXKY, doravante referido apenas como AIR, é um sistema operacional de tempo real voltado para aplicações da industria aeronáutica. O AIR foi desenvolvido com um *Hypervisor* que é totalmente compatível com a norma ARINC 653, ou seja, segue as diretriz para o APEX propostas na norma.

Outra característica importante para este trabalho é a possibilidade de rodar o sistema operacional AIR em arquiteturas ARM, sendo portanto, compatível com a Beagle Bone Black.

O AIR conta ainda com um *toolchain* que facilita o processo de transformar arquivos de configuração XML para os arquivos em formato binários prontos para a compilação final da aplicação.

As aplicações desenvolvidas em XKY são baseadas normalmente no módulo e suas subdivisões, as partições. O módulo é a principal divisão do programa, ele abriga todas as partes da aplicação, desde as partições até *scripts* de inicialização. O principal componente que define o módulo é o arquivo de configuração XML, as principais configurações respeitas por todas as partições do programa. As partições por sua vez, são subdivisões dos módulos que executam atividades menores do processo, podendo operar como diferentes tipos de ação, região de entrada ou saída, execução de rotina definida.

4.6.3.1 Módulos

O arquivo XML de configuração do módulo possui, como linha inicial especificações de metadados, como versão e codificação. Em seguida se incia as especificações do módulo propriamente dito, dentro das *tags XKYModule* estará especificado todos os atributos do código. Note que as tags de definição do módulo possui um atributo para o nome e outra para a versão.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<XKYModule Name="module_0" Version="1">
    <!--Module Code -->
</XKYModule>

```

As *tags* de *Module Partitions* tem por finalidade definir as partições que serão utilizadas na aplicação. Como pode ser visto no trecho de código abaixo, cada partição é definida por uma *tag* de nome *Partition*. Cada *tag* possui como atributo um identificador, um nome, qual utilizará e o nome do arquivo de configuração ao qual está associada. A *tag* do tipo *Partition* pode se fechar nela própria como a do identificador 1, entretanto aquelas que possuem alguma permissão especial possui sintaxe diferente, nelas a permissão é mostrada logo após a declaração da partição como no caso da partição 0.

```
<ModulePartitions>
    <Partition Identifier="0" Name="partition_0" Cores="1"
    Configuration="partition_0/partition.xml" >
            <Permissions>SET_SCHEDULE</Permissions>
    </Partition>
    <Partition Identifier="1" Name="partition_1" Cores="1"
```

```
                Configuration="partition_1/partition.xml" />
            </ModulePartitions>
```

As *tags* de *Module Schedules* tem por finalidade definir o escalonamento de tarefas das partições, definindo o tempo total de execução para todas as partições. Também cabe a essa tag definir os tempos para cada partição individualmente.

```
            <ModuleSchedules>
                <Schedule Identifier="0" Name="schedule_0" MajorFrame="2"
    Initial="true">
                    <PartitionSchedule PartitionIdentifier="0">
                        <Window Offset="0" Duration="1" />
                    </PartitionSchedule>
                    <PartitionSchedule PartitionIdentifier="1">
                        <Window Offset="1" Duration="1" />
                    </PartitionSchedule>
                </Schedule>
            </ModuleSchedules>
```

Uma possibilidade é a utilização de múltiplos escalonamento de tarefas dentro dentro um mesmo módulo, possibilitando diferentes combinações de execuções entre as partições. Note que cada tag *Schedules* possui seu próprio identificador e nome.

```
            <ModuleSchedules>
                <Schedule Identifier="0" Name="schedule_0" MajorFrame="2"
    Initial="true">
                    <PartitionSchedule PartitionIdentifier="0">
                        <Window Offset="0" Duration="1" />
                    </PartitionSchedule>
                    <PartitionSchedule PartitionIdentifier="1">
                        <Window Offset="1" Duration="1" />
                    </PartitionSchedule>
                </Schedule>
                <Schedule Identifier="1" Name="schedule_1" MajorFrame="3"
    Initial="true">
                    <PartitionSchedule PartitionIdentifier="0">
                        <Window Offset="0" Duration="1" PeriodStart="
    true" />
                    </PartitionSchedule>
                    <PartitionSchedule PartitionIdentifier="1">
                        <Window Offset="1" Duration="1" />
                    </PartitionSchedule>
```

```
                    <PartitionSchedule PartitionIdentifier="2">
18                      <Window Offset="2" Duration="1" />
                    </PartitionSchedule>
20              </Schedule>
            </ModuleSchedules>
22
```

Quando deseja-se trabalhar a transmissão de informação entre as partições, tanto em *queuing* ou em *sampling*, conforme a norma ARINC-653, é preciso definir nas configurações do módulo qual partição será responsável por gerar informação e qual será responsável por processa-la.

```
1           <ModuleChannels>
                <Channel Identifier="0" Name="channel_0" QoS="256kb">
3                   <Source>
                        <StandardPartition PartitionIdentifier="0" />
5                   </Source>
                    <Destination>
7                       <StandardPartition PartitionIdentifier="1" />
                    </Destination>
9               </Channel>
            </ModuleChannels>
11
```

4.6.3.2 Partições

Similarmente às atribuições presentes no arquivo de configuração do módulo, o arquivo de configuração XML das partições também possui informações de metadado e as *tags* principais da partição, *Partitions*.

```
            <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2           <Partition Identifier="1" Name="partition_1">
                <!-- Partition Code -->
4           </Partition>
```

As *tags* de *BuildConfiguration* residem outras duas *tags* que determinam pontos principais de funcionalidade da partição. A *tags* do tipo *Libraries* informa à partição as bibliotecas necessárias para que a aplicação alocada nessa partição funcione adequadamente, no caso do código do quadro são bibliotecas necessárias para que a aplicação em C alocada possa executar adequadamente.

```
            <BuildConfiguration>
                <Personality Name="BARE">
                    <BareConfiguration EntryPoint="entry_point"
    CodeMemoryRegion="0" DataMemoryRegion="1" />
                </Personality>
                <Libraries>
                    <Library Name="LIBXKY" Options="" />
                    <Library Name="LIBPRINTF" Options="" />
                </Libraries>
            </BuildConfiguration>

```

As *tags* de *MemoryMap* são responsáveis por definir as alocações de memória da partição e como cada região deve executar com respeito as atribuições de memória.

```
            <MemoryMap>
                    <Region Identifier="0" Name="code" Permissions="RX"
    CacheAttributes="COPY_BACK" />
                    <Region Identifier="1" Name="data" Permissions="RW"
    CacheAttributes="WRITE_THROUGH" />
            </MemoryMap>

```

Existem também as *tags* de *Ports* utilizadas para as partições que recebem data de outras partições. As *Ports* podem ser do tipo *QueuingPorts* ou *SampligPorts*, diferentemente do treco de código abaixo, ambas não são aplicadas ao mesmo tempo, ou se utiliza uma abordagem de *queuing* ou *sampling*.

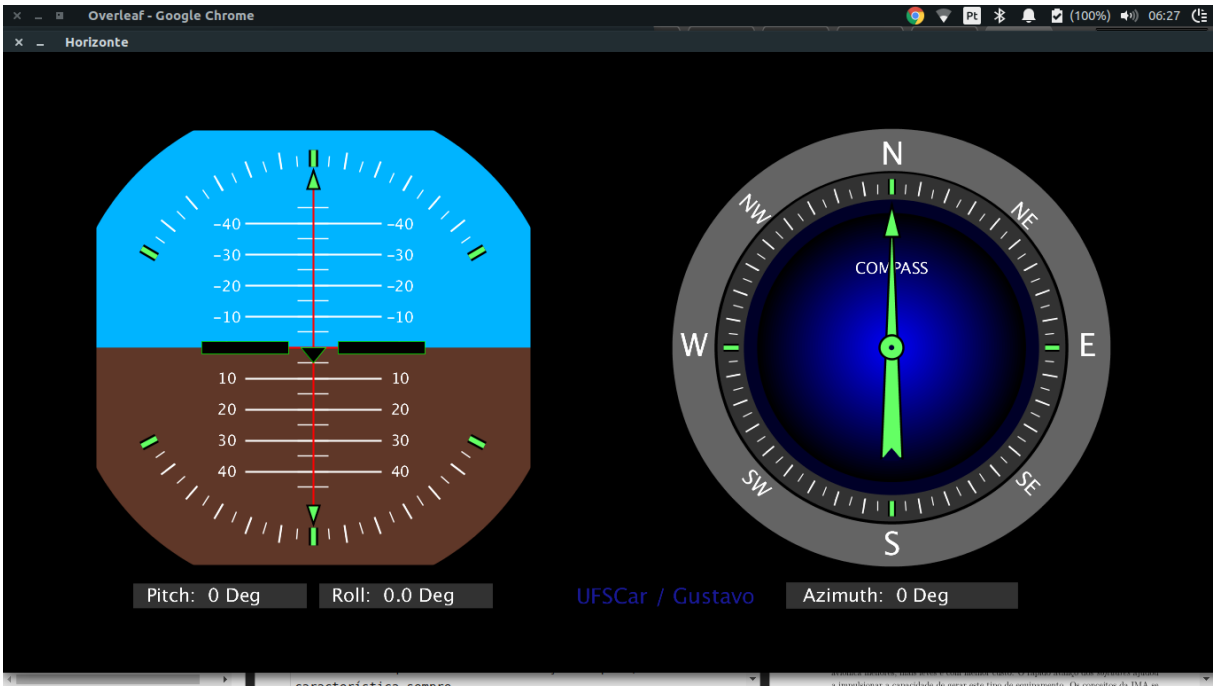
```
            <Ports>
                <QueuingPort ChannelIdentifier="0" Name="port_0"
                    Direction="DESTINATION" MaxMessageSize="16"
    MaxNbMessages="8" />
                <SamplingPort ChannelIdentifier="0" Name="port_0"
                    Direction="SOURCE" MaxMessageSize="32" RefreshPeriod="2
    s" />
            </Ports>

```

4.6.4 Unidade de Visualização

A Unidade de Visualização foi feita usando Processing. Procesing é uma linguagem de programação voltada para o desenvolvimento de projetos que possuam algum tipo de

interface ou outro característica mais voltada para visual. A linguagem é *open source* e utiliza uma IDE muito semelhante a do Arduino IDE. A Figura 6 mostra a aplicação desenvolvida para o presente trabalho, um horizonte artificial e uma bússola, que funciona mostrando dados em tempo real coletados da rede DIMA. A aplicação foi adaptada do trabalho de Bahat(2014).

Figura 6 – Horizonte artificial desenvolvido em Processing



Fonte: do Autor

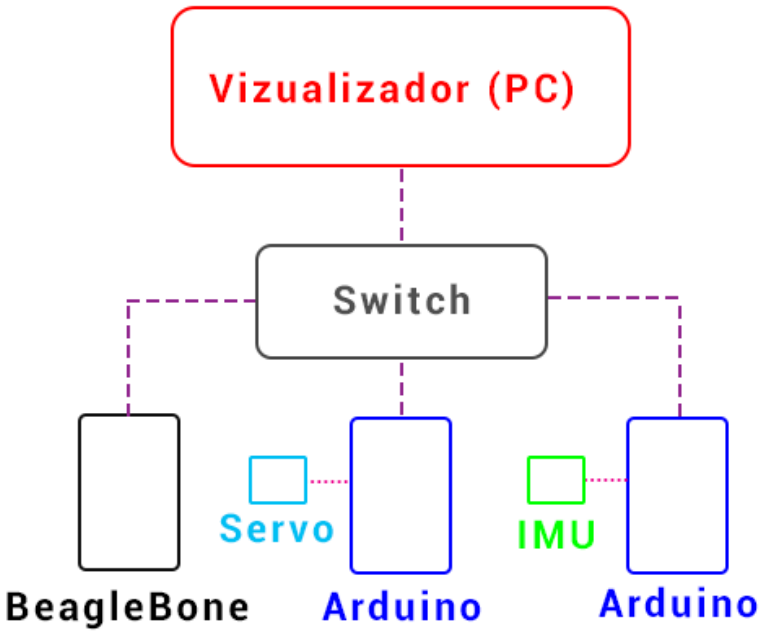
5 . ARQUITETURA DIMA PROPOSTA

A Figura 7 demonstra através de um diagrama de blocos a topologia da rede proposta. Essa rede conta com uma BB, dois Arduinos, um roteador e um visualizador (PC). Todos os componentes existentes na rede operam utilizando o protocolo UDP, sendo atribuído previamente a cada um deles um IP fixo *hard coded* em seus respectivos programas. Cada componente é então conectado ao roteador utilizando cabos Ethernet, representados na cor roxa tracejada, o que permite a comunicação entre cada uma dessas entidades.

A rede deveria ser concebida para fornecer suporte ao padrão *Full Duplex Ethernet Switch*, entretanto será utilizado um roteador de baixo custo pela impossibilidade de se fazer uso do equipamento adequado. O roteador permitirá a realização da entrega de mensagens entre os componentes da rede, similarmente ao papel desempenhado por *switch* AFDX a menos das vantagens de garantia de alocação de tempos por porta.

As seções 5.2 até 5.4 provém informações detalhadas sobre cada componente, seu comportamento e funcionamento. A seção 5.5 fornece explicações de como o processamento ocorre na BB utilizando o AIR. Finalmente a seção 5.1 explica os por menores da comunicação entre os componentes das redes.

Figura 7 – Diagrama de blocos da rede proposta



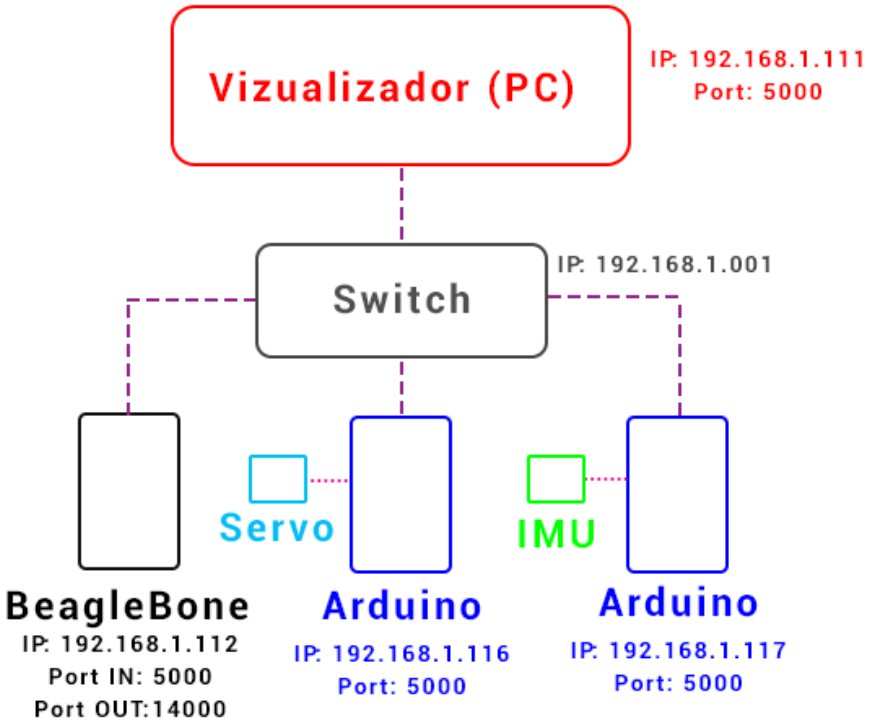
Fonte: do Autor

5.1 Integração e comunicação

Para permitir a comunicação entre os diferentes componentes da rede proposta,o AIR possui a capacidade de enviar e receber mensagens utilizando o protocolo UDP. Apesar da presença de um roteador, o que possibilita a utilização de alocação de IP de forma dinâmica, cada componente possui especificado no código da aplicação o seu próprio IP estático e os demais IPs estáticos necessários para comunicação, bem como as portas disponíveis para envio e recebimento de mensagem de cada IP. Note que o AIR precisa ter associado duas portas, uma para recebimento e outro para envio de mensagem.

Na comunicação dessa rede (Figura 8), todas as mensagens devem passar pelo roteador e então pela BB para serem processadas, dessa forma na configurações do AIR foi especificado cada uma das partições necessárias para operação da rede, os IPs associados a cada um dos componentes com os quais está associada.

Figura 8 – Diagrama da rede proposta com IP e portas atribuídos



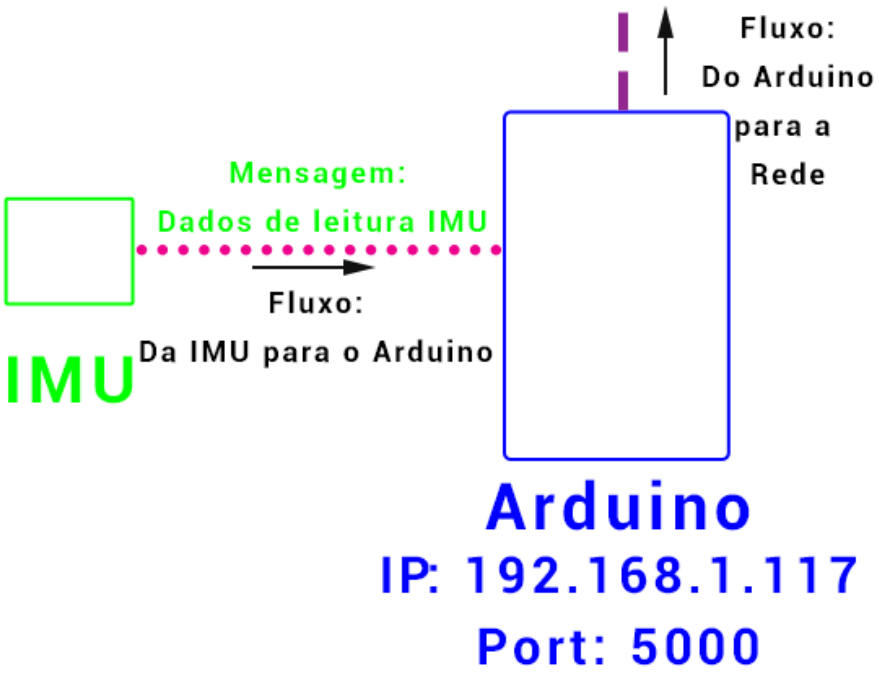
Fonte: do Autor

5.2 Sensores

Como citado na seção 4.2, o presente trabalho utiliza como sensor uma unidade de medida inercial, para permitir a sua leitura dos dados coletados é um utilizado um Arduino. Esse conjunto composto entre a IMU e o Arduino representa um *payload* de entrada genérico.

O Arduino captura os valores de leitura da IMU para permitir seu envio na rede. Utilizando o protocolo UDP o pacote de dados é enviado para a BB que os recebe utilizando a partição de I/O (Figura 9). Por uma questão de desempenho não são todas as leituras que são enviadas para a BB.

Figura 9 – Diagrama de blocos leitura de da IMU utilizando Arduino



Fonte: do Autor

5.3 Atuadores

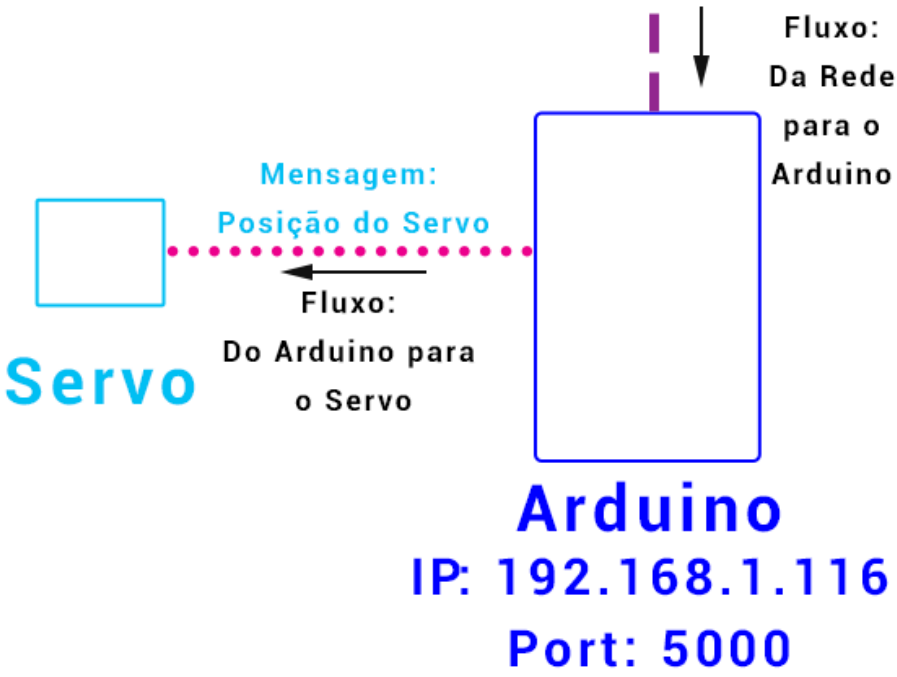
Similarmente à IMU, o servo utilizado como atuador no projeto utiliza de um Arduino para o controlá-lo. Esse conjunto de servo e Arduino é o equivalente ao um *payload* de saída genérico.

O Arduino recebe, através da rede, os valores da leitura da IMU processados pela BB, assim como os outro elementos via protocolo UDP. O Arduino processa os dados recebidos para que possam ser enviados para o servo, desta forma a posição do servo pode ser alterada com relação aos dados coletados pela IMU (Figura 10).

5.4 Visualização de dados

Em ordem de permitir a visualização dos processos de recebimento de dados e envio, o unidade de visualização (PC) utiliza uma aplicação de horizonte artificial para simular

Figura 10 – Diagrama de blocos para posicionamento do Servo utilizando Arduino



Fonte: do Autor

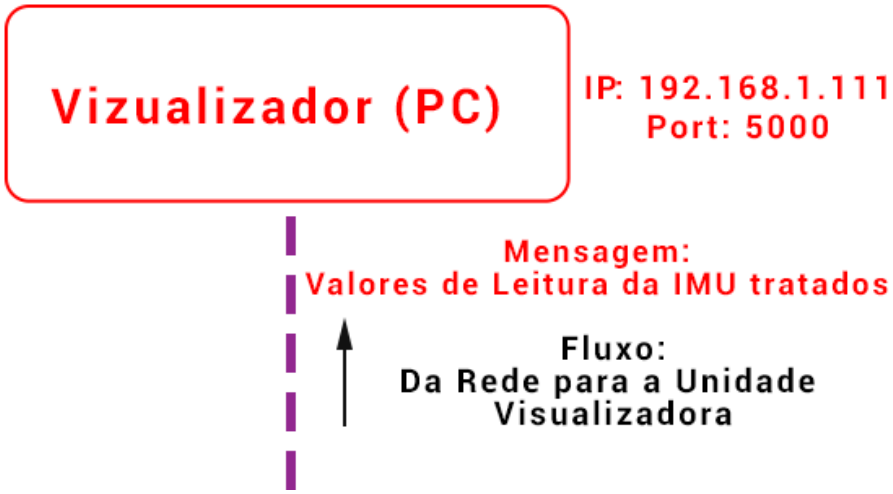
os valores obtidos na IMU. A aplicação de horizonte artificial foi desenvolvida utilizando a linguagem de programação Processing, com base no trabalho de Bahat (BAHAT, 2014).

Assim como o caso dos atuadores e dos sensores, a unidade visualizadora utiliza o protocolo UDP para receber as mensagem advindas da BB, nesse caso os valores de posição obtidos da IMU. Os valores são recebidos na aplicação do horizonte artificial e convertida para os parâmetros necessários para que a representação ocorra adequadamente.

5.5 Controle

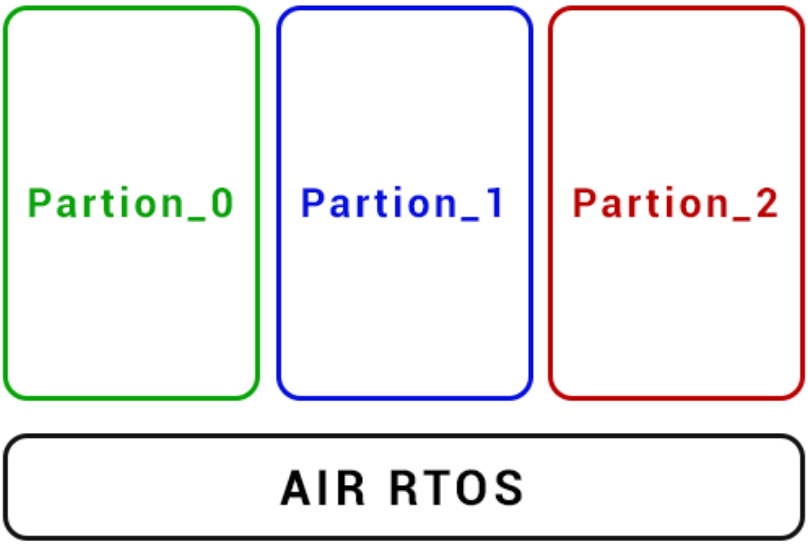
Para mediar o envio e recebimento dos dados existe uma terceira partição no AIR responsável pelo controle dos dados (Figura 12, *Partion_1*). Os dados recebidos pela partição de IOP são colocados em uma fila para que possam ser processados, a fila é então lida pela partição de controle esse dados são então colocados na fila de novo, pego pela partição de I/O que então envia para o Arduino que controla os atuadores.

Figura 11 – Diagrama de blocos do recebimento pela unidade visualizadora



Fonte: do Autor

Figura 12 – Diagrama de blocos do AIR RTOS



Fonte: do Autor

6 . RESULTADOS

Na Figura 13 é possível observar a disposição final da rede e seus elementos associados. Assim como proposto o roteador, que aqui substitui o *switch* AFDX, tem conectado a si todos os cabos Ethernet originários dos diversos dispositivos da rede. O computador presente realiza a função de visualizador para rede como mencionado na seção 4.4.

6.1 Modo de operação

A presente rede da Figura 13 opera através dos envios de mensagens via protocolo UDP (*User Datagram Protocol*), que tem melhor desempenho de tempo real quando comparado ao TCP (*Transfer Control Protocol*). Os dados de leitura da IMU são capturados pelo Arduino a ela ligado, e enviados via rede respeitando um determinado número de leituras. A mensagem então é enviada para a BeagleBone que captura os pacotes utilizando a partição de I/O (Apêndice D , seção D.2.1) e os dados são então enviados para a próxima partição responsável por realizar o controle, essa partição então processa os valores obtidos, determinado dados de controle para serem enviados a atuadores.

Dessa forma, os valores processados na partição de controle são enviados para a partição de I/O e enviados novamente para a rede. Os valores calculados pelo controle são enviados para o Arduino conectado ao servo o servo motor, que recebe os valores utilizando um servidor UDP semelhante ao utilizado para a IMU. Este valores também são enviados para a unidade visualizadora, nela os dados recebidos são utilizado pelo o *script* do Processing (Apêndice D, seção D.4) para então gerar a simulação de horizonte artificial e de bússola com atualização automática.

Figura 13 – Rede DIMA proposta

Fonte: do Autor

7 . CONCLUSÃO

As novas demandas de automação, conforto e segurança, vêm justificando cada vez mais o uso de arquiteturas IMA e DIMA para sistemas de aviônica. O estudo, desenvolvimento e pesquisa nesta área se torna relevante para o Brasil e para a formação de profissionais competentes na área. Dessa forma, este trabalho apresentou um estudo de sistemas de aviônica de próxima geração, uma implementação funcional e análise de seus resultados.

Para fins didáticos e de pesquisa, a rede foi implementada com materiais de baixo custo (Placas Arduino e Beagle Bone), porém sua arquitetura é totalmente compatível com uma rede que pode ser usada em aplicações reais. Assim, foi possível perceber que a rede proposta, apesar de simples, apresenta e se comporta como uma rede DIMA convencional. A menos das simplificações propostas, a rede tem a capacidade de receber entradas de dispositivos externos bastando ao mesmo apenas enviar os seus dados utilizando o protocolo UDP. Com isso, esta rede é capaz de receber entradas de qualquer dispositivo real que utilize esse padrão normalizado de comunicação, oferecendo flexibilidade e escalabilidade ao sistema. Similarmente aos dados de entrada, a rede proposta também é capaz de enviar dados para dispositivos reais que atendessem os requisitos de comunicação especificados.

Quanto ao sistema de controle, implementou-se uma lei simplificada de controle apenas para validar a rede proposta e sua arquitetura. É possível concluir que a mesma abordagem seria aplicável à uma rede DIMA real a menos do processamento. Os dados de entrada utilizados no presente trabalho não representam o mesmo montante que vários dispositivos reais gerariam em um rede real, portanto ainda é necessário realizar maiores estudos para verificar se os tempos de processamento e transmissão são suficientes ou não, ficando para trabalhos futuros o entendimento deste requisito.

Um ponto conclusivo é que a forma de processamento opera adequadamente como esperado de um RTOS para ARINC653, todos os dados recebidos são colocado em em fila e são processados em uma partição especificada para isso, respeitando requisitos de prioridade com adequado isolamento temporal e físico entre tarefas, comunicação, e outras partes do sistema.

Finalmente, nota-se que o uso de ferramentas profissionais para a implementação do sistema, como o sistema AIR, gentilmente cedido pela GMV, demandou bastante tempo de estudo, configuração e aprendizado sobre sistemas operacionais de tempo real (RTOS), sobre a norma ARINC, configurações via XML, comunicação em rede. Além disso, ressalta-se ainda a natureza interdisciplinar deste trabalho, integrando software, hardware, sistemas de tempo real, comunicação em redes, protocolos de redes, leituras de sensores, acionamento de atuadores, programação de microcontrolares, dentre outras atividades.

7.1 Trabalhos futuros

O desenvolvimento deste trabalho foi um primeiro passo no desenvolvimento de uma arquitetura DIMA para pesquisas e estudo de sistemas de aviônica de próxima geração. Sugerem-se diversos trabalhos futuros, tais como:

- Integração do *Health Monitor*, especificado pela ARINC 653 e que não foi contemplado nesee trabalho;
- Desenvolvimento de uma aplicação real de controle;
- Análise de robustez, requisitos de tempo real e desempenho da rede;
- Testes com um Switch AFDX convencional, para validar a operação sobre infraestrutura AFDX;
- Conexão de mais módulos na rede e operação distribuída.

REFERÊNCIAS

ANNIGHÖFER, B.; THIELECKE, F. Supporting the design of distributed integrated modular aavionics systems with binary programming. **journal**, volume, n. number, p. pages, month 2012. Note.

BAHAT, M. **Artificial Horizon and Compass Using Arduino-Processing-MPU6050**. 2014. Disponível em: <http://mfurkanbahat.blogspot.com.br/2014/11/artificial-horizon-and-compass-using.html>. Acesso em: 22 dez 2017.

BEAGLEBONE. **BeabgleBoneBlack**. 2017. Disponível em: <https://beagleboard.org/black>. Acesso em: 22 dez 2017.

DETLEV SCHAADT, CTO, SYSGO AG. **Concept, Design, Implementation and Beyond**: White paper afdx/arinc 664. Rio de Janeiro, 2007. Disponível em: <http://www.cems.uwe.ac.uk/~a2-lenz/n-gunton/afdx_arinc664.pdf>. Acesso em: 22 dez 2017.

EFKEMANN, C.; PELESKAR, J. Model-based testing for the second generation of integrated modular avionics. In: SOFTWARE TESTING, VERIFICATION AND VALIDATION WORKSHOPS (ICSTW), 2011 IEEE COMPUTER SOCIETY FOURTH INTERNATIONAL CONFERENCE, 2011, Berlin, Germany. Berlin, Germany: IEEE Computer Society, 2011.

KUROSE, J. F.; ROSS., K. W. Computer networking : a top-down approach. In: _____. 6. ed.. ed. São Paulo: Pearson, 2013. p. 198–200.

LEITE, V. R. Fly by wire. **Special Issue on Aeronautics**, vol 5, n. 6, p. pag. 80–106, Nov. 2014.

LI, Z.; LI, Q.; XIONG, H. Avionics clouds: A generic scheme for future avionics systems. In: DIGITAL AVIONICS SYSTEMS CONFERENCE (DASC), 2012, 31., 2012, Williamsburg, VA, USA. Williamsburg, VA, USA: IEEE Computer Society, 2012.

MOIGNE, R. L.; PASQUIER, O.; CALVEZ, J.-P. A generic rtos model for real-time systems simulation with systemc. In: **Proceedings of the Conference on Design, Automation and Test in Europe - Volume 3**. Washington, DC, USA: IEEE Computer Society, 2004. p. 30082–. ISBN 0-7695-2085-5. Disponível em: <http://dl.acm.org/citation.cfm?id=968880.969264>.

POSTEL, J. **User Datagram Protocol**. 1980. Howpublished. Disponível em: <https://www.ietf.org/rfc/rfc768.txt>. Acesso em: 22 dez 2017.

POSTEL, J. **Internet Protocol**. 1981. Disponível em: <https://tools.ietf.org/html/rfc791>. Acesso em: 22 dez 2017.

PRISAZNUK, P. J. Integrated modular avionics. In: AEROSPACE AND ELECTRONICS CONFERENCE, 1992. NAECON 1992., 1992, Dayton, OH, USA. Dayton: IEEE Computer Society, 1992.

PRISAZNUK, P. J. Arinc 653 role in integrated modular avionics (ima). In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 2008. DASC '08. IEEE COMPUTER SOCIETY/AIAA 27TH, 27., 2008, St. Paul, MN, USA. St. Paul, MN, USA: IEEE Computer Society, 2007.

ŠEGVIĆ, M.; NIKOLIĆ, K. K.; IVANJKO, E. A proposal for a fully distributed flight control system design. In: INFORMATION AND COMMUNICATION TECHNOLOGY, ELECTRONICS AND MICROELECTRONICS (MIPRO), 39., 2016, Opatija, Croatia. Opatija, Croatia: IEEE Computer Society, 2016.

TANENBAUM, A. **Computer Networks**. 4th. ed. [S.l.]: Prentice Hall Professional Technical Reference, 2002. ISBN 0130661023.

WARDEN, N. Overview and effect of deterministic ethernet on test strategies. In: AUTOTESTCON, 2017 IEEE COMPUTER SOCIETY, 2017, Schaumburg, IL, USA. Schaumburg, IL, USA: IEEE Computer Society, 2017.

WATKINS, C. B.; WALTER, R. Transitioning from federated avionics architectures to integrated modular avionics. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 2007. DASC '07. IEEE COMPUTER SOCIETY/AIAA 26TH, 26., 2007, Dallas. Dallas, TX, USA: IEEE Computer Society, 2007.

WOLFIG, R.; JAKOVLJEVIC, M. Distributed ima and do-297: Architectural, communication and certification attributes. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 2008. DASC '08. IEEE COMPUTER SOCIETY/AIAA 27TH, 27., 2008, St. Paul, MN, USA. St. Paul, MN, USA: IEEE Computer Society, 2007.

APÊNDICE A – . INTRO

A.1 U-Boot

Esta sessão que se desenrolara, demonstra os passos necessarios para tornar disponível o U-Boot, descrevendo em detalhes os procedimentos utilizados no presente trabalho. As informações aqui apresentadas são baseadas nas orientações fornecidas por Nelson.

A.1.1 Obtendo o U-Boot

Como citado anteriormente, o *U-Boot* tem por finalidade permitir a inicializacao de sistemas embarcados. Apesar das inúmeras derivações que o programa sofreu ao longo dos anos, o presente trabalho utilizara da distribuicao oferecida pela própria DENX.

O código fonte do U-Boot foi obtido no repositório disponível no site especializado em versionamento GitHub. O código fonte pode ser obtido realizando download diretamente da pagina, ou diretamente pelo terminal utilizando a ferramenta de versionamento git, através do comando "git clone https://github.com/u-boot/u-boot". A este código fonte foram aplicadas duas atualizações de correcao, elas foram obtidas e instaladas utilizando os seguintes comandos no terminal do Linux.

```
    git clone https://github.com/u-boot/u-boot
2   cd u-boot/
    wget -c https://rcn-ee.com/repos/git/u-boot-patches/v2017.09/0001-
    am335x_evm-uEnv.txt-bootz-n-fixes.patch
4   wget -c https://rcn-ee.com/repos/git/u-boot-patches/v2017.09/0002-U-
    Boot-BeagleBone-Cape-Manager.patch

6   patch -p1 < 0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch
    patch -p1 < 0002-U-Boot-BeagleBone-Cape-Manager.patch
8
```

A.1.2 Obtendo o GNU Compiler Collection (GCC)

Em ordem de possibilitar a utilizacao do U-Boot é necessario compila-lo, para tanto foi utilizado uma versão do Linaro GCC para maquinas hosts de 64 bits, este compilador permite compilar para arquitetura ARM, utilizada em placa beaglebone.

Utilizando o terminal do Linux e o comando wget obteve-se o arquivo na extensão tar, depois de descomprimido, o o compilador foi exportado para a variavel $PATH em ordem de se tornar disponível globalmente no sistema.

```
1   wget -c https://releases.linaro.org/components/toolchain/binaries
    /6.4-2017.08/arm-linux-gnueabihf/gcc-linaro-6.4.1-2017.08-x86_64_arm-
    linux-gnueabihf.tar.xz
```

```
    tar xf gcc-linaro-6.4.1-2017.08-x86_64_arm-linux-gnueabihf.tar.xz
3   export CC=`pwd`/gcc-linaro-6.4.1-2017.08-x86_64_arm-linux-gnueabihf/bin
    /arm-linux-gnueabihf-
```

A.1.3 Compilando o U-Boot

Finalmente o U-Boot foi compilado o GCC. Para tanto foram utilizado os comandos para limpar o diretório de uma possível antiga versão do compilador, seguido pelo comando que configura o compilador para o processador da Beaglebone e pelo comando de compilacao.

```
1   make ARCH=arm CROSS_COMPILE=${CC} distclean
    make ARCH=arm CROSS_COMPILE=${CC} am335x_evm_defconfig
3   make ARCH=arm CROSS_COMPILE=${CC}
```

A.2 Configurações de memória

Para habilitar a inicializacao utilizando o novo U-Boot foi necessario configurar um Secure Digital Card (SD card) para que o mesmo suportasse as especificações de inicializacao do bootloader. Para obter tais especificações utilizou-se o utilitario fdisk disponível no Linux Ubuntu.

Utilizando o utilitario fdisk criou-se duas partições uma com especificações de boot, FAT32 e tamanho 71Mb. Para esta particao foram copiados os arquivos MLO e Uboot.img gerados utilizando o programa U-Boot, e o arquivo UEnv.txt criado utilizando um editor de texto.

A.3 Instalação e Configuração do TFTP

Para que a instalacao do TFTP ocorra basta utilizar o comando terminal da maquina Linux host.

```
1   sudo apt-get install tftpd-hpa #Servidor
    sudo apt-get install tftp-hpa #Cliente
3
```

O servidor precisa então ser configurado. Utilizando o editor nano, o arquivo de configuracao do servidor pode ser editado no caminho etcdefaulttftpd-hpa.

```
    TFTP_USERNAME="tftp"
2   TFTP_DIRECTORY= /var/lib/tftpboot/
    TFTP_ADDRESS="0.0.0.0:69"
4   TFTP_OPTIONS="--secure"
```

O servidor pode então ser reiniciado para garantir a aplicacao das novas configurações.

```
    sudo systemctl restart tftp
2
```

A.4 Compilando o AIRXKY

Testar alias,e então escrever sobre isso A distribuicao do AIR possui em seu código fonte o toolchain necessario para sua compilacao e instalacao, para realizar tal procedimento é necessario possuir instalado na maquina host o Python 2.7 e o pacote mako instalados.

No diretório que contém o arquivo de extensão tar da aplicacao, os seguintes comando devem ser executados.

```
1       tar -xvf toolchain.tar.xz
        cd toolchain
3       ./build -target arm-eabi -host linux
```

Após toda compilacao das diversas ferramentas do AIR, ficara disponível o script xmf_make que prepara as configurações necessaria utilizando os arquivos xml, tornando o arquivo pronto para gerar o .exe da aplicacao. Como é necessario especificar sempre o caminho para utilizar o script, a criacao de um alias agiliza o processo de compilar.

```
1       alias premake='~/Desktop/TCC/xky/xky/xmf_make'
```

É então necessario exportar o caminho contento as ferramentas compiladas dando ao comando make a capacidade de compilar aplicações AIRXKY.

```
1   export PATH=$PATH:/home/tamanaka/Desktop/TCC/xky_v1.2/linux/arm-xky-
    eabi/bin
```

A.5 Carregando um aplicacao na BeagleBone

```
1       setenv bootfile 192.168.1.109:XKYAPP.exe
        run netloadimage
3       bootelf 0x82000000
```

A.6 Aplicacao Final

Na aplicacao final, o objetivo deste trabalho, a entrada do Arduíno responsavel por coletar informações teve sua entrada dada por uma mensagem substituida por uma IMU, os dados são então transmitidos para Beagle Bone, repassaddos entre as partições e devolvidos para um Arduíno contento um servo e para o computador host rodando um script de horizonte artificial.

APÊNDICE B – . HELLO WORLD

Obviamente, como pratica recorrente no desenvolvimento de uma aplicacao o primeiro teste deve ocorrer com um programa simples, comumente referido como "Hello World". Este programa tem por unica funcionalidade escrever na tela as palavras Hello World. Sua utilidade é observar se as ferramentas de compilacao funcionam adequadamente e se a aplicacao consegue ser inicializada na BeagleBone.

B.1 module.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<XKYModule Name="module_0" Version="1">

    <ModulePartitions>
        <Partition Identifier="0" Name="partition_0" Cores="1" Configuration="partition_0/partition.xml" >
            <Permissions>SET_PARTITION_MODE</Permissions>
        </Partition>
    </ModulePartitions>

    <ModuleSchedules>
        <Schedule Identifier="0" Name="schedule_0" MajorFrame="1" Initial="true">
            <PartitionSchedule PartitionIdentifier="0">
                <Window Offset="0" Duration="1" />
            </PartitionSchedule>
        </Schedule>
    </ModuleSchedules>
</XKYModule>
```

B.2 partition.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Partition Identifier="0" Name="partition_0">
    <BuildConfiguration>
        <Personality Name="BARE">
            <BareConfiguration EntryPoint="entry_point" CodeMemoryRegion="0" DataMemoryRegion="1" />
        </Personality>
        <Libraries>
            <Library Name="LIBXKY" Options="" />
            <Library Name="LIBPRINTF" Options="" />
        </Libraries>
```

```
11              </BuildConfiguration>
                <MemoryMap>
13                  <Region Identifier="0" Name="code" Permissions="RX"
    CacheAttributes="COPY_BACK" />
                    <Region Identifier="1" Name="data" Permissions="RW"
    CacheAttributes="WRITE_THROUGH" />
15              </MemoryMap>
            </Partition>
17
```

B.3 main.c

```
2   #include <xky.h>
    #include <bare.h>
4   #include <string.h>
    #include <xky_printf.h>
6
    void entry_point() {
8
        while (1) {
10
            xky_time_t time_stamp;
12
            xky_printf("Hello World\n",);
14          xky_printf("\n");
            bare_wake_in_next_mtf();
16      }
    }
18
```

APÊNDICE C – . TRANSMISSÃO ENTRE PARTIÇÕES

C.1 module.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<XKYModule Name="module_5" Version="1">

    <ModulePartitions>
        <Partition Identifier="0" Name="partition_0" Cores="1"
Configuration="partition_0/partition.xml" />
        <Partition Identifier="1" Name="partition_1" Cores="1"
Configuration="partition_1/partition.xml" />
    </ModulePartitions>

    <ModuleSchedules>
        <Schedule Identifier="0" Name="schedule_0" MajorFrame="2"
Initial="true">
            <PartitionSchedule PartitionIdentifier="0">
                <Window Offset="0" Duration="1" />
            </PartitionSchedule>
            <PartitionSchedule PartitionIdentifier="1">
                <Window Offset="1" Duration="1" />
            </PartitionSchedule>
        </Schedule>
    </ModuleSchedules>

    <ModuleChannels>
        <Channel Identifier="0" Name="channel_0" QoS="256kb">
            <Source>
                <StandardPartition PartitionIdentifier="0" />
            </Source>
            <Destination>
                <StandardPartition PartitionIdentifier="1" />
            </Destination>
        </Channel>
    </ModuleChannels>
</XKYModule>


```

C.2 partion.xml

C.2.1 partition_0

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
        <Partition Identifier="0" Name="partition_0">
            <BuildConfiguration>
                <Personality Name="BARE">
                    <BareConfiguration EntryPoint="entry_point"
CodeMemoryRegion="0" DataMemoryRegion="1" />
                </Personality>
                <Libraries>
                    <Library Name="LIBXKY" Options="" />
                    <Library Name="LIBPRINTF" Options="" />
                </Libraries>
            </BuildConfiguration>

            <Ports>
                <QueuingPort ChannelIdentifier="0" Name="port_0"
Direction="SOURCE" MaxMessageSize="32" MaxNbMessages="8" />
            </Ports>

            <MemoryMap>
                <Region Identifier="0" Name="code" Permissions="RX"
CacheAttributes="COPY_BACK" />
                <Region Identifier="1" Name="data" Permissions="RW"
CacheAttributes="WRITE_THROUGH" />
            </MemoryMap>
        </Partition>
```

C.2.2 partition_1

```
        <?xml version="1.0" encoding="UTF-8" standalone="no"?>
        <Partition Identifier="1" Name="partition_1">
            <BuildConfiguration>
                <Personality Name="BARE">
                    <BareConfiguration EntryPoint="entry_point"
CodeMemoryRegion="0" DataMemoryRegion="1" />
                </Personality>
                <Libraries>
                    <Library Name="LIBXKY" Options="" />
                    <Library Name="LIBPRINTF" Options="" />
                </Libraries>
            </BuildConfiguration>

            <Ports>
                <QueuingPort ChannelIdentifier="0" Name="port_0"
Direction="DESTINATION" MaxMessageSize="16" MaxNbMessages="8" />
            </Ports>
```

```
17                  <MemoryMap>
                        <Region Identifier="0" Name="code" Permissions="RX"
    CacheAttributes="COPY_BACK" />
19                      <Region Identifier="1" Name="data" Permissions="RW"
    CacheAttributes="WRITE_THROUGH" />
                    </MemoryMap>
21              </Partition>
```

C.3 main.c

C.3.1 partition_0

```
1
            #include <xky.h>
3           #include <bare.h>
            #include <string.h>
5           #include <xky_printf.h>

7           void entry_point(void) {

9               xky_u32_t i;
                xky_id_t my_id, port_id;
11              xky_port_status_t port_status;

13              my_id = xky_syscall_get_partition_id(NULL);
                port_id = xky_syscall_get_port_id("port_0");
15              xky_syscall_get_port_status(port_id, &port_status);

17              xky_printf(
                        "                    Port id: %i\n"
19                      "                Port status:\n"
                        "                      Type - %i\n"
21                      "                 Direction - %i\n"
                        " Max. Number of Messages - %i\n"
23                      "       Max. Message size - %i\n"
                        "          Refresh Period - %lli\n"
25                      "       Messages in queue - %i\n\n",
                        port_id,
27                      port_status.type,
                        port_status.configuration.direction,
29                      port_status.configuration.max_nb_message,
                        port_status.configuration.max_message_size,
31                      port_status.configuration.refresh_period,
```

```
                        port_status.nb_message);

            while (1) {


                        char buffer[32] = { 0 };
                        strcpy(buffer,"Hello World");
                        xky_printf("Particao %i: esta enviando %s (%i)\n",
                                my_id, buffer,
                                xky_syscall_write_port(port_id, buffer,
    strlen(buffer)));

                    bare_wake_in_next_mtf();

                }

            }


```

C.3.2 partition_1

```
        #include <xky.h>
        #include <bare.h>
        #include <xky_printf.h>

        void entry_point(void) {

            xky_sz_t m_size;
            xky_time_t m_time;
            xky_u32_t i, m_status;
            xky_id_t my_id, port_id;
            xky_port_status_t port_status;

            my_id = xky_syscall_get_partition_id(NULL);
            port_id = xky_syscall_get_port_id("port_0");
            xky_syscall_get_port_status(port_id, &port_status);

            xky_printf(
                    "                    Port id: %i\n"
                    "                Port status:\n"
                    "                      Type - %i\n"
                    "                 Direction - %i\n"
                    " Max. Number of Messages - %i\n"
```

```
24                      "        Max. Message size - %i\n"
                        "           Refresh Period - %lli\n"
26                      "        Messages in queue - %i\n\n",
                        port_id,
28                      port_status.type,
                        port_status.configuration.direction,
30                      port_status.configuration.max_nb_message,
                        port_status.configuration.max_message_size,
32                      port_status.configuration.refresh_period,
                        port_status.nb_message);
34

36              while (1) {

38                  char buffer[32] = { 0 };

40

                    m_size = xky_syscall_read_port(port_id, buffer, &m_time
    , &m_status);
42                  // if (m_size > 0) {
                        xky_printf("Particao %i: recebeu %s (%i) (%lli)",
    my_id, buffer, m_size, m_time);
44                      xky_printf("\n");

46

48                  bare_wake_in_next_mtf();

50

            }
52

        }
54
```

APÊNDICE D – . APLICACAO FINAL

D.1 module.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<XKYModule Name="iop_example" Version="1">

    <ModulePartitions>
        <Partition Identifier="0" Name="p0" Cores="1" Configuration
="p0/partition_0.xml">
            <Permissions>SUPERVISOR;</Permissions>
        </Partition>
        <Partition Identifier="1" Name="p1" Cores="1" Configuration
="p1/partition_1.xml">
            <Permissions>SUPERVISOR;</Permissions>
        </Partition>
        <Partition Identifier="2" Name="p2" Cores="1" Configuration
="p2/partition_2.xml" />

    </ModulePartitions>

    <ModuleSchedules>
        <Schedule Identifier="1" Name="schedule" MajorFrame="1.5">
            <PartitionSchedule PartitionIdentifier="0">
                <Window Offset="0" Duration="0.5" />
            </PartitionSchedule>
            <PartitionSchedule PartitionIdentifier="1">
                <Window Offset="0.5" Duration="0.5" />
            </PartitionSchedule>
            <PartitionSchedule PartitionIdentifier="2">
                <Window Offset="1" Duration="0.5" />
            </PartitionSchedule>
        </Schedule>
    </ModuleSchedules>

    <ModuleChannels>
        <Channel Identifier="0" Name="iop_to_sender" QoS="128kb">
            <Source>
                <StandardPartition PartitionIdentifier="0" />
            </Source>
            <Destination>
                <StandardPartition PartitionIdentifier="1" />
            </Destination>
        </Channel>
        <Channel Identifier="1" Name="sender_to_iop" QoS="256kb">
            <Source>
                <StandardPartition PartitionIdentifier="1" />
            </Source>
```

```
                    <Destination>
                        <StandardPartition PartitionIdentifier="2" />
                    </Destination>
                </Channel>
                <Channel Identifier="2" Name="to_out" QoS="256kb">
                    <Source>
                        <StandardPartition PartitionIdentifier="2" />
                    </Source>
                    <Destination>
                        <StandardPartition PartitionIdentifier="0" />
                    </Destination>
                </Channel>
            </ModuleChannels>

            <ModuleMemoryMap>
                <PartitionMemoryMap PartitionIdentifier="0">
                    <Region Identifier="0" Size="1MB" />
                    <Region Identifier="1" Size="1MB" />
                </PartitionMemoryMap>

            <!-- Shared memory for use with ARM-BBB - cpsw -->
                 <SharedMemoryRegion Identifier="10" Name="cpsw"
    PhysicalAddress="CPSW">
                    <Access PartitionIdentifier="0" Permissions="RW"
    CacheAttributes="DEVICE"/>
                </SharedMemoryRegion>

            </ModuleMemoryMap>

        </XKYModule>

```

D.2 partion.xml

D.2.1 partition_0

```
            <?xml version="1.0" encoding="UTF-8" standalone="no"?>
            <Partition Identifier="0" Name="p0">

                <BuildConfiguration>
                    <Personality Name="IOP">
                        <IOPartition>
                            <!-- Configuration -->
```

```
8                       <Configuration RequestCount="64" TimeToLive="5"
    />

10                      <!-- Physical Devices configuration  -->
                        <PhysicalDevices>
12                        <Device Id="0" Device="ETH0" Ip="192.168.1.112"
     MAC="78:a5:04:c0:da:80" TXD="32" RXD="32"/>
                        </PhysicalDevices>
14
                        <Routes>
16                        <Route Id="1" PortId="to_out" Device="ETH0"
    Direction="OUT">
                              <Header Ip="192.168.1.111" MAC="d0:bf:9c
    :89:2b:67" Port="5000"/>
18                        </Route>
                          <Route Id="2" PortId="iop_to_sender" Device="
    ETH0" Direction="IN">
20                            <Header Ip="192.168.1.117" MAC="d0:bf:9c
    :89:2b:67" Port="14000"/>
                          </Route>
22                      </Routes>

24                      <!-- Schedules -->
                        <Schedules>
26                        <Schedule ScheduleIdentifier="1">
                            <DevicesConfiguration>
28                            <Device DeviceId="0" Reads="5"/>
                            </DevicesConfiguration>
30                          <RoutesConfiguration>
                              <Route RouteId="1" Active="true"/>
32                            <Route RouteId="2" Active="true"/>
                            </RoutesConfiguration>
34                        </Schedule>
                        </Schedules>
36                    </IOPartition>
                  </Personality>
38                <Libraries>
                      <Library Name="LIBXKY" Options="" />
40                    <Library Name="LIBPRINTF" Options="" />
                  </Libraries>
42              </BuildConfiguration>
                <Ports>
44                  <QueuingPort ChannelIdentifier="0" Name="iop_to_sender"
     Direction="SOURCE" MaxMessageSize="256" MaxNbMessages="32" />
                    <QueuingPort ChannelIdentifier="2" Name="to_out"
    Direction="DESTINATION" MaxMessageSize="256" MaxNbMessages="32" />
46              </Ports>
```

```
            <MemoryMap>
48              <Region Identifier="0" Name="code" Permissions="RX"
    CacheAttributes="COPY_BACK" />
                <Region Identifier="1" Name="data" Permissions="RW"
    CacheAttributes="WRITE_THROUGH" />
50
        <!-- Shared memory for use with ARM-BBB - cpsw -->
52          <SharedRegion Identifier="10"/>

54      <!-- Shared memory for use with PPC - etsec port 2 -->
          <!--  <SharedRegion Identifier="10"/>
56          <SharedRegion Identifier="11"/>
            <SharedRegion Identifier="12"/>       -->
58      </MemoryMap>

60      </Partition>
```

D.2.2 partition_1

```
        /**
2       * @file
        * @author pfnf
4       * @brief Partition 1 example code
        */
6
       #include <xky.h>
8      #include <bare.h>
       #include <xky_printf.h>
10
       void entry_point(void) {
12         int rb;
           xky_sz_t m_size;
14         xky_time_t m_time;
           xky_u32_t i, m_status;
16         xky_id_t my_id, port_id;
           xky_port_status_t port_status;
18
           my_id = xky_syscall_get_partition_id(NULL);
20         port_id = xky_syscall_get_port_id("sender_to_iop");
           xky_syscall_get_port_status(port_id, &port_status);
22

24         xky_printf(
                   "                    Port id: %i\n"
```

```
26                      "                Port status:\n"
                        "                     Type - %i\n"
28                      "                Direction - %i\n"
                        " Max. Number of Messages - %i\n"
30                      "       Max. Message size - %i\n"
                        "          Refresh Period - %lli\n"
32                      "       Messages in queue - %i\n\n",
                        port_id,
34                      port_status.type,
                        port_status.configuration.direction,
36                      port_status.configuration.max_nb_message,
                        port_status.configuration.max_message_size,
38                      port_status.configuration.refresh_period,
                        port_status.nb_message);
40
                xky_id_t send_id = xky_syscall_get_port_id("to_out");
42              if (send_id < 0) {
                    xky_printf(" error %i getting sampling port id\n",
    send_id);
44              }
                while (1) {
46
                    char buffer[256] = { 0 };
48

50                      m_size = xky_syscall_read_port(port_id, buffer, &
    m_time, &m_status);
                        if (m_size > 0) {
52                          xky_printf("Chegou %i: %s (%i) (%lli)", my_id,
    buffer, m_size, m_time);
                            xky_printf("\n");
54                          rb = xky_syscall_write_port(send_id, buffer,
    strlen(buffer));
                            }
56

58                  bare_wake_in_next_mtf();

60

                }
62
                /* this code will never execute */
64              xky_printf("I shouldn't be running\n");
            }
66
```

D.2.3 partition_2

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Partition Identifier="2" Name="p2">
    <BuildConfiguration>
        <Personality Name="BARE">
            <BareConfiguration EntryPoint="entry_point"
CodeMemoryRegion="0" DataMemoryRegion="1" />
        </Personality>
        <Libraries>
            <Library Name="LIBXKY" Options="" />
            <Library Name="LIBPRINTF" Options="" />
        </Libraries>
    </BuildConfiguration>

    <Ports>
        <QueuingPort ChannelIdentifier="1" Name="sender_to_iop"
 Direction="DESTINATION" MaxMessageSize="256" MaxNbMessages="8" />
        <QueuingPort ChannelIdentifier="2" Name="to_out"
Direction="SOURCE" MaxMessageSize="256" MaxNbMessages="8" />
    </Ports>

    <MemoryMap>
        <Region Identifier="0" Name="code" Permissions="RX"
CacheAttributes="COPY_BACK" />
        <Region Identifier="1" Name="data" Permissions="RW"
CacheAttributes="WRITE_THROUGH" />
    </MemoryMap>
</Partition>
```

D.3 main.c

D.3.1 partition_1

```
/**
* @file
* @author pfnf
* @brief Example 4 - Partition 1
*/

#include <xky.h>
#include <bare.h>
#include <string.h>
#include <xky_printf.h>
```

```
12          /**
             * @brief Partition entry point
14           */
            void entry_point() {
16              xky_id_t my_id, port_id;
                xky_port_status_t port_status;
18              my_id = xky_syscall_get_partition_id(NULL);
                port_id = xky_syscall_get_port_id("port_0");
20              xky_syscall_get_port_status(port_id, &port_status);

22              /* create recv port */
                xky_id_t recv_id = xky_syscall_get_port_id("iop_to_sender");
24              if (recv_id < 0) {
                    xky_printf(" error %i getting queuing port id\n",
    recv_id);
26              }

28              /* create SEND port */
                xky_id_t send_id = xky_syscall_get_port_id("sender_to_iop");
30              if (send_id < 0) {
                    xky_printf(" error %i getting sampling port id\n",
    send_id);
32              }

34              while (1) {

36                  int rc;
                    xky_time_t time_stamp;
38                  xky_u32_t status;
                    char message[256] = { 0 };
40                  char buffer[256] = { 0 };


42


44                  /* receive */
                    do {
46                      rc = xky_syscall_read_port(recv_id, message, &
    time_stamp, &status);
                        if (rc > 0) {
48                          xky_printf(" Arduino mandou: %s %li\n", message,
     time_stamp);
                            xky_printf("\n");
50                          strcpy(buffer,message);
                            xky_syscall_write_port(send_id, buffer, strlen(
    buffer));
52
                        }
```

```
54                  }
                    while (rc > 0);
56
58
                    bare_wake_in_next_mtf();
60              }
            }
62
```

D.3.2 partition_2

```
            /**
2           * @file
            * @author pfnf
4           * @brief Partition 1 example code
            */
6
           #include <xky.h>
8          #include <bare.h>
           #include <xky_printf.h>
10
           void entry_point(void) {
12             int rb;
               xky_sz_t m_size;
14             xky_time_t m_time;
               xky_u32_t i, m_status;
16             xky_id_t my_id, port_id;
               xky_port_status_t port_status;
18
               my_id = xky_syscall_get_partition_id(NULL);
20             port_id = xky_syscall_get_port_id("sender_to_iop");
               xky_syscall_get_port_status(port_id, &port_status);
22
24             xky_printf(
                       "                    Port id: %i\n"
26                     "               Port status:\n"
                       "                     Type - %i\n"
28                     "                Direction - %i\n"
                       " Max. Number of Messages - %i\n"
30                     "       Max. Message size - %i\n"
                       "           Refresh Period - %lli\n"
32                     "       Messages in queue - %i\n\n",
```

```
                    port_id,
                    port_status.type,
                    port_status.configuration.direction,
                    port_status.configuration.max_nb_message,
                    port_status.configuration.max_message_size,
                    port_status.configuration.refresh_period,
                    port_status.nb_message);

            xky_id_t send_id = xky_syscall_get_port_id("to_out");
            if (send_id < 0) {
                xky_printf(" error %i getting sampling port id\n",
send_id);
            }
            while (1) {

                char buffer[256] = { 0 };


                    m_size = xky_syscall_read_port(port_id, buffer, &
m_time, &m_status);
                    if (m_size > 0) {
                        xky_printf("Chegou %i: %s (%i) (%lli)", my_id,
buffer, m_size, m_time);
                        xky_printf("\n");
                        rb = xky_syscall_write_port(send_id, buffer,
strlen(buffer));
                        }


                bare_wake_in_next_mtf();


            }

            /* this code will never execute */
            xky_printf("I shouldn't be running\n");
        }

```

D.4 processing

```
//Thanks to Adrian Fernandez
//Communication updates by M.Furkan Bahat November 2014
//For more information http://mfurkanbahat.blogspot.com.tr/
```

```
4
        import processing.serial.*;
6
        //Para UDP
8       import hypermedia.net.*;

10      //Para serial
        //import cc.arduino.*;
12
        int W=1400; //My Laptop's screen width
14      int H=700;  //My Laptop's screen height
        float Pitch;
16      float Bank;
        float Azimuth;
18      float ArtificialHoizonMagnificationFactor=0.7;
        float CompassMagnificationFactor=0.85;
20      float SpanAngle=120;
        int NumberOfScaleMajorDivisions;
22      int NumberOfScaleMinorDivisions;
        PVector v1, v2;
24


26      Serial port;
        float Phi;    //Dimensional axis
28      float Theta;
        float Psi;
30


32      UDP udp;  // define the UDP object


34
        void setup()
36      {
          //size(W, H);
38        size(1400, 700);
          rectMode(CENTER);
40        smooth();
          strokeCap(SQUARE);//Optional
42
        //  println(Serial.list()); //Shows your connected serial ports
44      //  port = new Serial(this, Serial.list()[0], 115200);
          //Up there you should select port which arduino connected and
    same baud rate.
46      //  port.bufferUntil('\n');

48        udp = new UDP( this, 6000 );
          udp.log( true );     // <-- printout the connection activity
```

```
50          udp.listen( true );

52        }
          void draw()
54        {
            background(0);
56          translate(W/4, H/2.1);
            MakeAnglesDependentOnMPU6050();
58          Horizon();
            rotate(-Bank);
60          PitchScale();
            Axis();
62          rotate(Bank);
            Borders();
64          Plane();
            ShowAngles();
66          Compass();
            ShowAzimuth();
68        }

70        void receive( byte[] data, String ip, int port ) {  // <-- extended
     handler

72
            // get the "real" message =
74          // forget the ";\n" at the end <-- !!! only for a communication
    with Pd !!!
            data = subset(data, 0, data.length-2);
76          String message = new String( data );

78          // print the result
            println( "receive: \""+message+"\" from "+ip+" on port "+port );
80
            String input = message;
82
              if(input != null){
84           input = trim(input);
            String[] values = split(input, ",");
86          if(values.length == 3){
            float phi = float(values[0]);
88          float theta = float(values[1]);
            float psi = float(values[2]);
90          print(phi);
            print(theta);
92          println(psi);
            Phi = phi;
94          Theta = theta;
```

```
        Psi = psi;
96       }
        }
98    }

100   void serialEvent(Serial port) //Reading the datas by Processing.
      {
102      String input = port.readStringUntil('\n');
        if(input != null){
104      input = trim(input);
        String[] values = split(input, " ");
106     if(values.length == 3){
        float phi = float(values[0]);
108     float theta = float(values[1]);
        float psi = float(values[2]);
110     print(phi);
        print(theta);
112     println(psi);
        Phi = phi;
114     Theta = theta;
        Psi = psi;
116      }
        }
118   }
      void MakeAnglesDependentOnMPU6050()
120   {
        Bank =Phi;
122     Pitch=Theta;
        Azimuth=Psi;
124   }
      void Horizon()
126   {
        scale(ArtificialHoizonMagnificationFactor);
128     noStroke();
        fill(0, 180, 255);
130     rect(0, -100, 900, 1000);
        fill(95, 55, 40);
132     rotate(-Bank);
        rect(0, 400+Pitch, 900, 800);
134     rotate(Bank);
        rotate(-PI-PI/6);
136     SpanAngle=120;
        NumberOfScaleMajorDivisions=12;
138     NumberOfScaleMinorDivisions=24;
        CircularScale();
140     rotate(PI+PI/6);
        rotate(-PI/6);
```

```
142             CircularScale();
                rotate(PI/6);
144           }
              void ShowAzimuth()
146           {
                fill(50);
148             noStroke();
                rect(20, 470, 440, 50);
150             int Azimuth1=round(Azimuth);
                textAlign(CORNER);
152             textSize(35);
                fill(255);
154             text("Azimuth:  "+Azimuth1+" Deg", 80, 477, 500, 60);
                textSize(40);
156             fill(25,25,150);
                text("UFSCar / Gustavo Tamanaka :-) ", -350, 477, 500, 60);
158           }
              void Compass()
160           {
                translate(2*W/3, 0);
162             scale(CompassMagnificationFactor);
                noFill();
164             stroke(100);
                strokeWeight(80);
166             ellipse(0, 0, 750, 750);
                strokeWeight(50);
168             stroke(50);
                fill(0, 0, 40);
170             ellipse(0, 0, 610, 610);
                for (int k=255;k>0;k=k-5)
172             {
                  noStroke();
174               fill(0, 0, 255-k);
                  ellipse(0, 0, 2*k, 2*k);
176             }
                strokeWeight(20);
178             NumberOfScaleMajorDivisions=18;
                NumberOfScaleMinorDivisions=36;
180             SpanAngle=180;
                CircularScale();
182             rotate(PI);
                SpanAngle=180;
184             CircularScale();
                rotate(-PI);
186             fill(255);
                textSize(60);
188             textAlign(CENTER);
```

```
    text("W", -375, 0, 100, 80);
    text("E", 370, 0, 100, 80);
    text("N", 0, -365, 100, 80);
    text("S", 0, 375, 100, 80);
    textSize(30);
    text("COMPASS", 0, -130, 500, 80);
    rotate(PI/4);
    textSize(40);
    text("NW", -370, 0, 100, 50);
    text("SE", 365, 0, 100, 50);
    text("NE", 0, -355, 100, 50);
    text("SW", 0, 365, 100, 50);
    rotate(-PI/4);
    CompassPointer();
  }
  void CompassPointer()
  {
    rotate(PI+radians(Azimuth));
    stroke(0);
    strokeWeight(4);
    fill(100, 255, 100);
    triangle(-20, -210, 20, -210, 0, 270);
    triangle(-15, 210, 15, 210, 0, 270);
    ellipse(0, 0, 45, 45);
    fill(0, 0, 50);
    noStroke();
    ellipse(0, 0, 10, 10);
    triangle(-20, -213, 20, -213, 0, -190);
    triangle(-15, -215, 15, -215, 0, -200);
    rotate(-PI-radians(Azimuth));
  }
  void Plane()
  {
    fill(0);
    strokeWeight(1);
    stroke(0, 255, 0);
    triangle(-20, 0, 20, 0, 0, 25);
    rect(110, 0, 140, 20);
    rect(-110, 0, 140, 20);
  }
  void CircularScale()
  {
    float GaugeWidth=800;
    textSize(GaugeWidth/30);
    float StrokeWidth=1;
    float an;
    float DivxPhasorCloser;
```

```
236         float DivxPhasorDistal;
            float DivyPhasorCloser;
238         float DivyPhasorDistal;
            strokeWeight(2*StrokeWidth);
240         stroke(255);
            float DivCloserPhasorLenght=GaugeWidth/2-GaugeWidth/9-StrokeWidth
    ;
242         float DivDistalPhasorLenght=GaugeWidth/2-GaugeWidth/7.5-
    StrokeWidth;
            for (int Division=0;Division<NumberOfScaleMinorDivisions+1;
    Division++)
244         {
              an=SpanAngle/2+Division*SpanAngle/NumberOfScaleMinorDivisions;
246           DivxPhasorCloser=DivCloserPhasorLenght*cos(radians(an));
              DivxPhasorDistal=DivDistalPhasorLenght*cos(radians(an));
248           DivyPhasorCloser=DivCloserPhasorLenght*sin(radians(an));
              DivyPhasorDistal=DivDistalPhasorLenght*sin(radians(an));
250           line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
    DivyPhasorDistal);
            }
252         DivCloserPhasorLenght=GaugeWidth/2-GaugeWidth/10-StrokeWidth;
            DivDistalPhasorLenght=GaugeWidth/2-GaugeWidth/7.4-StrokeWidth;
254         for (int Division=0;Division<NumberOfScaleMajorDivisions+1;
    Division++)
            {
256           an=SpanAngle/2+Division*SpanAngle/NumberOfScaleMajorDivisions;
              DivxPhasorCloser=DivCloserPhasorLenght*cos(radians(an));
258           DivxPhasorDistal=DivDistalPhasorLenght*cos(radians(an));
              DivyPhasorCloser=DivCloserPhasorLenght*sin(radians(an));
260           DivyPhasorDistal=DivDistalPhasorLenght*sin(radians(an));
              if (Division==NumberOfScaleMajorDivisions/2|Division==0|
    Division==NumberOfScaleMajorDivisions)
262           {
                strokeWeight(15);
264             stroke(0);
                line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
    DivyPhasorDistal);
266             strokeWeight(8);
                stroke(100, 255, 100);
268             line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
    DivyPhasorDistal);
              }
270           else
              {
272             strokeWeight(3);
                stroke(255);
```

```
274             line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
    DivyPhasorDistal);
              }
276         }
        }
278     void Axis()
        {
280       stroke(255, 0, 0);
          strokeWeight(3);
282       line(-115, 0, 115, 0);
          line(0, 280, 0, -280);
284       fill(100, 255, 100);
          stroke(0);
286       triangle(0, -285, -10, -255, 10, -255);
          triangle(0, 285, -10, 255, 10, 255);
288     }
        void ShowAngles()
290     {
          textSize(30);
292       fill(50);
          noStroke();
294       rect(-150, 400, 280, 40);
          rect(150, 400, 280, 40);
296       fill(255);
          Pitch=Pitch/5;
298       int Pitch1=round(Pitch);
          text("Pitch:  "+Pitch1+" Deg", -20, 411, 500, 60);
300       text("Roll:  "+Bank+" Deg", 280, 411, 500, 60);
        }
302     void Borders()
        {
304       noFill();
          stroke(0);
306       strokeWeight(400);
          rect(0, 0, 1100, 1100);
308       strokeWeight(200);
          ellipse(0, 0, 1000, 1000);
310       fill(0);
          noStroke();
312       rect(4*W/5, 0, W, 2*H);
          rect(-4*W/5, 0, W, 2*H);
314     }
        void PitchScale()
316     {
          stroke(255);
318       fill(255);
          strokeWeight(3);
```

```
320             textSize(24);
                textAlign(CENTER);
322             for (int i=-4;i<5;i++)
                {
324               if ((i==0)==false)
                  {
326                 line(110, 50*i, -110, 50*i);
                  }
328               text(""+i*10, 140, 50*i, 100, 30);
                  text(""+i*10, -140, 50*i, 100, 30);
330             }
                textAlign(CORNER);
332             strokeWeight(2);
                for (int i=-9;i<10;i++)
334             {
                  if ((i==0)==false)
336               {
                    line(25, 25*i, -25, 25*i);
338               }
                }
340           }
```

D.5 Arduino

D.5.1 Servo

```
2         #include <SPI.h>          // needed for Arduino versions later than
      0018
          #include <Ethernet.h>
4         #include <EthernetUdp.h>          // UDP library from: bjoern@cs.
      stanford.edu 12/30/2008
          #include <Servo.h>
6
          #define SS_SD_CARD   4
8         #define SS_ETHERNET 10

10        Servo myservo;
          void scCardCode() {
12            // ...
              digitalWrite(SS_SD_CARD, LOW);  // SD Card ACTIVE
14            // code that sends to the sd card slave device over SPI
              // using SPI.transfer() etc.
16            digitalWrite(SS_SD_CARD, HIGH); // SD Card not active
```

```
        // ...
    }

    void ethernetCode() {
        // ...
        digitalWrite(SS_ETHERNET, LOW);  // Ethernet ACTIVE
        // code that sends to the ethernet slave device over SPI
        // using SPI.transfer() etc.
        //digitalWrite(SS_ETHERNET, HIGH); // Ethernet not active
        // ...
    }

    // Enter a MAC address and IP address for your controller below.
    // The IP address will be dependent on your local network:
    byte mac[] = {
    0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEC
    };
    IPAddress ip(192, 168, 1, 116);

    unsigned int localPort = 5000 ;      // local port to listen on

    // buffers for receiving and sending data
    char packetBuffer[UDP_TX_PACKET_MAX_SIZE];  //buffer to hold
incoming packet,
    char  ReplyBuffer[] = "Oi Aroca, receber ta funcionando!";       //
 a string to send back
    int pos = 0;
    // An EthernetUDP instance to let us send and receive packets over
UDP
    EthernetUDP Udp;

    void setup() {

        // ...
        pinMode(SS_SD_CARD, OUTPUT);
        pinMode(SS_ETHERNET, OUTPUT);
        digitalWrite(SS_SD_CARD, HIGH);  // SD Card not active
        digitalWrite(SS_ETHERNET, HIGH); // Ethernet not active
        // ...
        myservo.attach(9);
    // start the Ethernet and UDP:
    Serial.begin(9600);


        // start the Ethernet connection and the server:
    Ethernet.begin(mac, ip);
    //  Ethernet.begin(mac);
```

```
62            Udp.begin(localPort);

64            Serial.print("Meu servidor ");
        Serial.println(Ethernet.localIP());
66


68        }

70        void loop() {
        // if there's data available, read a packet
72        int packetSize = Udp.parsePacket();
        if (packetSize) {
74            Serial.print("Received packet of size ");
            Serial.println(packetSize);
76            Serial.print("From ");
            IPAddress remote = Udp.remoteIP();
78            for (int i = 0; i < 4; i++) {
            Serial.print(remote[i], DEC);
80            if (i < 3) {
                Serial.print(".");
82            }
            }
84            Serial.print(", port ");
            Serial.println(Udp.remotePort());
86
            // read the packet into packetBufffer
88            Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
            Serial.println("Contents:");
90            Serial.println(packetBuffer);
            pos = strok (packetBuffer,",")
92            myservo.write(pos);
        }
94        delay(10);
        }
96
```

D.5.2 IMU

```
      #include <SPI.h>         // needed for Arduino versions later than
  0018
2     #include <Ethernet.h>
      #include <EthernetUdp.h>         // UDP library from: bjoern@cs.
  stanford.edu 12/30/2008
```

```
4       #include <Servo.h>

6       #define SS_SD_CARD   4
        #define SS_ETHERNET 10
8
        Servo myservo;
10      void scCardCode() {
            // ...
12          digitalWrite(SS_SD_CARD, LOW);  // SD Card ACTIVE
            // code that sends to the sd card slave device over SPI
14          // using SPI.transfer() etc.
            digitalWrite(SS_SD_CARD, HIGH); // SD Card not active
16          // ...
        }
18
        void ethernetCode() {
20          // ...
            digitalWrite(SS_ETHERNET, LOW);  // Ethernet ACTIVE
22          // code that sends to the ethernet slave device over SPI
            // using SPI.transfer() etc.
24          //digitalWrite(SS_ETHERNET, HIGH); // Ethernet not active
            // ...
26      }

28      // Enter a MAC address and IP address for your controller below.
        // The IP address will be dependent on your local network:
30      byte mac[] = {
          0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEC
32      };
        IPAddress ip(192, 168, 1, 117);
34
        unsigned int localPort = 5000 ;      // local port to listen on
36
        // buffers for receiving and sending data
38      char packetBuffer[UDP_TX_PACKET_MAX_SIZE];  //buffer to hold
    incoming packet,
        int pos = 0;
40      // An EthernetUDP instance to let us send and receive packets over
    UDP
        EthernetUDP Udp;
42      // Uncomment the below line to use this axis definition:
           // X axis pointing forward
44         // Y axis pointing to the right
           // and Z axis pointing down.
46      // Positive pitch : nose up
        // Positive roll : right wing down
48      // Positive yaw : clockwise
```

```
    int SENSOR_SIGN[9] = {1,1,1,-1,-1,-1,1,1,1}; //Correct directions x
,y,z - gyro, accelerometer, magnetometer
50  // Uncomment the below line to use this axis definition:
      // X axis pointing forward
52    // Y axis pointing to the left
      // and Z axis pointing up.
54  // Positive pitch : nose down
    // Positive roll : right wing down
56  // Positive yaw : counterclockwise
    //int SENSOR_SIGN[9] = {1,-1,-1,-1,1,1,1,-1,-1}; //Correct
directions x,y,z - gyro, accelerometer, magnetometer
58
    // tested with Arduino Uno with ATmega328 and Arduino Duemilanove
with ATMega168
60
    #include <Wire.h>
62
    // accelerometer: 8 g sensitivity
64  // 3.9 mg/digit; 1 g = 256
    #define GRAVITY 256  //this equivalent to 1G in the raw data coming
 from the accelerometer
66
    #define ToRad(x) ((x)*0.01745329252)  // *pi/180
68  #define ToDeg(x) ((x)*57.2957795131)  // *180/pi

70  // gyro: 2000 dps full scale
    // 70 mdps/digit; 1 dps = 0.07
72  #define Gyro_Gain_X 0.07 //X axis Gyro gain
    #define Gyro_Gain_Y 0.07 //Y axis Gyro gain
74  #define Gyro_Gain_Z 0.07 //Z axis Gyro gain
    #define Gyro_Scaled_X(x) ((x)*ToRad(Gyro_Gain_X)) //Return the
scaled ADC raw data of the gyro in radians for second
76  #define Gyro_Scaled_Y(x) ((x)*ToRad(Gyro_Gain_Y)) //Return the
scaled ADC raw data of the gyro in radians for second
    #define Gyro_Scaled_Z(x) ((x)*ToRad(Gyro_Gain_Z)) //Return the
scaled ADC raw data of the gyro in radians for second
78
    // LSM303/LIS3MDL magnetometer calibration constants; use the
Calibrate example from
80  // the Pololu LSM303 or LIS3MDL library to find the right values
for your board

82  #define M_X_MIN -1000
    #define M_Y_MIN -1000
84  #define M_Z_MIN -1000
    #define M_X_MAX +1000
86  #define M_Y_MAX +1000
```

```
    #define M_Z_MAX +1000
88
    #define Kp_ROLLPITCH 0.02
90  #define Ki_ROLLPITCH 0.00002
    #define Kp_YAW 1.2
92  #define Ki_YAW 0.00002

94  /*For debugging purposes*/
    //OUTPUTMODE=1 will print the corrected data,
96  //OUTPUTMODE=0 will print uncorrected data of the gyros (with drift
    )
    #define OUTPUTMODE 1
98
    #define PRINT_DCM 0     //Will print the whole direction cosine
    matrix
100 #define PRINT_ANALOGS 0 //Will print the analog raw data
    #define PRINT_EULER 1   //Will print the Euler angles Roll, Pitch
    and Yaw
102
    #define STATUS_LED 13
104
    float G_Dt=0.02;    // Integration time (DCM algorithm)  We will
    run the integration loop at 50Hz if possible
106
    long timer=0;   //general purpuse timer
108 long timer_old;
    long timer24=0; //Second timer used to print values
110 int AN[6]; //array that stores the gyro and accelerometer data
    int AN_OFFSET[6]={0,0,0,0,0,0}; //Array that stores the Offset of
    the sensors
112
    int gyro_x;
114 int gyro_y;
    int gyro_z;
116 int accel_x;
    int accel_y;
118 int accel_z;
    int magnetom_x;
120 int magnetom_y;
    int magnetom_z;
122 float c_magnetom_x;
    float c_magnetom_y;
124 float c_magnetom_z;
    float MAG_Heading;
126
    float Accel_Vector[3]= {0,0,0}; //Store the acceleration in a
    vector
```

```
128         float Gyro_Vector[3]= {0,0,0};//Store the gyros turn rate in a
    vector
            float Omega_Vector[3]= {0,0,0}; //Corrected Gyro_Vector data
130         float Omega_P[3]= {0,0,0};//Omega Proportional correction
            float Omega_I[3]= {0,0,0};//Omega Integrator
132         float Omega[3]= {0,0,0};

134         // Euler angles
            float roll;
136         float pitch;
            float yaw;
138
            float errorRollPitch[3]= {0,0,0};
140         float errorYaw[3]= {0,0,0};

142         unsigned int counter=0;
            byte gyro_sat=0;
144
            float DCM_Matrix[3][3]= {
146           {
                1,0,0  }
148           ,{
                0,1,0  }
150           ,{
                0,0,1  }
152         };
            float Update_Matrix[3][3]={{0,1,2},{3,4,5},{6,7,8}}; //Gyros here
154


156         float Temporary_Matrix[3][3]={
              {
158             0,0,0  }
              ,{
160             0,0,0  }
              ,{
162             0,0,0  }
            };
164         void setup() {

166             // ...
              pinMode(SS_SD_CARD, OUTPUT);
168           pinMode(SS_ETHERNET, OUTPUT);
              digitalWrite(SS_SD_CARD, HIGH);  // SD Card not active
170           digitalWrite(SS_ETHERNET, HIGH); // Ethernet not active
              // ...
172
            // start the Ethernet and UDP:
```

```
174          Serial.begin(9600);
             digitalWrite(STATUS_LED,LOW);
176           delay(1500);

178             // start the Ethernet connection and the server:
             Ethernet.begin(mac, ip);
180         //  Ethernet.begin(mac);

182             Udp.begin(localPort);

184               Serial.print("Meu servidor ");
             Serial.println(Ethernet.localIP());
186         Accel_Init();
             Compass_Init();
188          Gyro_Init();

190          delay(20);

192          for(int i=0;i<32;i++)     // We take some readings...
               {
194            Read_Gyro();
               Read_Accel();
196            for(int y=0; y<6; y++)   // Cumulate values
                 AN_OFFSET[y] += AN[y];
198            delay(20);
               }
200
             for(int y=0; y<6; y++)
202            AN_OFFSET[y] = AN_OFFSET[y]/32;

204          AN_OFFSET[5]-=GRAVITY*SENSOR_SIGN[5];

206          //Serial.println("Offset:");
             for(int y=0; y<6; y++)
208            Serial.println(AN_OFFSET[y]);

210          delay(2000);
             digitalWrite(STATUS_LED,HIGH);
212
             timer=millis();
214          delay(20);
             counter=0;
216
           }
218
           void loop() {
220           if((millis()-timer)>=20)  // Main loop runs at 50Hz
```

```
        {
222       counter++;
          timer_old = timer;
224       timer=millis();
          if (timer>timer_old)
226       {
            G_Dt = (timer-timer_old)/1000.0;    // Real time of loop run.
    We use this on the DCM algorithm (gyro integration time)
228         if (G_Dt > 0.2)
              G_Dt = 0; // ignore integration times over 200 ms
230       }
          else
232         G_Dt = 0;


234


236       // *** DCM algorithm
          // Data adquisition
238       Read_Gyro();   // This read gyro data
          Read_Accel();     // Read I2C accelerometer
240
          if (counter > 5)  // Read compass data at 10Hz... (5 loop runs)
242       {
            counter=0;
244         Read_Compass();     // Read I2C magnetometer
            Compass_Heading(); // Calculate magnetic heading
246       }

248       // Calculations...
          Matrix_update();
250       Normalize();
          Drift_correction();
252       Euler_angles();
          // ***
254       ReplyBuffer = string(pitch) +","+ string(row) +","+string(yaw)
          printdata();
256     }

258     //send a reply to the IP address and port that sent us the packet
    we received
          IPAddress destinationIP(192, 168, 1, 112);
260       Udp.beginPacket(destinationIP, 14000);
          Udp.write(ReplyBuffer);
262       Udp.endPacket();
        }
264     delay(10);
      }
```

266