

UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA MECÂNICA

TRABALHO DE CONCLUSÃO DE CURSO

**Interface para Teleoperação de Braço Robótico Usando
Movimentos do Braço Humano**

Autor:

Vinicio BAZAN PINTO
FERNANDES

Orientador:

Prof. Dr. Glauco AUGUSTO
DE PAULA CAURIN

9 de novembro de 2015

Dedicatória

Dedico esse trabalho aos meus familiares e amigos, que me deram o suporte para estar de pé durante a longa caminhada.

Agradecimentos

Gostaria de começar agradecendo a Deus por iluminar meu caminho e me dar sanidade para levar este trabalho até o fim.

Agradeço à minha família e amigos, que foram essenciais para que eu me motivasse e me mantivesse firme não só durante a execução deste trabalho, como durante toda a minha graduação. O apoio incondicional de cada um deles me fez sempre ter em mente o que sou sou, onde estou e onde quero chegar.

Um agradecimento mais que especial fica para os grandes apoiadores e orientadores do projeto: o Professor Dr. Glauco Augusto de Paula Caurin, por sua amizade, suporte e ensinamentos nas etapas do meu aprendizado em que um tutor e amigo foram imprescindíveis; igualmente agradeço a Guilherme Fernandes, da Toradex Brasil, que foi como meu segundo orientador, me ajudando a desvendar as novidades do equipamento usado, me dando inspiração para alcançar novas ideias e, sobretudo, por ser um grande pilar motivacional e um exemplo a ser seguido. O Professor Dr. Daniel Varela Magalhães foi outra pessoa de grande importância para se completar esse trabalho, dando um grande suporte de conhecimento.

Não menos importantes são meus companheiros de laboratório, o estudante de PhD e Professor Kleber de Oliveira Andrade, o Mestrando Henrique Borges Garcia e o graduando Leonardo José Consoni, os quais tiveram sempre boas ideias para me ajudar quando precisei de opiniões e ideias de solução.

Agradeço ainda a Isabela Baldim, do Zaená Oficina Criativa, que não só ofereceu soluções engenhosas para a parte física do projeto, como foi uma grande apoiadora, motivando-me a sempre seguir em frente.

Apesar de não terem contato direto com esse trabalho, meus companheiros de laboratório nos EUA, no Mechatronics Laboratory da Polytechnic Institute of New York University, Vikram Kapila, que foi meu orientador; Jared A. Frank, David Lopez, Valentin Siderskyi e Giancarlo Gramazio, companheiros de trabalho, foram essenciais para o surgimento e desenvolvimento da primeira ideia atrelada a esse projeto.

Por fim, gostaria de agradecer à Universidade de São Paulo, mais especificamente à Escola de Engenharia de São Carlos, e a todos seus funcionários, que direta ou indiretamente contribuíram para essa realização.

Epígrafe

"You look at where you're going and where you are and it never makes sense, but then you look back at where you've been and a pattern seems to emerge."

Robert M. Pirsig, Zen e a Arte da Manutenção de Motocicletas

Resumo

Este relatório tem por objetivo apresentar em detalhes os conceitos e ferramentas usados para o desenvolvimento do projeto de conclusão de curso de Engenharia Mecatrônica, pela Escola de Engenharia de São Carlos, Universidade de São Paulo (USP). O projeto se baseia numa interface de captura dos movimentos do braço de um usuário, de forma a usar as informações de posicionamento espacial deste como maneira de controlar a posição da garra de um braço robótico. O projeto se baseia em dois conceitos fundamentais: intuitividade e teleoperação. É visado prover ao usuário uma maneira fácil de controlar o robô, sem necessitar que ele passe por um processo longo de aprendizado, como ocorre com a programação de robôs e uso de painéis de controle específicos. A metáfora de operação usada é baseada nos próprios movimentos do braço do usuário. O robô busca posicionar seu *end-effector* de maneira que sua trajetória seja comparável ao movimento do braço do usuário seja. Valendo-se de uma transmissão de dados sem fio, é possível teleoperar o braço robótico sem que o operador precise compartilhar o mesmo ambiente que o robô. O desenvolvimento desse sistema passa pelo estudo da geometria e cinemática do braço humano e do robô utilizado; pelo uso de sensores inteligentes, no caso, sensores iniciais; pela utilização de um computador em módulo a fim de produzir um sistema embarcado de tempo real; processamento de dados e controle de um braço robótico.

Sumário

1	Introdução	7
1.1	Motivação	7
1.2	Objetivo	8
2	Hardware e Componentes	9
2.1	Sensores Inerciais - IMU	9
2.1.1	Composição e funcionamento	9
2.1.2	Aplicação no Projeto	10
2.1.3	Interface de Transmissão de Dados	11
2.1.4	Alimentação para o Sensor Inercial	12
2.2	Computador Embarcado	13
2.2.1	Colibri T30	13
2.2.2	Iris	14
2.3	Braço robótico	15
2.4	Esquema de Hardware	17
3	Sistema e Software	18
3.1	Overview - Processamento e Mapeamento	18
3.2	Conexão com o computador	18
3.2.1	Windows Mobile Device Center	18
3.2.2	Remote Display	19
3.3	Visual Studio 2008	19
3.3.1	Instalando SDKs do Windows CE	20
3.3.2	Primeiro projeto C++	21
3.3.3	Primeiro projeto C#	24
3.3.4	Acessando a aplicação no Colibri	25
3.4	Windows Embedded Compact 7	26
3.4.1	Dispositivos Embarcados	26
3.4.2	Sistemas Operacionais de Tempo-Real	26
3.4.3	Prioridades e Agendamento Round-Robin	27
3.4.4	Gerando uma nova imagem do Windows Embedded Compact	28
3.4.5	Restaurando a imagem de fábrica	28
3.5	Bibliotecas e Funcionalidades dos módulos Toradex	29
3.5.1	Bluetooth	29
3.6	Comunicação Serial via UART	32
3.6.1	Pinos UART na Iris	32
3.7	Comunicação Wireless via TCP/IP	34
3.7.1	Usando Connectify Hotspot para criar uma rede WiFi	35
3.7.2	Sockets	36
3.7.3	Winsock	37

3.7.4	Estabelecendo a comunicação	37
3.8	Fluxograma do Software	38
3.9	Teleoperação	39
3.9.1	Mapeamento do espaço de trabalho do usuário para o espaço do robô	40
3.9.2	Animação do Robô SCARA 7545 em MATLAB	41
4	Resultados e Discussão	46
5	Próximas Etapas	47
6	Conclusão	48
Apêndice A	Cinemática Direta do Braço Humano	49
Apêndice B	Firmware e Programação da IMU Razor 9Dof	52
B.1	Upload do Firmware para o sensor	52
B.2	Animação gráfica em Processing	55
B.3	Calibrando os sensores internos	56
B.3.1	Calibrando o acelerômetro	56
B.3.2	Calibrando o Giroscópio	58
B.3.3	Calibrando o Magnetômetro	58
Apêndice C	Configuração do Modem Bluetooth Mate Silver	60
Apêndice D	Stack Bluetooth do Windows Embedded Compact	64

1 Introdução

1.1 Motivação

A prática da teleoperação de sistemas robóticos está presente em diversos âmbitos na atualidade, seja na área médica, com a utilização de robôs para a realização de cirurgias, como na área de segurança, com o desarme de bombas à distância. A capacidade de se operar um braço robótico e decidir a trajetória de sua garra traz ao ser humano mobilidade e domínio sobre sistemas robustos. Contudo, ainda hoje, várias metáforas utilizadas para a operação de manipuladores robóticos é fraca no que diz respeito a trazer ao operador uma maneira natural e de rápido aprendizado de se realizar as operações desejadas. Nesse tocante há grande interesse em se desenvolver uma metáfora de operação que seja tão simples e natural como se movimentar.

Em um projeto similar realizado anteriormente no *Mechatronics and Controls Laboratory* da *Polytechnic School of Engineering of NYU* [1], onde uma interface de igual intuito à deste projeto foi usada, realizou-se experimentos que mostraram que, comparado com outras metáforas de teleoperação de robôs tidas como intuitivas - controles de Videogame e aplicações em Tablet e Smartphones - a metáfora de uso dos movimentos do braço humano obteve maior grau de sucesso e maior performance do usuário com pouca ou nenhuma experiência na operação de um manipulador robótico.

Dessa forma, o projeto é motivado pela crescente universalização do conceito de intuitividade. Em vários aspectos do nosso cotidiano nos deparamos com aplicativos, aparelhos e objetos que simplesmente não precisam de manual de operação. Com uma curva de aprendizado bem reduzida, na maioria dos casos basta ao usuário alguns minutos ou horas para ter total controle sobre as ações do aplicativo ou dispositivo. Mais além, ao se ter uma interface que se acople ao braço humano, o usuário fica com suas mãos livres e sente menos influência do equipamento sobre seu corpo, como peso, inércia, entre outros. Assim, reduz-se o desconforto e/ou fracasso na utilização devido a influências externas sobre o usuário.

Como mencionado no início, este projeto tem por motivação principal criar um equipamento que possa ser utilizado em situações onde a interação homem-máquina (na maioria dos casos um robô manipulador) é necessária. Por outro lado, dado o fato de que, para a aplicação presente, o movimento do braço humano necessita ser sensoriado e mapeado, há ainda aplicações na área de reabilitação de pacientes que perderam total ou parcialmente os movimentos do braço e na terapia para pessoas com dificuldade de movimentação. Uma área, em específico, de crescente interesse é a de Reabilitação Robótica, que se vale de robôs para auxiliar o paciente na correção e realização de movimentos com o membro debilitado. Mais além, o equipamento pode ser aplicado em tarefas de mapeamento do braço humano para a criação de movimentos virtuais em jogos, animações e softwares.

1.2 Objetivo

De maneira simplificada, o braço humano possui 7 Graus de Liberdade (GdL), respectivamente, 3 eixos de rotação ortogonais no ombro, que funciona como uma junta esférica, um eixo de rotação no cotovelo e 3 rotações do pulso: uma rotação do antebraço que possibilita o movimento de pronação-supinação; o movimento de flexão-extensão e desvios ulnar e radial.

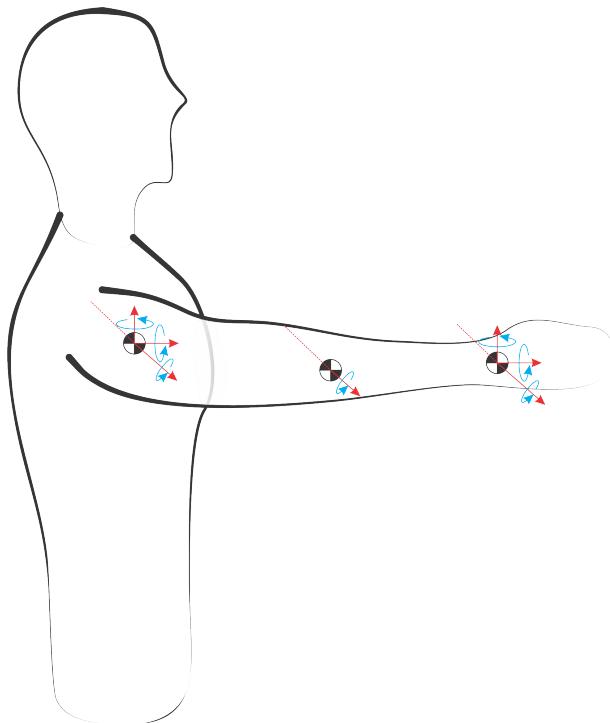


Figura 1: Os 7 Graus de Liberdade Simplificados do Braço Humano.

O objetivo do projeto é desenvolver uma interface que, valendo-se de sensores inerciais, meça os 7 GdL referidos e envie tais dados para um computador embarcado rodando um sistema operacional de tempo real, o qual processa tais dados, a fim de calcular, usando a cinemática direta do braço humano A, a posição cartesiana do pulso do usuário e usar tal informação, via mapeamento de coordenadas, para controlar a posição cartesiana da garra do robô SCARA. Para tal, os sensores inerciais são acoplados ao braço do operador por meio de braceletes.

Os detalhes de componentes utilizados para o projeto e as camadas do mesmo são expostas em maiores detalhes nas seções a seguir.

2 Hardware e Componentes

2.1 Sensores Iniciais - IMU

2.1.1 Composição e funcionamento

Um sensor inercial, ou Inertial Measurement Unit (IMU), é um tipo de sensor usado para medições em 3 eixos ortogonais de posicionamento angular, tornando possível o cálculo da orientação espacial, em 3 dimensões, do mesmo. É possível obter outras medidas, ainda, como aceleração linear, orientação magnética, entre outros, dependendo das características construtivas do sensor e do seu software de processamento de dados. Um sensor inercial geralmente consiste de uma combinação de 3 sensores: um acelerômetro 3-eixos, um giroscópio 3-eixos e um magnetômetro 3-eixos. O sensor inercial escolhido inicialmente para este projeto é o Razor 9DoF, vendido pela empresa norteamericana SparkFun. Este sensor possui as características mencionadas anteriormente, bem como um processador integrado, ATmega328, para processamento interno dos dados de medição, que são transmitidos via interface serial. Por possuir esse processador, o sensor inercial é tratado como um *Arduino Pro ou Pro Mini de 3.3V com ATmega 328 e clock de 8MHz*. Seus três sensores integrados são: ITG-3200 (giroscópio MEMS 3-eixos), ADXL345 (acelerômetro 3-eixos), and HMC5883L (magnetômetro 3-eixos).

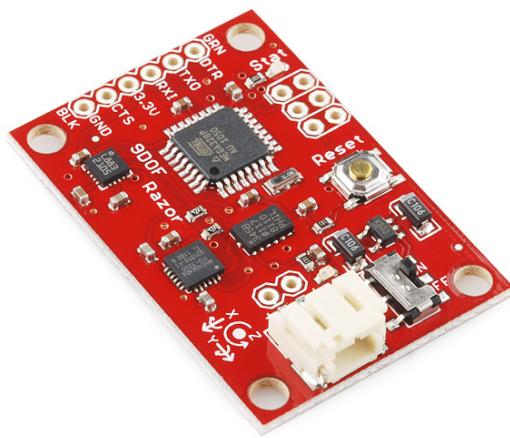


Figura 2: Sensor Inercial Razor 9Dof.

O acelerômetro é responsável por medir acelerações. Com seus 3 eixos ortogonais de medição, é possível medir acelerações lineares em cada um deles. É ainda possível se medir a componente da gravidade atuando em dois de seus eixos, quando é apenas inclinado, sem movimento linear. Assim se obtém a inclinação do sensor.

O giroscópio é um sensor que mede velocidade angular em seus eixos de rotação.

O magnetômetro é um sensor orientado pelo campo magnético da Terra e portando fornece uma orientação espacial semelhante à de uma bússola. Esse sensor é, contudo, suscetível a interferências de outros objetos geradores de campo magnético.

A junção desses três sensores em um só encapsulamento forma, como já mencionado, o sensor inercial. Utiliza-se tal estratégia para medição de posicionamento espacial pois cada um desses três sensores é capaz de fornecer informações úteis para o propósito em questão, mas sozinhos sofrem com ruído, imprecisão de medição devido a instabilidade de orientação, entre outros fatores. Dessa forma, realiza-se a fusão dos dados dos três sensores via software, a qual passa por um processo de filtragem. Assim obtém-se dados bastante precisos de posicionamento espacial, com menor variação por interferência de fontes externas e robustos ao longo do tempo. Há diferentes séries de sensores inerciais disponíveis no mercado, desde os mais simples, que apresentam simplesmente os dados originais de cada um dos três componentes, como os avançados que já possuem software de fusão e filtro, bem como processador interno e comunicação com dispositivos externos. Naturalmente, quanto maior a complexidade de tais sensores, maior o preço.

2.1.2 Aplicação no Projeto

Para o presente projeto, visamos empregar três sensores inerciais, posicionados no braço, antebraço e mão, respectivamente, de modo a medir todos os 7 GdL do braço humano (mais detalhes adiante). O emprego de três desses sensores fornece uma medição com redundância. Em outras palavras, como cada sensor inercial (IMU) mede orientação em três direções ortogonais, calcula-se 9 GdL, que serão usados para um ajuste mais fino do cálculo dos 7 GdL visados.

Cada IMU é usada para medir, portanto, os ângulos de Yaw, Pitch, Roll em sua orientação atual. A fim de evitar o problema de Gimbal Lock [22] inerente a essa medição, vários sensores presentes no mercado fornecem a medição em quatérnios, que são vetores de 4 dimensões, com componentes w, x, y, z , que representam um vetor espacial que é o eixo de rotação momentâneo e o ângulo de rotação em torno desse eixo. Um quatérnio é representado por

$$q = q_w + iq_x + jq_y + kq_z$$

e a correção desses parâmetros com os ângulos de Yaw, Pitch e Roll é realizada através das seguintes equações:

$$Roll = atan2(2(q_wq_x + q_yq_z), 1 - 2(q_x^2 + q_y^2)) \quad (1)$$

$$Pitch = arcsin(2(q_wq_y - q_zq_x)) \quad (2)$$

$$Yaw = \text{atan}2(2(q_x q_z + q_y q_z), 1 - 2(q_y^2 + q_z^2)) \quad (3)$$

O sensor inercial Razor 9DoF é bastante versátil, sendo vendido a um custo reduzido, se comparado a outros sensores iniciais do mercado. Por possuir um microprocessador embarcado, existe um firmware que é programado em sua memória. A programação do sensor e os procedimentos de calibração e transmissão de dados foram feitos segundo [20], através do qual se mostra como programar o firmware, usar interfaces em Arduino, Processing, Matlab, C++ e Android. No firmware do sensor, é possível se escolher vários parâmetros de utilização, como por exemplo a saída de dados em forma de texto ou binária.

Um ponto bastante importante sobre esse sensor é que seu software já realiza a compensação do *drift* na leitura inicial do valor de Yaw. Mais além, o firmware é programado para a linguagem de programação Arduino, possuindo uma demonstração gráfica em Processing, código C++ para aquisição de dados (compilado para Linux, Unix e MacOsx) e Android (para comunicação por Bluetooth).

2.1.3 Interface de Transmissão de Dados

O sensor Razor 9DoF possui um conjunto de pinos FTDI para transmissão de dados. Esses pinos são: GND, CTS, 3.3V, RX, TX e DTR. Dessa forma, é possível usar o pino TX para transmissão serial de dados. A programação do microprocessador do sensor é feita usando os pinos FTDI mencionados e para tal, necessita-se um cabo ou adaptador FTDI. Esse adaptador possui um chip FT232 embarcado que faz o papel de converter os sinais seriais da USB para TTL (lógica de CI da IMU).

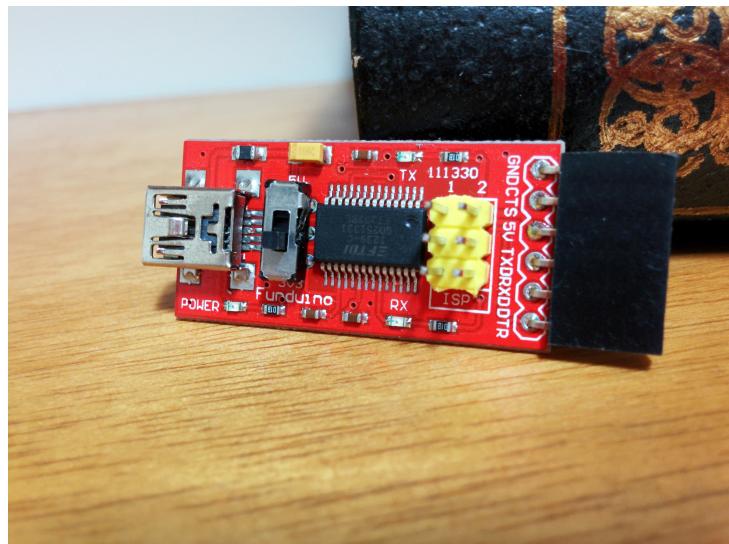


Figura 3: Adaptador USB-FTDI 5V/3.3V.

Esse adaptador, portanto, permite a conexão entre o sensor inercial e um PC. Porém, se se deseja uma conexão sem fio, é possível também utilizar um modem Bluetooth para tal finalidade. O modem *Bluetooth Mate Silver* possui os mesmos seis pinos de comunicação do sensor inercial. É facilmente programado, como mostrado no Apêndice C. Basicamente na programação do modem bluetooth, realizada via terminal serial como *TeraTerm* [26], configura-se basicamente a *Baud Rate* de transmissão de dados, o modo de conexão com outro dispositivo e endereço ao qual o dispositivo deve se conectar.



Figura 4: Modem Bluetooth - Bluetooth Mate Silver.

2.1.4 Alimentação para o Sensor Inercial

O sensor inercial opera em 3.3V e portanto requer alimentação nesse nível de tensão. Ele pode ser alimentado através dos pinos GND e 3.3V da sequência de pinos FTDI ou pelo conector de bateria. Para esse efeito, uma solução viável é usar uma bateria recarregável de Lítio, juntamente com um carregador USB para essa bateria. A bateria escolhida para o projeto é uma bateria *LiPoly* de 3.7V em 850mAh encontrada em [27]. O carregador possui conexão USB para conectar ao computador, bem como um jack para conexão com uma fonte de parede. Possui também a conexão para a bateria e a saída para a conexão com o sensor inercial. O carregador é encontrado em [28].

A composição dos elementos citados anteriormente forma um conjunto IMU-Bateria-Carregador-Bluetooth que permite a fácil transmissão de dados sem fio, como mostrado na Figura 5

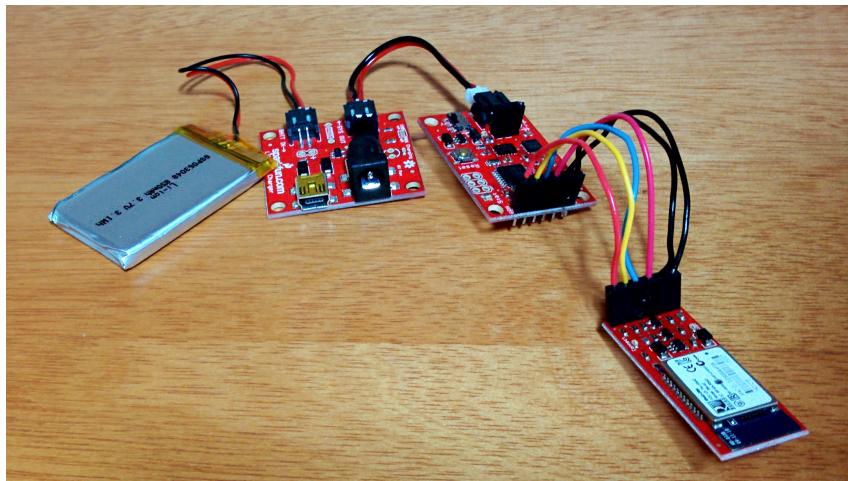


Figura 5: Conjunto IMU-Bateria-Carregador-Bluetooth.

2.2 Computador Embarcado

2.2.1 Colibri T30

O processamento principal de dados do projeto é realizado com um computador em módulo Colibri T30, montado em uma placa de suporte Iris, ambos da empresa Toradex [2]. O Colibri T30 apresenta um processador NVIDIA®Tegra™3 Quad-Core de arquitetura ARM Cortex™-A9. Esse módulo apresenta o sistema operacional Windows Embedded Compact 7, tendo suporte também para Linux e Android.



Figura 6: Toradex Colibri T30.

2.2.2 Iris

O Colibri T30 é montado em uma placa de suporte Iris, também produzida pela Toradex, que possui interfaces de entrada e saída para aumentar a versatilidade do projeto. A placa possui, entre outras propriedades, pinos GPIO, interface para chip GSM, conector Ethernet, pinos de comunicação serial RS232, 1 porta USB de alta velocidade, saída de vídeo VGA/DVI, jack de áudio, e entrada de alimentação 6-27 Volts DC.

Com tais características é possível conectar dispositivos externos à placa de desenvolvimento de modo a adicionar funcionalidades ao projeto. Em primeira instância foi planejado conectar um modem bluetooth para a comunicação com os modems bluetooth dos sensores inerciais. Porém, devido a dificuldades que surgiram em adequar a imagem do sistema operacional a conter as DLLs e arquivos necessários para o stack Bluetooth funcionar, procedeu-se com o uso dos pinos seriais, relacionados a portas UART, para a aquisição de dados. Assim, os dados podem ser adquiridos pelo computador em módulo e processados internamente, de modo a enviar os dados finais de posicionamento ao robô a ser teleoperado.

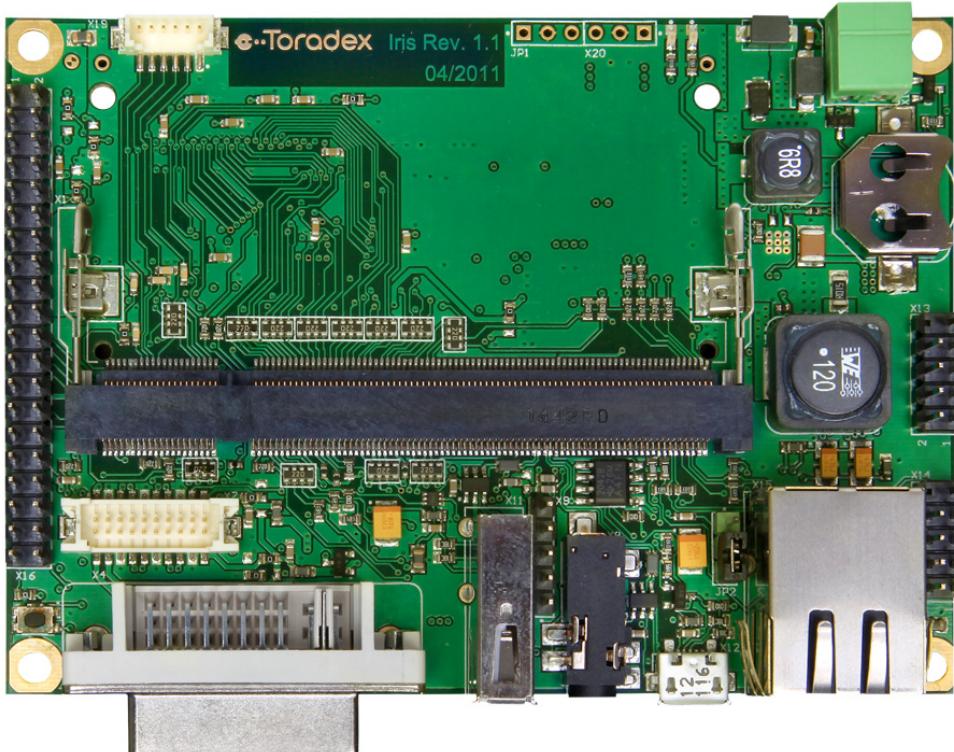


Figura 7: Placa de Suporte Iris.

2.3 Braço robótico

O braço robótico escolhido para o projeto é um SCARA (Selective Compliance Assembly Robot Arm ou Selective Compliance Articulated Robot Arm). Este robô é produzido pela IBM, modelo 7545, e apresentado na figura 8. Ele possui três eixos de rotação e um prismático, permitindo-o alcançar um grande envelope no espaço, de maneira mais rápida que um robô cartesiano.



Figura 8: Robô SCARA 7545 da IBM

Este é um robô que pode ser robustamente usado com controle de posição, de modo a atingir posições desejadas com precisão. Um esquemático do robô é mostrado na figura 9.

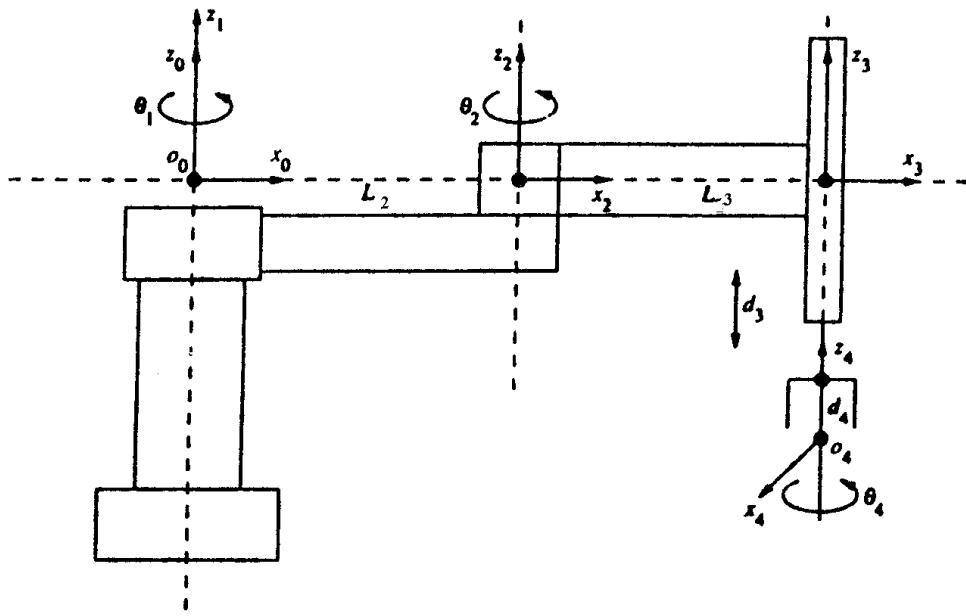


Figura 9: Esquemático de um robô SCARA.

Como mencionado anteriormente, o SCARA 7545 pode atingir um grande envelope de trabalho no espaço, como mostrado na Figura 10. Isso é possível devido ao alcance de cada uma de suas juntas. A primeira junta rotacional (base) tem alcance de 0° até 200° ; a segunda, de 0° a 135° ; a junta prismática pode alcançar de $0mm$ a $200mm$, enquanto a junta final de rotação, que move apenas a ferramenta, alcança entre -180° até 180°

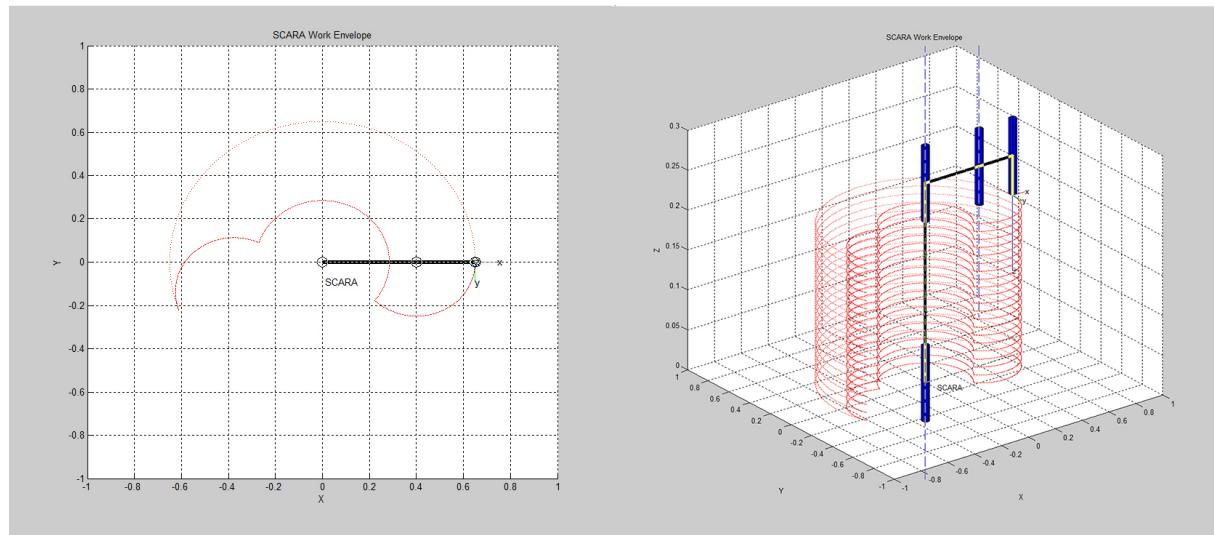


Figura 10: Evelope de trabalho do robô SCARA 7545. Medidas em metros.

2.4 Esquema de Hardware

Agregando o hardware e componentes descritos acima, a Figura 11 ilustra como os componentes do hardware interagem entre si.

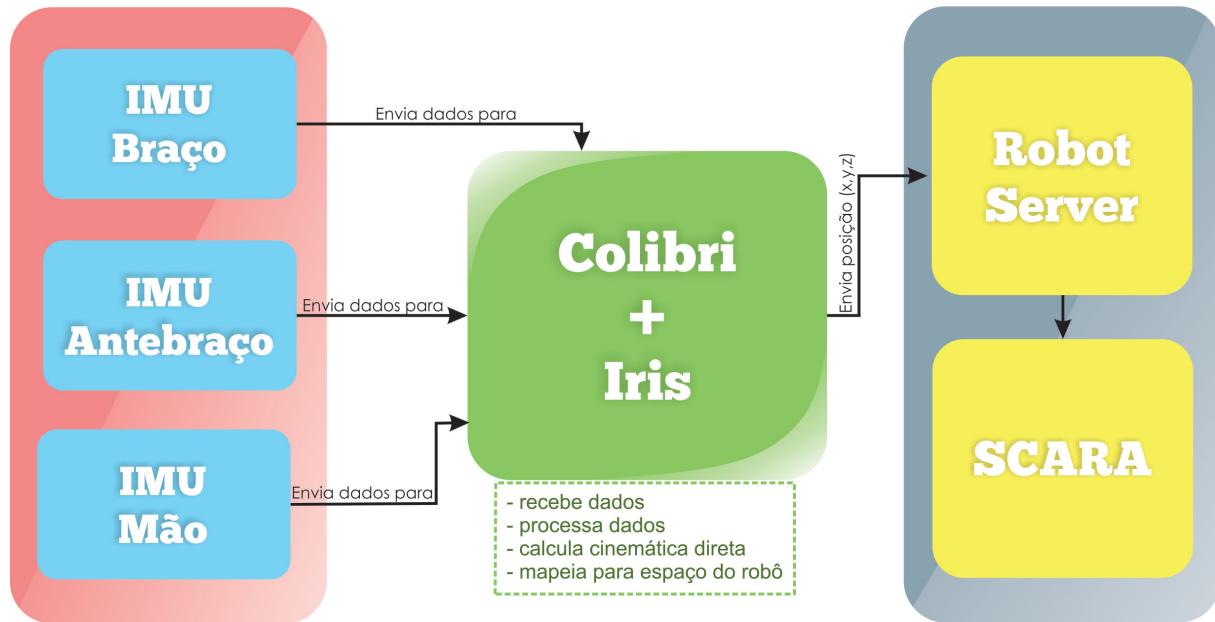


Figura 11: Diagrama ilustrativo das interações entre os componentes do hardware.

3 Sistema e Software

3.1 Overview - Processamento e Mapeamento

Os dados de orientação espacial de cada seção do braço do operador não servem diretamente como dados de controle de posição do robô teleoperado. Portanto, faz-se necessário o uso da cinemática direta para, dados os ângulos de rotação de cada junta do braço humano, calcular a posição cartesiana em 3 dimensões do pulso do operador. O processo matemático que envolve esse passo está contido no Apêndice A. A partir daí, é necessário realizar um mapeamento do espaço atingível pelo operador para o espaço atingível pelo robô. Um mapeamento linear é realizado. Com esses dados calculados, é possível enviá-los, via Ethernet, para o controlador do robô, que recebe como dados de entrada a posição cartesiana espacial da garra do mesmo. Esse controlador calcula, por sua vez a cinemática reversa para os ângulos de rotação de cada junta do robô. Os esquema de fluxo de dados é mostrado a seguir.



Figura 12: Diagrama de Blocos: fluxo de dados.

3.2 Conexão com o computador

Antes de iniciar a programação de aplicações para o Colibri, é necessário conectá-lo a um PC e proceder com a instalação dos drivers necessários.

3.2.1 Windows Mobile Device Center

O *Windows Mobile Device Center* é um software que gerencia a conexão de dispositivos que operam com sistemas Windows embarcados. É necessário ter esse software instalado para que seja possível para o PC comunicar-se com o Colibri. O Colibri, acoplado à Iris, deve ser conectado ao PC via cabo USB. Na Iris, deve se conectar a saída microUSB à uma porta USB do PC. Ao fazer isso, o driver do Colibri é instalado e o *Windows Mobile Device Center* também, caso ainda não esteja instalado no PC. Com isso pronto, o módulo pode começar a se comunicar com o PC através da IDE de programação *Visual Studio*.

Figura 13: Painel do Windows Mobile Device Center

3.2.2 Remote Display

Uma alternativa ao uso de um cabo VGA ou DVI para conectar o conjunto Iris-Colibri a um monitor é usar o software *Remote Display* [17] para mostrar a tela do SO do Colibri na tela do PC. Para esse fim, além do Remote Display, é necessário que o módulo esteja conectado ao computador por meio de um cabo USB-MicroUSB.

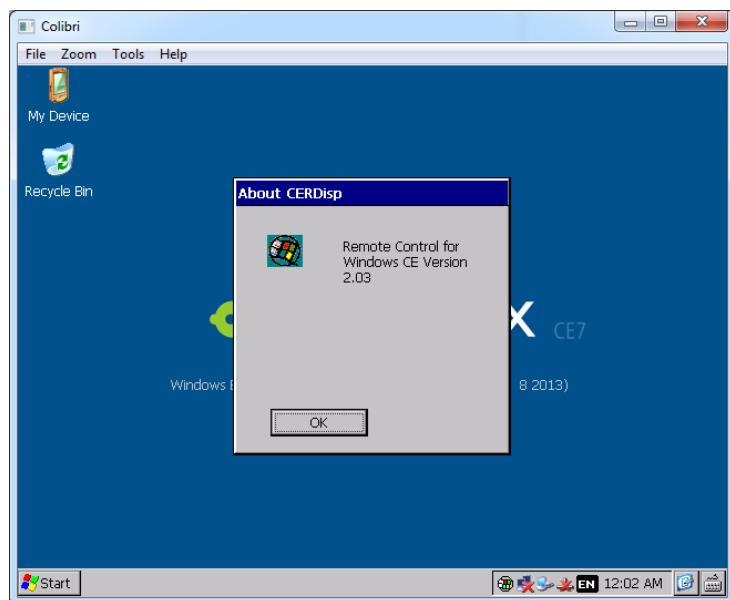


Figura 14: Usando o Remote Display

3.3 Visual Studio 2008

O *Visual Studio 2008* é uma IDE com diversos recursos para o desenvolvimento de aplicações de software. A versão 2008 é a usada para a programação de aplicações para os módulos da Toradex.

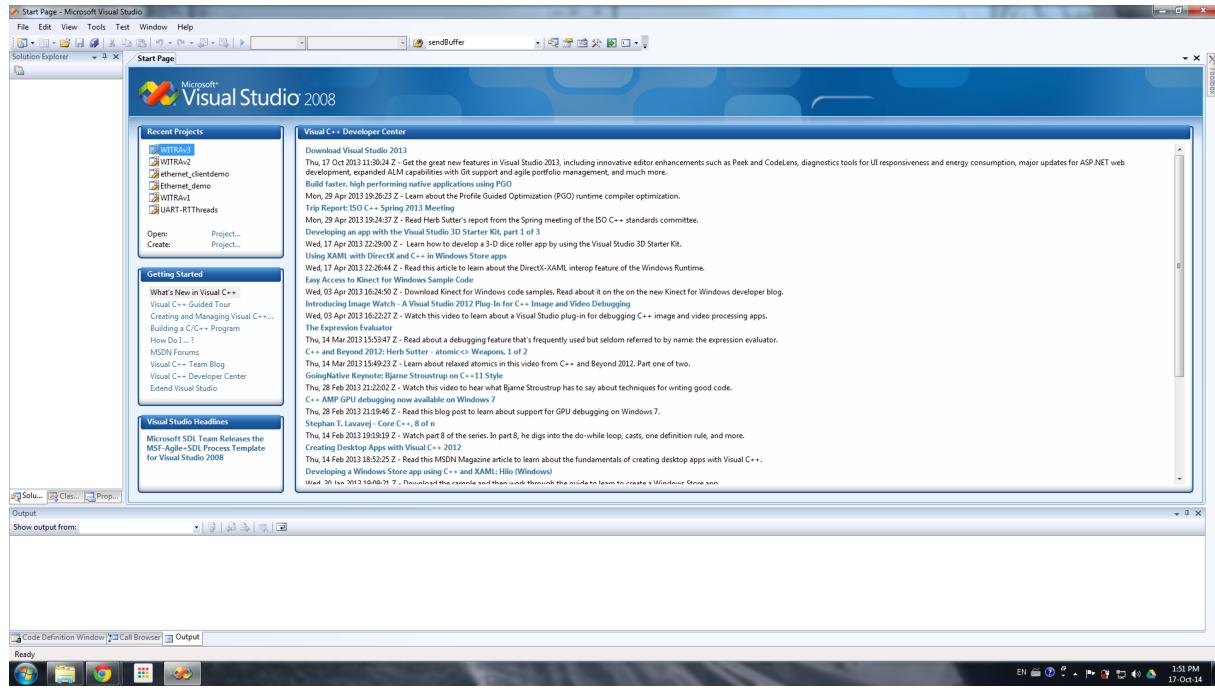


Figura 15: Interface do Visual Studio 2008

3.3.1 Instalando SDKs do Windows CE

Antes de se iniciar um novo projeto de aplicação para o módulo Colibri, é necessário que se tenha instalado, no computador no qual se usará o Visual Studio, os SDKs (Software Development Kit) referentes ao Windows CE, mais especificamente, os fornecidos pela Toradex [2] em [6]. Deve-se instalar pelo menos o SDK para Windows CE 6, mais o SDK referente à versão do sistema operacional instalado no Colibri. No nosso caso, o sistema operacional usado é o Windows Embedded Compact 7 e, portanto, também o SDK deste deve ser instalado.

É importante frisar que, para se instalar os SDKs, a IDE *Visual Studio 2008* deve estar instalada no PC. Com isso, pode-se prosseguir com a instalação, primeiramente do SDK para Windows CE 6.0. Para instalá-lo, deve-se seguir os seguintes passos:

- Visual Studio 2008 deve estar instalado e atualizado. Feche-o antes da instalação;
- Baixe o SDK para Windows CE 6.0;
- Comece o processo de instalação;
- Selecione *Custom Installation*. A instalação da documentação leva a uma parada inesperada do instalado. Portanto, marque a documentação como *Entire feature will be unavailable*;

- Termine o processo de instalação.

Para instalar o SDK do Windows Embedded Compact 7, o processo é mais simples:

- Instale as atualizações do *Visual Studio 2008* específicas para Windows Embedded Compact 7: [29], [30];
- Feche o *Visual Studio 2008* antes da instalação;
- Baixe o SDK da Toradex para Windows Embedded Compact 7;
- Execute o instalador.

Com os SDKs instalados, pode-se começar a programar aplicações para Windows CE usando o *Visual Studio 2008*.

3.3.2 Primeiro projeto C++

Para iniciar um novo projeto C++ no *Visual Studio 2008*, deve-se seguir os seguintes passos:

1. Abrir o *Visual Studio 2008*;
2. No menu superior, deve-se clicar em *File*, então em *New* e em *Project*. Uma caixa de diálogo de *New Project* aparece;

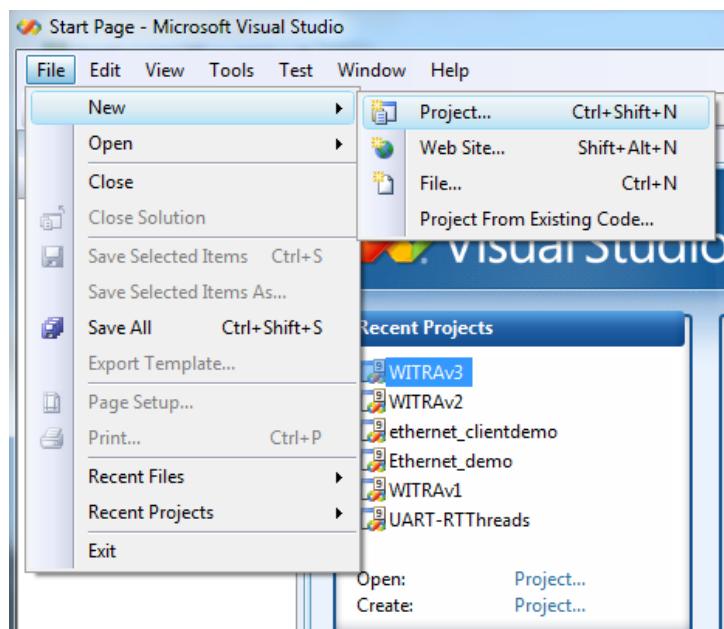


Figura 16: Criando um novo projeto no VS2008

3. Na área de *Project types*, deve-se clicar em *Visual C++* e então em *Smart Device*;

4. Na lista *Templates* à direita, deve-se selecionar *Win32 Smart Device Project*;
5. Na página seguinte, adicionamos os SDKs para compilar o programa. Deve-se adicionar Colibri600 e/ou SDKwince7;
6. Por fim, na página seguinte escolhe-se *Console application* e *Empty project* e então *Finish*. A partir daí, basta adicionar os arquivos *.cpp* e/ou *.h* necessários.

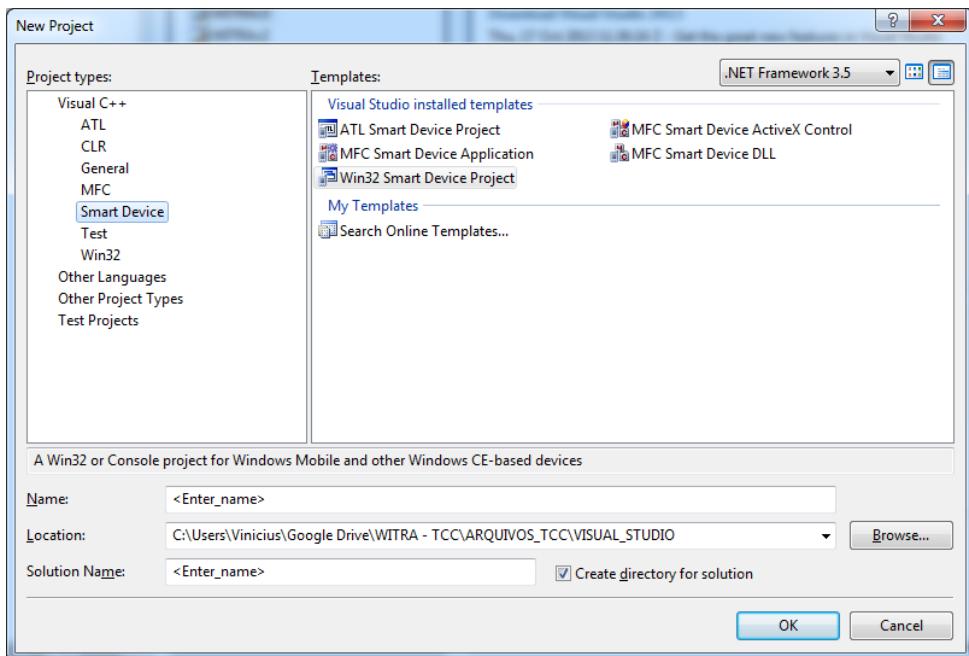


Figura 17: Criando um novo projeto no VS2008 - Nome do Projeto

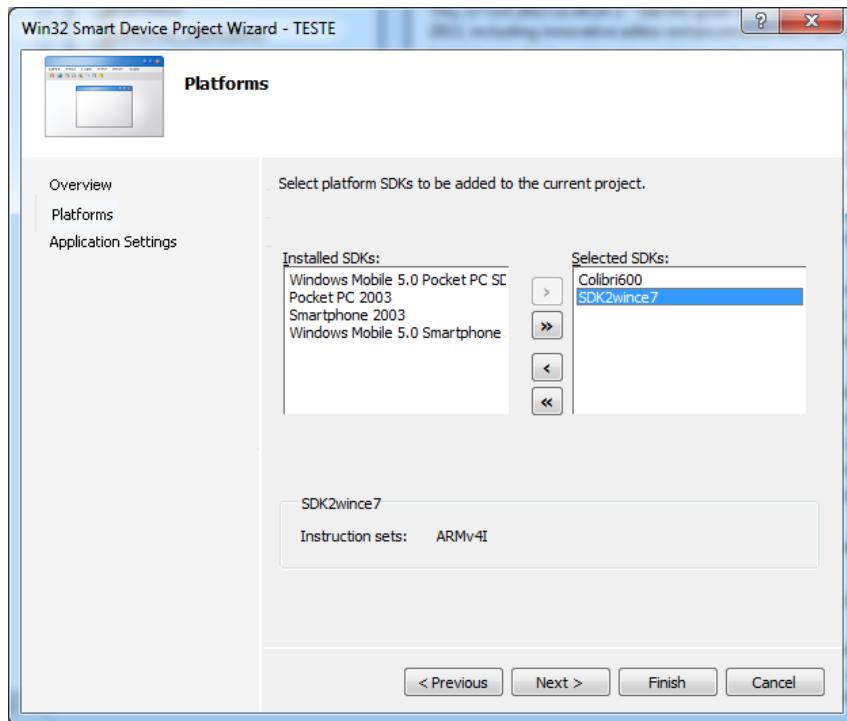


Figura 18: Criando um novo projeto no VS2008 - Incluindo SDKs

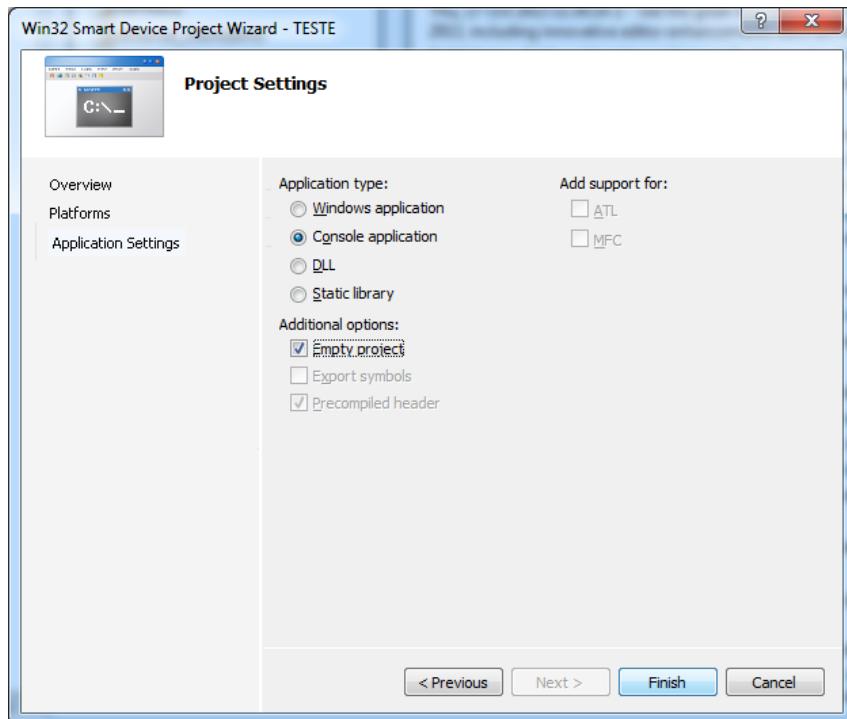


Figura 19: Criando um novo projeto no VS2008 - Finalizando

Para compilar o programa, basta clicar em no menu *Build* e então em *Build Solution*. Para enviar o programa para o módulo, deve-se clicar no menu *Build* e então em *Deploy Solution*.

3.3.3 Primeiro projeto C#

Criar um novo projeto C# é bastante parecido com o projeto C++. Para tal deve-se seguir os seguintes passos:

1. Abrir o *Visual Studio 2008*;
2. No menu superior, deve-se clicar em *File*, então em *New* e em *Project*. Uma caixa de diálogo de *New Project* aparece. Vide Figura 16;
3. Na área de *Project types*, deve-se clicar em *Other Languages*, *Visual C#* e então em *Smart Device*, como mostrado na Figura 20;
4. Na janela que aparece, deve-se ter *Target Platform* como **Windows CE**, *.NET Compact Framework version* como **.NET Compact Framework Version 3.5** (o qual deve estar instalado no PC) e deve-se selecionar **Device Application** em *Templates*, como mostra a Figura 21.

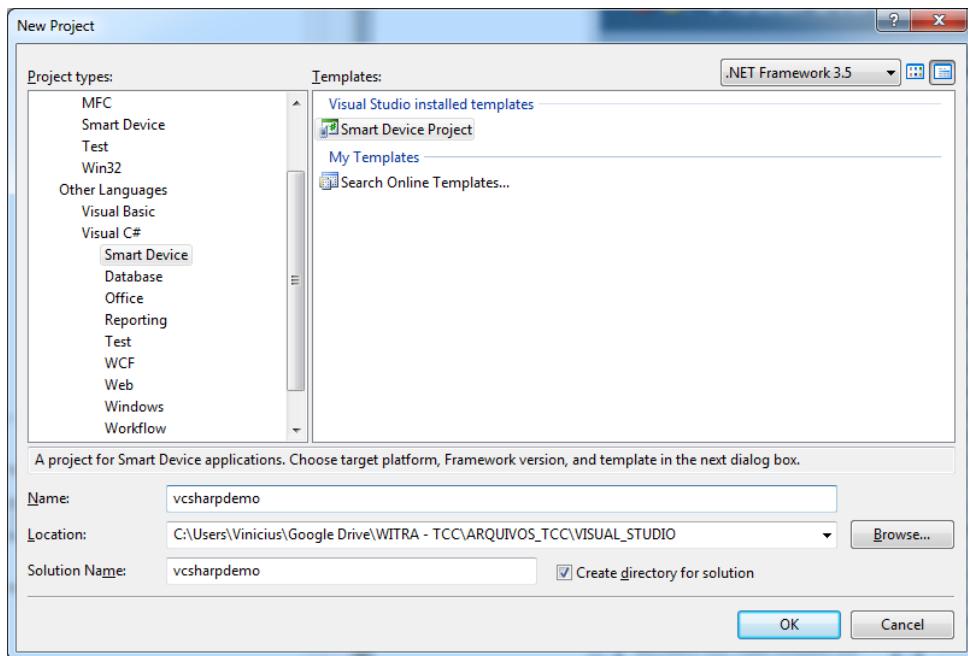


Figura 20: Criando um novo projeto C# no VS2008 - Nome do Projeto

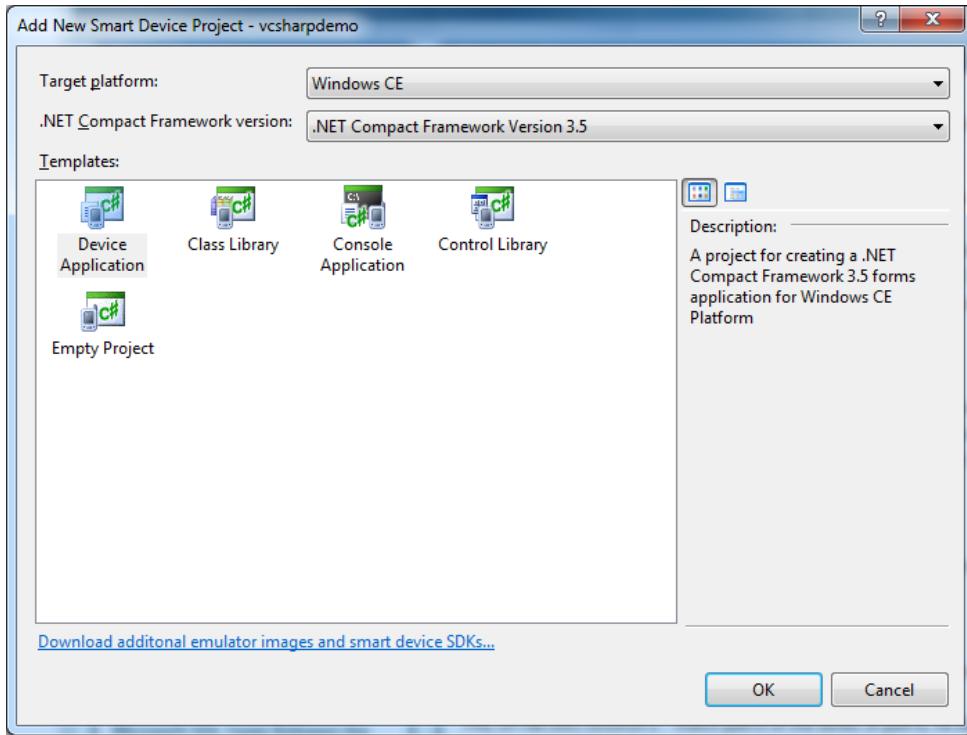


Figura 21: Criando um novo projeto C# no VS2008 - Tipo da Aplicação

Para compilar o programa, basta clicar em no menu *Build* e então em *Build Solution*. Antes de enviar o programa para o Colibri, deve-se ter instalado, no FlashDisk do Colibri, o pacote *.NET Compact Framework*, obtido em [2]. Além disso, deve-se clicar em *Solution->Properties->Devices*. Lá, deve-se desmarcar a opção *Deploy the Lastest version of the .NET Compact Framework(including Service Packs)*. Para enviar o programa para o módulo, deve-se clicar no menu *Build* e então em *Deploy Solution*. Uma janela será aberta, onde é necessário escolher para qual tipo de dispositivo enviar o programa. Se o Colibri está rodando com Windows CE 6.0, a opção será *Colibri600 ARMV4I Device*. Se estiver rodando Windows Embedded Compact 7, a opção será *SDK2wince7 ARMV7*.

3.3.4 Acessando a aplicação no Colibri

Após a aplicação criada no projeto em *Visual Studio* ser enviada para o módulo Colibri, ela será acessível na pasta com mesmo nome que o projeto dentro da pasta *Program Files* do Colibri. É importante ressaltar que os programas salvos nessa pasta são apagados quando o módulo é desligado ou reiniciado. Para manter o programa no módulo, deve-se copiar a pasta do programa para o *FlashDisk*.

3.4 Windows Embedded Compact 7

O sistema operacional Windows Embedded Compact 7 advém de uma linha de sistemas operacionais (SO) para sistemas embarcados que teve sua primeira versão lançada em 1996. Essa versão de sistema operacional não tem muito em comum com as versões para PC do Windows. Trata-se de um sistema voltado para aplicações específicas, tendo suporte para variadas arquiteturas de processador, como ARM, MIPS e x86. É um sistema de 32 bits com capacidades *hard real-time*.

O Windows Embedded Compact é, ainda, um SO de *multi-threading*, *multi-tasking* e preemptivo.

3.4.1 Dispositivos Embarcados

O conceito de *embarcado* é bastante amplo e, no caso presente, aplicado para descrever sistemas e dispositivos que apresentam funções otimizadas e/ou específicas como velocidade de processamento, tamanho físico, memória, de modo a servirem a aplicações de projetos em substituição a um computador que normalmente apresenta maior número de recursos, porém menor número de funcionalidades dedicadas. Não se trata necessariamente de dispositivos de qualidade reduzida ou capacidades limitadas, mas sim de dispositivos com propósitos mais específicos e bem definidos. Na atualidade, por exemplo, já se tem dispositivos embarcados com memória na ordem de grandeza de gigabytes (GB) e processadores com frequência na casa de GHz.

Assim como PCs, dispositivos embarcados apresentam um processador, memória de sistema, memória de armazenamento e aplicações de software. Por outro lado, as diferenças residem no fato de um dispositivo embarcado se prestar a propósitos específicos e, portanto, carece de abundantes dispositivos de entrada e saída e apresenta memórias de sistema e de armazenamento reduzidas, em proporção necessária para armazenar os dados correspondentes às aplicações criadas.

Dessa forma, dispositivos embarcados podem ser encontrados em diversos âmbitos de nosso dia-a-dia, como aparelhos celulares, televisões, impressoras, aparelhos de GPS, máquinas de cartão de crédito, câmeras digitais, entre outros. Em linhas gerais, nesses dispositivos não se observa em uma tela a interface padrão dos SOs comuns (Windows, Linux, Apple, etc.), mas há um processador, memória e dispositivos de entrada e saída que realizam funções específicas para cada aplicação.

3.4.2 Sistemas Operacionais de Tempo-Real

Um sistema operacional é basicamente um conjunto software-hardware que serve de ambiente para execução de programas, bem como rotinas de controle para operar um computador. Adicionar a denominação *Tempo-Real* traz consigo a definição de um sistema operacional que adicionalmente dá ao usuário controle sobre os recursos do computador, além de controle sobre a temporização dos processos ao permitir manipulação das priori-

dades de tarefas e opções de agendamento.

Em um sistema de tempo real (*RTOS - Real-Time Operating System*) há a preocupação em se completar tarefas e processos dentro de um pacote de tempo previamente estabelecido pelo usuário ou pelo programa. Podemos separar os RTOS em dois tipos básicos:

- Soft Real-Time: pode-se, ocasionalmente, não responder aos requerimentos de tempo e ainda assim manter um nível razoável de performance.
- Hard Real-Time: nesse caso, o descumprimento do requisito de tempo em resposta ocasiona danos graves ao sistema.

O Windows Embedded Compact 7 é um sistema *Hard Real-Time* que apresenta baixa latência (1ms) e desempenho de tempo-real determinístico. O SO se destaca por apresentar execução em várias *threads* e ser preemptivo (tarefas de maior prioridade podem interromper tarefas de menor prioridade), além de ter um agendamento de *threads* baseado em prioridades.

3.4.3 Prioridades e Agendamento Round-Robin

Existem diferentes tipos de técnicas de agendamento de tarefas utilizadas por sistemas operacionais de tempo real. O Windows Embedded Compact 7 utiliza, adicionalmente a um sistema de prioridades o chamado agendamento Round-Robin. A lógica básica de agendamento no Compact 7 é:

1. Se uma tarefa tem prioridade maior que a outra, é executada primeiro;
2. Se duas tarefas têm prioridades igual, o sistema dá até 100ms de *quantum* de execução para cada uma. Se a tarefa não for concluída nesse *quantum*, é interrompida e a outra tarefa é iniciada (Round-Robin).

Duas definições são importantes antes de prosseguir:

- No Windows Embedded Compact 7, a prioridade de uma thread é um número definido entre 1 e 255, sendo 1 a maior prioridade e 255 a menor. O valor padrão de prioridade é 251 (THREAD_PRIORITY_NORMAL);
- *Quantum* é um pacote de tempo definido por sistema ou por software para o máximo tempo de execução de uma thread. No SO em questão, pode assumir até o valor de 100ms (padrão).

O conceito de Round-Robin é então designar fatias de tempo para cada processo em porções iguais em ordem circular, sem prioridades definidas (usado portanto no caso em que tarefas têm igual prioridade). O sistema de agendamento, então, faz um compartilhamento de tempo entre as tarefas, dando um pacote de tempo, o *quantum*, a cada tarefa e a interrompe caso não tenha sido concluída após o término do *quantum*. Essa tarefa é retomada quando um novo pacote de tempo é destinado a ela.

3.4.4 Gerando uma nova imagem do Windows Embedded Compact

O módulo Colibri já vem com uma imagem padrão do Windows Embedded Compact. Essa imagem, contudo não é a mais completa que se pode obter. Em outras palavras, existem recursos que não são inclusos nela, por serem muito específicos a certas aplicações. Esse foi o caso de algumas DLLs e arquivos referentes ao *Bluetooth Stack* do Windows Embedded Compact que, estando em falta na imagem padrão, impedem o funcionamento completo da comunicação via bluetooth com outros dispositivos.

Nesse caso, pode-se criar uma nova imagem customizada. Para tal, alguns passos devem ser seguidos:

1. Deve-se ter o Windows Embedded Compact (no caso presente, versão 7) instalado no PC, o que criará a pasta "C:\WINCE700";
2. Na sequência é preciso fazer download do Platform Builder Workspace e do BSP (*Board Support Package*) referentes à versão do Windows Embedded Compact que será instalada. No caso presente, a versão 7 é utilizada. Deve-se baixar o arquivo "Tegra_Toradex_Core_CE7_1.3beta3.zip" ou versão mais recente, presente em [7] e o arquivo "TEGRA_BSP_CE7_1.3beta3.zip" presente em [8].
3. Ambos os arquivos devem ser descompactados. Na pasta do arquivo "Tegra_Toradex_Core_CE7_1.3beta3.zip" encontra-se a pasta "OSDesings". Essa pasta deve ser copiada para "C:\WINCE700". Na pasta do arquivo "TEGRA_BSP_CE7_1.3beta3.zip" existe a pasta "nvap", a qual deve ser copiada para a pasta "C:\WINCE700\platform";
4. Agora, deve-se abrir o *Visual Studio 2008*. Em seguida, abra o arquivo "ColibriTegra_Core_CE7.pbxml";
5. No *Visual Studio*, clique em *View -> Other Windows -> Catalog View Items*
6. Com isso será possível ver todos os arquivos que já estão inclusos por default na imagem e pode-se adicionar/remover itens.
7. Para compilar a nova imagem, clique em *Build -> Advanced Build Commands -> Sysgen (blldemo -q)*. O processo pode levar vários minutos e, após sua conclusão, um arquivo com extensão .nb0 é criado. Esse arquivo deve ser copiado para o módulo Toradex. No módulo, executar o update tool e selecionar o arquivo da imagem como atualização.

3.4.5 Restaurando a imagem de fábrica

Caso seja necessário restaurar a imagem de fábrica do Colibri, é possível fazer isso baixando a imagem de [9] e seguindo os passos descritos em [10] para entrar em recovery mode e instalar a imagem.

3.5 Bibliotecas e Funcionalidades dos módulos Toradex

Como mencionado anteriormente, o Colibri pode ser programado em linguagens como C++, C# e Visual Basic. Sendo assim, pode-se usar qualquer biblioteca dessas linguagens que seja compilável para Windows CE. A documentação das bibliotecas e funções referentes a Windows CE é encontrado na documentação online da Microsoft [31]. Adicionalmente, a Toradex possui suas próprias bibliotecas, de modo a dar funcionalidade aos componentes do módulo. O acesso a tais bibliotecas, bem como o aprendizado por meio de tutoriais é realizado através da plataforma *Getting Started* da área de desenvolvedor do site da empresa suíça [11]. Dentre as principais bibliotecas e funcionalidades demonstradas, encontram-se o uso dos pinos GPIO das placas de suporte, geração de sinais PWM, utilização de interrupções, programação usando biblioteca CAN, transmissão de dados via UART e via Ethernet (sockets), uso das interfaces de vídeo e áudio, entre várias outras.

Dentre as funcionalidades citadas acima, algumas se destacam neste projeto e portanto são detalhadas na sequência.

3.5.1 Bluetooth

O uso de comunicação bluetooth entre dispositivos foi desde o início um objetivo do trabalho. Apesar de não levado até o final, pôde ser investigado a fundo. O desafio de se adicionar a funcionalidade de bluetooth ao Colibri reside em lidar com a arquitetura do *Bluetooth Stack* do Windows Embedded Compact, detalhada no Apêndice D. Basicamente, necessita-se ter as DLLs e arquivos específicos instalados na imagem do SO, bem como ter os drivers corretos instalados no módulo. A Toradex fornece os drivers específicos em [12] e [13]. A versão para Windows CE 6.0 pode ser instalada no Windows Embedded Compact 7. Além disso, é preciso usar algum adaptador bluetooth para a conexão com outros dispositivos e transmissão de dados. Optou-se por usar o *dongle*, adaptador USB, DELOCK Bluetooth USB 2.0 Micro Cl2 10m V3.0 EDR. Porém, qualquer adaptador bluetooth USB baseado no CSR BlueCore4 chipset (BC04) funciona com o stack bluetooth.



Figura 22: Adaptador Bluetooth USB

Para que se possa usar adaptador bluetooth para comunicação com outros dispositivos, adicionalmente à instalação do driver da Toradex, é preciso seguir alguns passos. O adaptador é reconhecido por outros dispositivos bluetooth, como um PC, por exemplo. Sendo assim, é possível parear os dois dispositivos, como mostrado na figura 23.

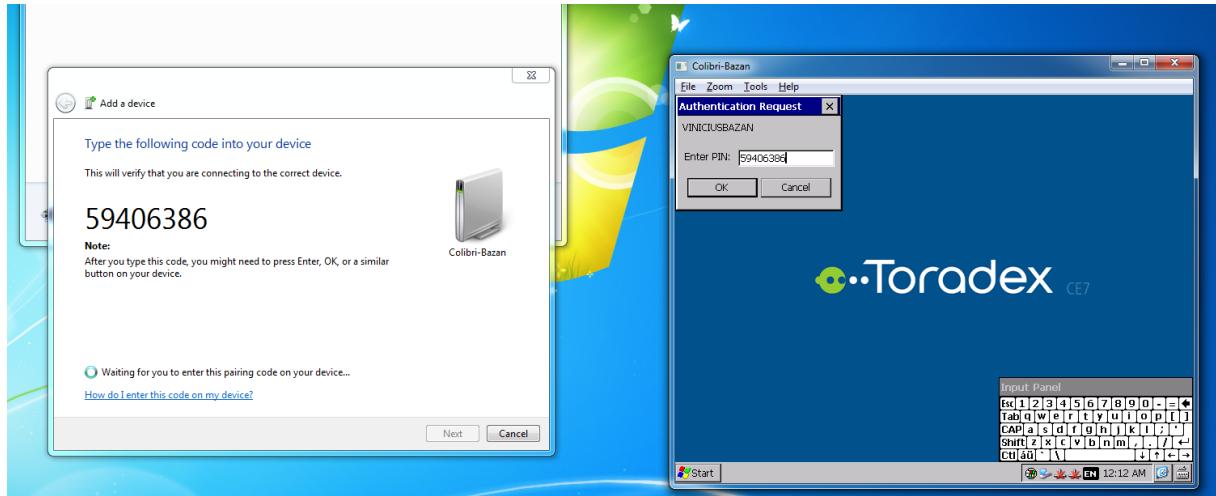


Figura 23: Pareamento entre o Colibri e outro dispositivo bluetooth

O driver bluetooth instalado conta com um *bluetooth manager*, presente no painel de controle ou acionado pelo arquivo **bthpn1** presente na pasta *AutoCopy\Windows* do driver instalado. Esse software permite escanear outros dispositivos bluetooth e adicioná-los

à lista *trusted* (confiável).

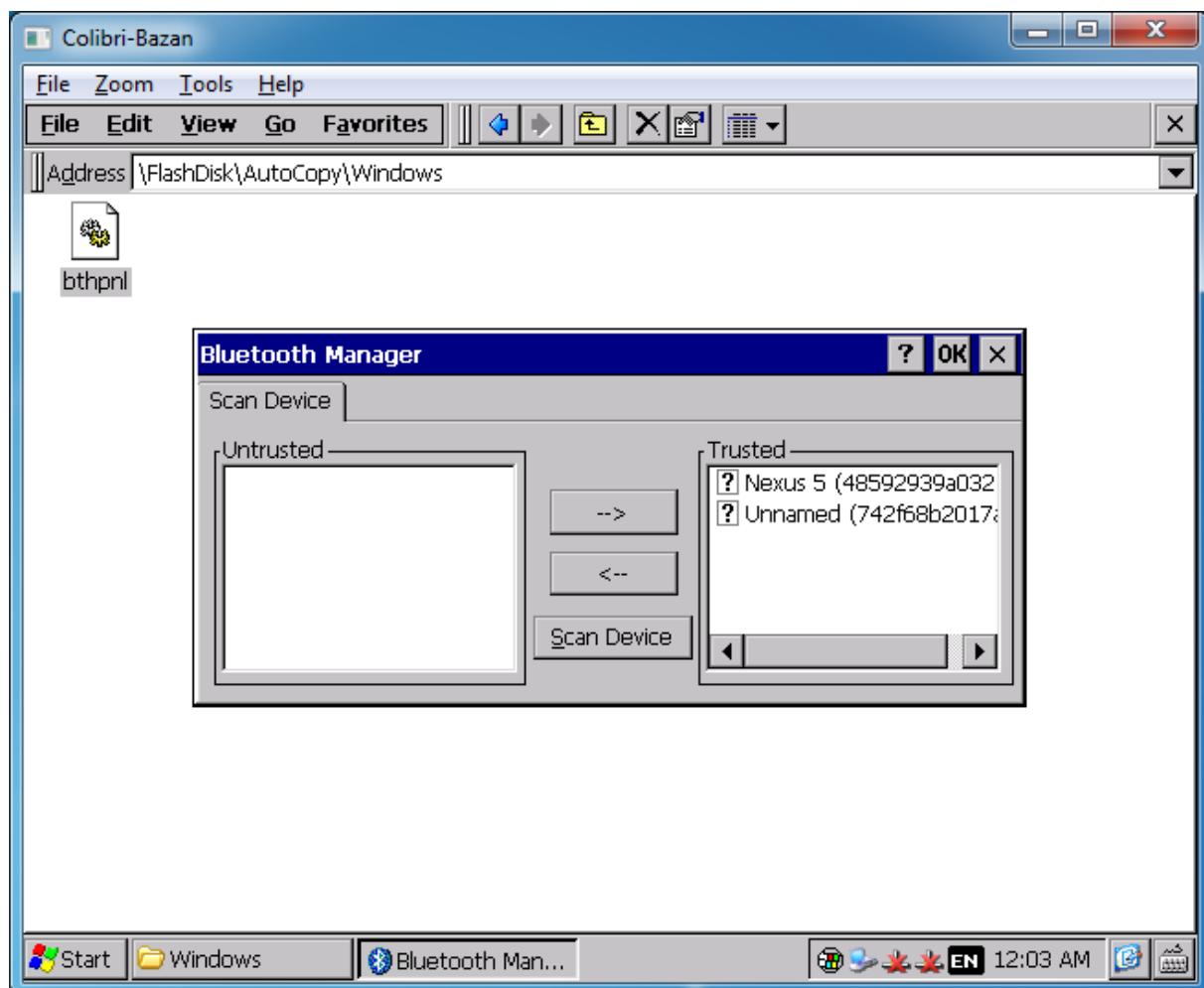


Figura 24: Pareamento entre o Colibri e outro dispositivo bluetooth

A partir daí, seria possível desenvolver uma aplicação para o Colibri que usasse com sucesso a funcionalidade de bluetooth.

Contudo, apenas a instalação dos drivers bluetooth não se mostrou suficiente para o correto funcionamento. A investigação do problema levou à hipótese de que seriam necessários arquivos e DLLs específicas que não estariam instaladas na imagem padrão do Windows Embedded Compact. Conforme a seção 3.4.4, é possível gerar uma nova imagem e a partir daí, incluir os arquivos faltantes. Esse passo foi realizado, porém não se obteve sucesso com a nova imagem. Esse é, portanto, um dos pontos deixados para desenvolvimento futuro e causou uma mudança em hardware e software do projeto. Como comunicação bluetooth é basicamente uma comunicação serial, optou-se pelo uso direto da camada UART para aquisição dos dados dos sensores inerciais. Esse tipo de transmissão

de dados é detalhado a seguir.

3.6 Comunicação Serial via UART

UART é a sigla para *Universal Asynchronous Receiver/Transmitter* (receptor/transmissor assíncrono universal) e é parte do hardware de um computador e traduz dados entre paralelo e serial. É geralmente usado com protocolos de comunicação, especificamente RS-232 no projeto presente. Trata-se geralmente de um circuito integrado usado para realizar comunicação serial através de portas seriais.

O funcionamento básico de uma porta serial se baseia em quatro pinos: VCC, GND, RX e TX:

- VCC - Alimentação positiva de tensão;
- GND - Ground: referência de tensão;
- RX - Receive: pino de recepção de dados;
- TX - Transmit: pino de transmissão de dados.

3.6.1 Pinos UART na Iris

O módulo Colibri Txx tem até cinco portas seriais, sendo três delas existentes por definição e outras duas sendo criadas pelo usuário mediante necessidade. Essas portas são integradas com a placa de suporte Iris. As Figuras 25 e 26 mostram a disposição desses pinos na placa, nos conjuntos de pinos X13, X14 e X16. A relação das três portas seriais padrão é a seguinte:

- COM1 ou UART_A - Chamada *Full Feature* (FF) UART, possui todas as linhas de controle;
- COM2 ou UART_B - Bluetooth UART;
- COM3 ou UART_C - Possui apenas os pinos Rx e Tx

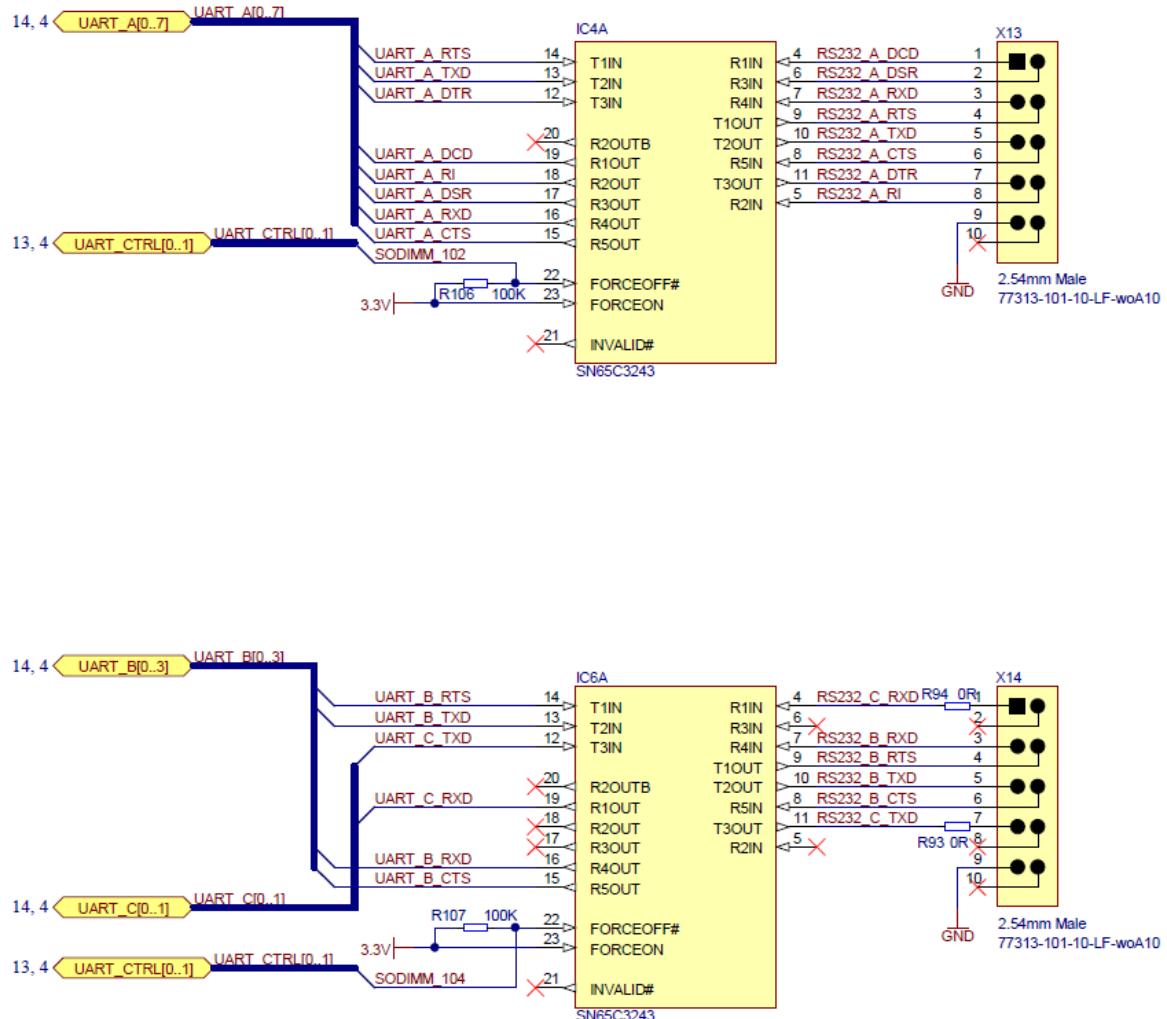


Figura 25: Esquemático da placa Iris - Pinos UART nos conjuntos X13 e X14. Retirado de [18].

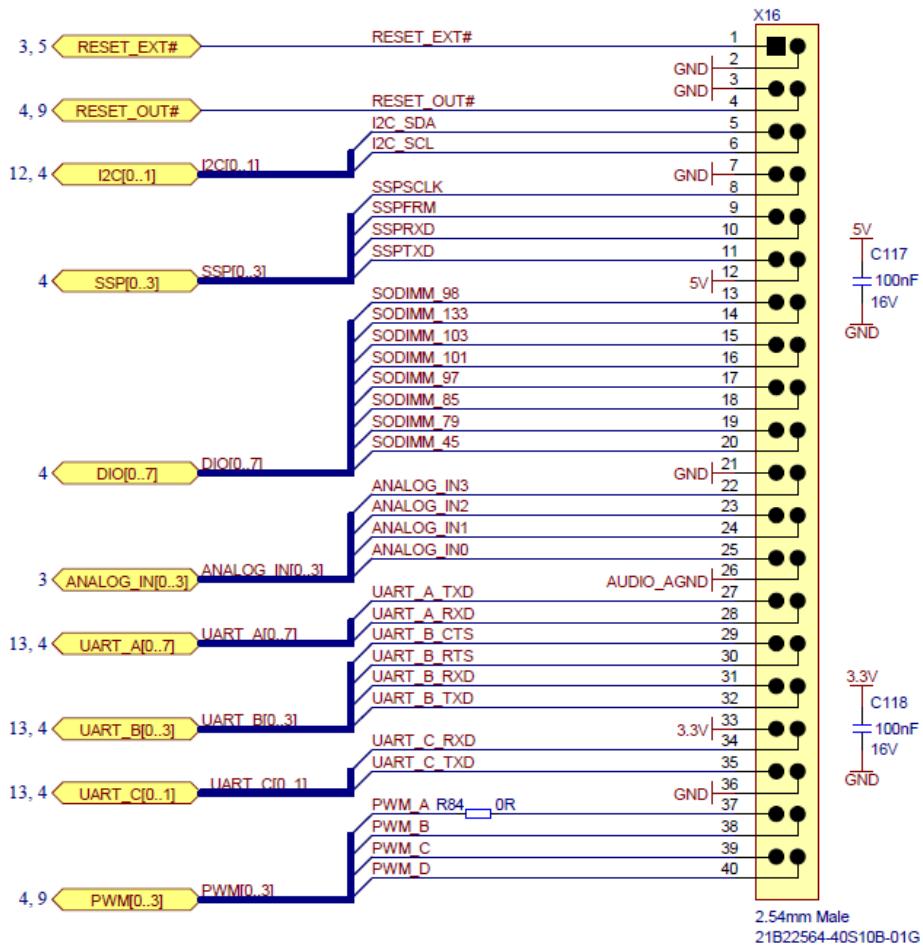


Figura 26: Esquemático da placa Iris - Pinos UART no conjunto X16. Retirado de [18].

3.7 Comunicação Wireless via TCP/IP

Além do uso da funcionalidade de bluetooth, é possível transmitir e receber dados no módulo através de uma interface wireless via TCP/IP. Esse protocolo, conhecido também por *Internet Protocol Suite*, é um conjunto de protocolos de comunicação usado para a internet e outras redes entre computadores. A sigla TCP/IP se refere ao *Transmission Control Protocol* e ao *Internet Protocol*, os dois protocolos de comunicação em rede definidos no padrão do Internet Protocol Suite.

TCP/IP especifica como os dados devem ser empacotados, endereçados, transmitidos, roteados e recebidos no destino. Em uma visão geral, tem-se um servidor e vários clientes que se conectam a esse servidor através de uma rede. Cada dispositivo (clientes e servidor) possui um endereço próprio, uma identidade, chamada de endereço de IP. Assim, é possível especificar para qual dispositivo se deve enviar os dados a cada momento.

Para estabelecer a comunicação entre o Colibri e um PC por meio desse protocolo,

necessita-se um adaptador wireless compatível com o módulo Toradex, como descrito em [14]. O adaptador usado foi o AmbiCom WL250N-USB 150Mbps Wireless-N USB Dongle. Para esse adaptador existem drivers fornecidos pela Toradex em [15]. Após a instalação do driver, é possível acessar a página de conexões do Colibri, onde pode se ver as redes disponíveis.

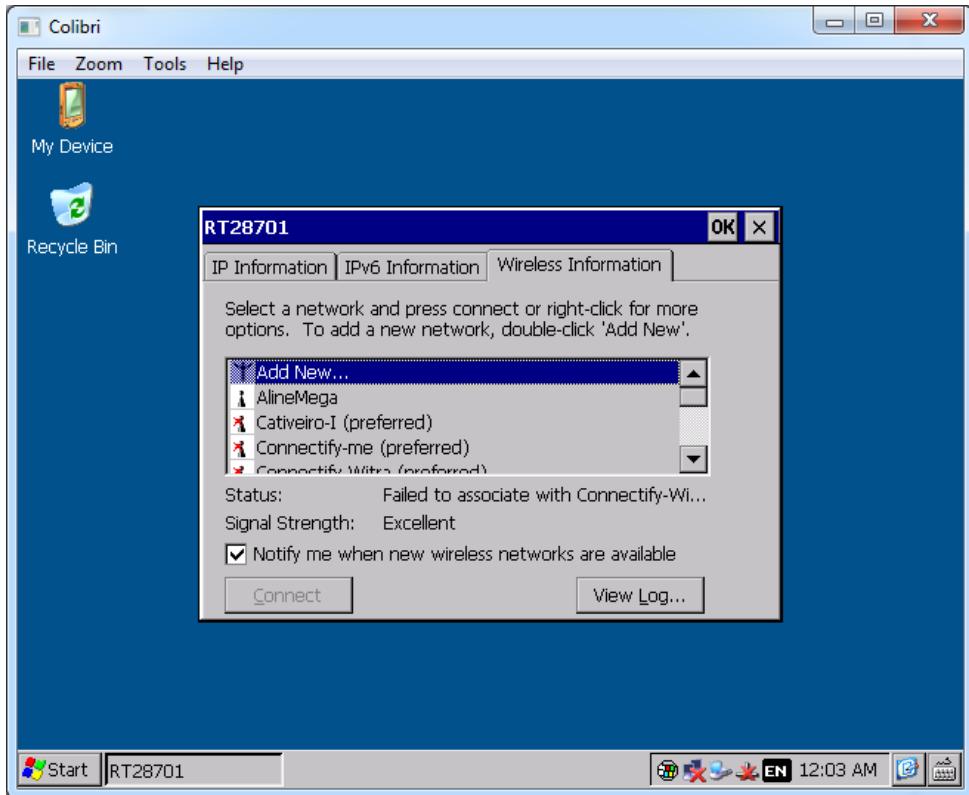


Figura 27: Redes WiFi disponíveis.

3.7.1 Usando Connectify Hotspot para criar uma rede WiFi

Para que dois dispositivos comuniquem entre si por TCP/IP, é preciso que se tenha uma rede WiFi à qual ambos dispositivos possam se conectar. Para tal, uma opção é usar um roteador e criar uma rede. É válido frisar que para essa aplicação de transmissão de dados sem fio, não é necessário o acesso à internet se esta não for requisitada.

Apesar de se poder criar uma rede usando um roteador, em alguns casos um roteador pode não estar disponível, além de ser um dispositivo a mais agregado ao projeto. Portanto, uma ferramenta bastante útil é o software *Connectify* [34], mostrado na Figura 28. Com ele, consegue-se criar um hotspot WiFi no PC de modo que uma rede que permite a conexão do PC e do Colibri seja criada. Há sobretudo a vantagem de se ter uma rede dedicada a esses dois dispositivos, não comprometendo a transmissão de dados devido a

outros dispositivos conectados à rede.



Figura 28: Interface do Software Connectify Hotspot

Após criar uma nova rede, o endereço de IP do PC é mostrado numa janela de notificação. Do lado do Colibri, pode-se conectar à rede criada, no exemplo Connectify-WITRA, e verificar o endereço de IP deste. Com isso, pode-se testar o ping usando o command prompt do PC e dando ping no endereço de IP do Colibri.

3.7.2 Sockets

O protocolo TCP/IP usa como meio de transmissão a tecnologia Ethernet. No Windows Embedded Compact, a programação para uso da tecnologia Ethernet é feita através

do uso de sockets, presentes na biblioteca Winsock. Um socket é o ponto final (*endpoint*) de uma conexão de comunicação de duas vias entre dois programas rodando na mesma rede. Um *endpoint* é uma combinação de um endereço de IP e um número de porta. É ligado a um número de porta, de modo que a camada TCP identifique para qual aplicação os dados devem ser enviados.

Como mencionado anteriormente, para a transmissão de dados via TCP/IP, faz-se necessário um servidor e um cliente. O servidor, que roda em um computador, possui um socket ligado a uma porta específica. O servidor então espera e fica escutando (*listen*), através do socket, por uma requisição de conexão de um cliente. O cliente, por sua vez, sabendo o endereço de IP ao qual deve se conectar através de uma porta específica, faz a requisição de conexão mencionada.

Quando a conexão é estabelecida, o servidor possuirá dois sockets (dois endpoints), um ligado à sua porta local estabelecida para comunicação e um remoto configurado para o endereço e porta do cliente. São necessários dois sockets para que o socket local continue escutando conexões de outros clientes e o socket remoto cuide da transmissão de dados. Por sua vez, o cliente possui um socket, através do qual se comunica com o servidor. A comunicação é realizada pela escrita e leitura nos sockets.

3.7.3 Winsock

A programação por sockets para Windows CE é realizada mediante o uso da interface **Winsock**. Ela oferece uma API (*Application Programming Interface*) geral de comunicação em redes baseada na interface de sockets da *University of California at Berkeley*. Winsock possui uma interface independente de protocolo que é totalmente capaz de sustentar funcionalidades de comunicação em rede, como por exemplo a comunicação de tempo real.

Trata-se de uma interface transparente a protocolos. Como Winsock não é um protocolo de comunicação em si, não afeta os bits no barramento de comunicação, e não precisa ser usado em ambos os pontos finais (*endpoints*) da linha de comunicação. Dessa maneira, oferece acesso para diversos protocolos de transporte, o que permite criar aplicações que sustentam variados tipos de sockets.

Winsock é, portanto, uma especificação técnica que define como os softwares de rede da Windows devem acessar serviços de rede, especialmente TCP/IP. Define-se uma interface padrão entre uma aplicação de cliente TCP/IP em Windows e a pilha do protocolo TCP/IP na camada inferior.

3.7.4 Estabelecendo a comunicação

O uso de sockets é imprescindível para a troca de dados entre dois dispositivos via TCP/IP. Uma aplicação para envio e recebimento de dados via Ethernet é mostrada na área de desenvolvedor do site da Toradex, presente em [16], e é aplicável para o caso de comunicação WiFi, bastando substituir o cabo Ethernet pelo adaptador WiFi. Com [16]

é possível criar um *server* ou *client*, dependendo das necessidades do projeto. É apenas importante ressaltar que um dos dois dispositivos deve ser o *server*, por exemplo o PC, e o outro, o *client*. Uma boa maneira de se testar o código responsável pela comunicação através de sockets é criando uma aplicação para o colibri (como *client*, por exemplo) e usando o programa *Hercules* [35], mostrado na Figura 29 para criar um server no PC.

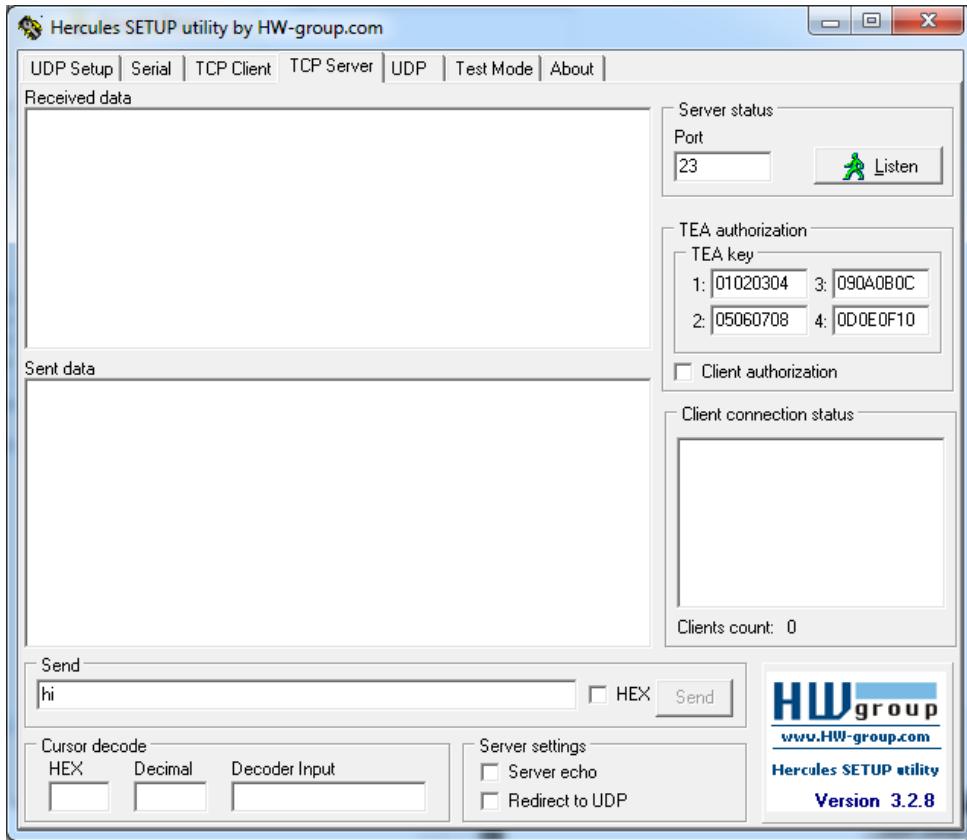


Figura 29: Interface do Software Hercules

3.8 Fluxograma do Software

A Figura 30 ilustra os estados do software, bem como sua sequência de execução. Esse é o software implementado no Colibri, com o conceito de programação de sistemas de tempo-real. O programa é executado em duas threads: a thread do kernel do sistema operacional, atrelada à função `main()` e a thread que realiza praticamente todo o trabalho, chamada de `workerThread()` para fins ilustrativos. Essa thread é responsável por ler os dados advindos da interface serial das IMUs, processá-los e enviar a informação de posicionamento para o robô. Ao passo que essa thread é executada, do lado da thread do kernel, o programa espera pelo usuário apertar a tecla ENTER para parar a execução do programa.

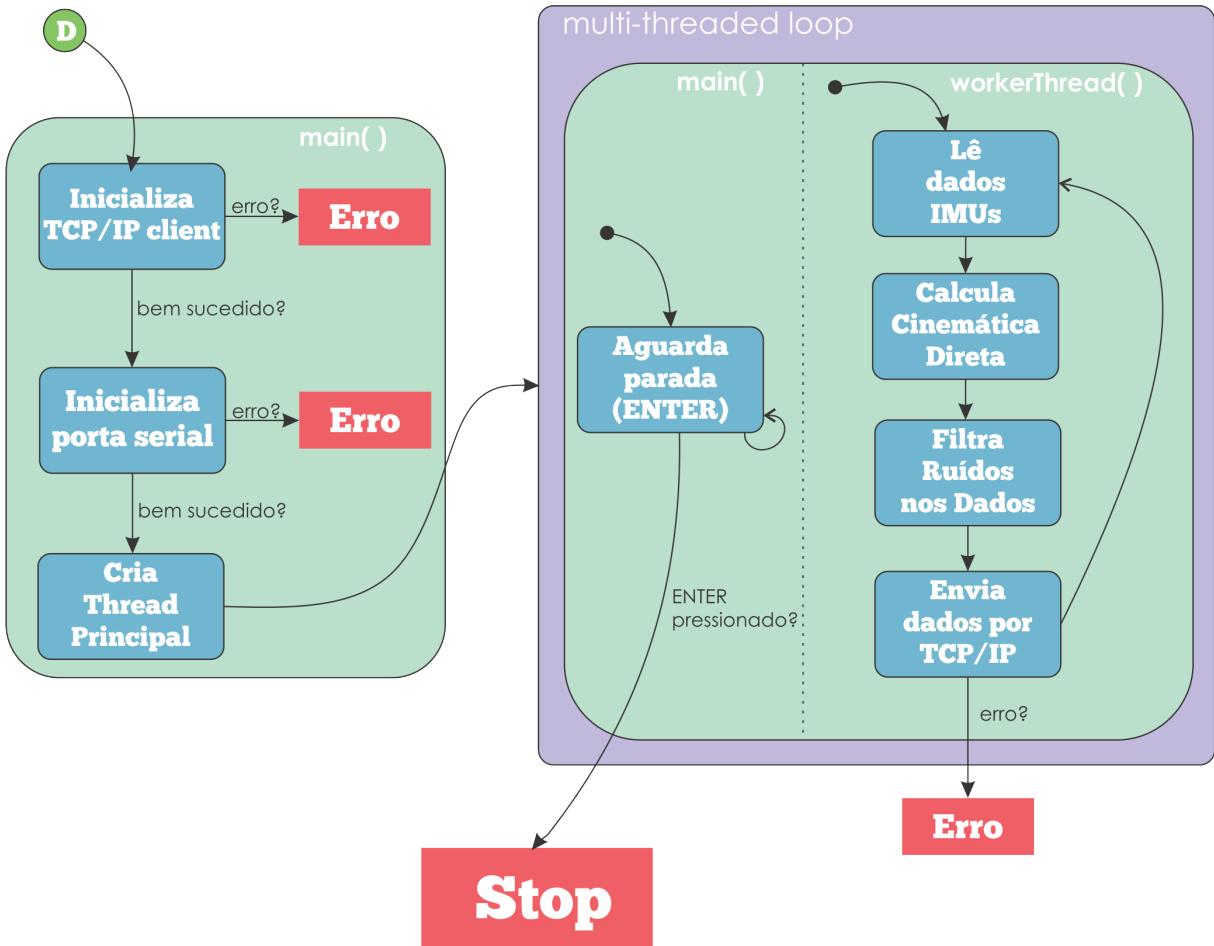


Figura 30: Representação em statechart do software

3.9 Teleoperação

O objetivo final do projeto é ser capaz de teleoperar um manipulador robótico. Na sequência, apresenta-se as maneiras nas quais essa tentativa foi realizada. Primeiramente, mostra-se como foi criada uma animação do robô SCARA 7545 em MATLAB para a demonstração do projeto. O grande propósito de se criar a animação do robô é o fato de que o software de animação pode receber os dados de posicionamento do robô da mesma maneira que o software do robô físico o faz. Isso se deve ao fato, mencionado em 3.7.3, de que o Winsock é uma interface transparente a protocolos e pelo fato de sockets são até certo ponto genéricos para diversas linguagens de programação. Dessa maneira, a parte de desenvolvimento de software do projeto em si para no envio dos dados de posicionamento do robô via TCP/IP. O software após esse ponto, seja ele o da animação em MATLAB ou do robô físico, recebem dados da mesma maneira.

Portanto, a demonstração de funcionamento com a animação é válida também para o funcionamento com o robô físico, a menos de detalhes técnicos. Tendo isso sido dito,

justifica-se a não apresentação, neste momento de uma demonstração com o robô físico, fato esse que foi causado por dificuldades inerentes ao acesso ao software de controle do SCARA 7545. Tal software foi desenvolvido e apresentado em [3] e, portanto, foi necessário contato direto com o autor. Devido a limitações de tempo, a aplicação do projeto usando a versão física do robô fica para uma etapa futura de desenvolvimento.

3.9.1 Mapeamento do espaço de trabalho do usuário para o espaço do robô

Esse é um grande desafio do projeto: mapear dois conjuntos físicos de geometrias e cinemáticas diferentes. Como já dito, o braço humano está sendo tratado como um conjunto multicorpos de 7 GdL, enquanto o robô SCARA possui apenas 4. Mais importante, os dois conjuntos possuem envelopes de trabalho diferentes. Para o braço humano, uma casca esférica, para o robô SCARA, a forma mostrada na Figura 10.

Dessa forma, uma maneira encontrada de se mapear um espaço no outro, sem usar fórmulas matemáticas complexas, que podem comprometer o tempo de execução do software, foi selecionar como espaço de trabalho seções do espaço de trabalho total de cada um (usuário e robô) que tivessem geometrias iguais.

Assim, analisando ambos os espaços de trabalho, conclui-se que para $\theta_1 = 0^\circ$ até $\theta_1 = 160^\circ$, o robô SCARA trabalha, visto de cima, dentro de um anel circular de raio interno aproximadamente igual a $MIN_SCARA = 0.28m$ e raio externo $MAX_SCARA = 0.65m$.

De maneira similar, o usuário consegue atingir um espaço similar, para seu θ_1 entre -70° e 70° (mesma amplitude angular). O raio interno desse anel circular para o usuário é dado pelo tamanho de seu antebraço e ocorre quando o usuário forma um ângulo de 90° no cotovelo, com o antebraço na horizontal. O raio externo é a soma das extensões do braço e antebraço.

Com a vista superior sendo um anel circular, na terceira dimensão o espaço de trabalho é uma simples extrusão do anel. Dessa forma, a altura do pulso do usuário é usada como entrada direta para o deslocamento da junta prismática do SCARA, mediante operação matemática de mapeamento linear. Também as duas coordenadas restantes são mapeadas mediante operação matemática, para garantir que quando o braço do usuário estiver totalmente esticado na horizontal, também o SCARA esteja totalmente esticado em seus dois links iniciais e, quando o braço do usuário estiver retraído, formando 90° no cotovelo, com o antebraço na horizontal, o SCARA atinge o raio interno de seu espaço de trabalho. A Figura 31 ilustra o que foi dito até o momento.

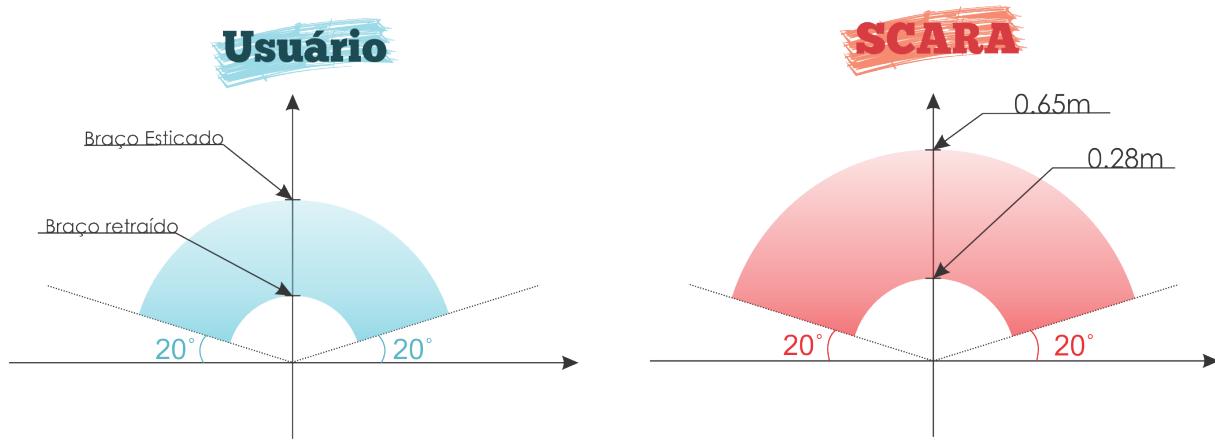


Figura 31: Mapeamento entre o espaço de trabalho do usuário e do robô SCARA

3.9.2 Animação do Robô SCARA 7545 em MATLAB

MATLAB é um software de engenharia largamente usado para simulação e cálculos complexos. Seu uso é de grande valia em aplicações de engenharia. Para uma construção de um modelo fiel de simulação do robô SCARA 7545, fez-se uso da *Robotics Toolbox* de Peter Corke, presente em [4]. Essa toolbox oferece diversas funções e funcionalidades para o estudo e simulação de manipuladores robóticos. Entre as funções implementadas, destacam-se o estudo da cinemática (direta e inversa) do robô; a animação do robô em si, criado a partir dos parâmetros de Denavit-Hartenberg; o estudo da dinâmica; e a geração de trajetórias. As funções são aplicáveis a robôs de links seriais, ou seja, de cadeia aberta e encapsulam os parâmetros robóticos em objetos no MATLAB: objetos do tipo robô podem ser criados pelo usuário e usados em conjunto com uma variedade de funções.

A seguir é mostrado o processo de criação de um robô usando o *Robotics Toolbox*. Como essa construção é baseada na cinemática direta do robô, a toolbox busca construir a matriz de transformação

$$T_{base}^{ferramenta} = T_0^1 T_1^2 T_2^3 T_3^4$$

Para o robô SCARA 7545, a tabela de parâmetros de Denavit-Hartenberg é dada por

Tabela 1: Parâmetros D-H

i	θ	d	a	α
1	q_1	d_1	a_1	π
2	q_2	0	a_2	0
3	0	q_3	0	0
4	q_4	d_4	0	0

Onde q_1 , q_2 , q_3 e q_4 são variáveis do robô e $a_1 = 0.400$, $a_2 = 0.250$, $d_1 = 0.250$ e

$d_4 = 0.050$, medidos em metros. Com esses dados em mãos, é possível criar os objetos do tipo *link*, que correspondem a cada um dos links do robô. Tendo esses links criados, constroi-se o objeto do tipo *robot*, como mostrado na Figura 32

```

18 %& DEFINIÇÃO DOS LINKS
19 % [alpha a theta d R/P]
20
21 - L0 = link([0 0 0 d1 0], 'standard'); %esse link foi adicionado pois o link 1 nos parametros DH
22 %possuia d e a e portanto era plotado
23 %como uma linha inclinada, ao invés de
24 %uma horizontal e uma vertical. Essa é
25 %a linha vertical e mantemos todos os
26 %parâmetros constantes
27
28 - L1 = link([pi a1 0 0 0], 'standard'); % variável = theta1
29 - L2 = link([0 a2 0 0 0], 'standard'); % variável = theta2
30 - L3 = link([0 0 0 0 1], 'standard'); % variável = d3
31 - L4 = link([0 0 0 d4 0], 'standard'); % variável = theta4
32
33 %% DEFINIÇÃO DO ROBÔ
34
35 - scara = robot({L0 L1 L2 L3 L4});
36 - scara.name = 'SCARA';
37

```

Figura 32: Código MATLAB para criação do robô SCARA 7545

Com o objeto do tipo *robot* criado, pode-se usar a função *plot()*, que é sobrecarregada (*overloaded*) para esse tipo de objeto, a fim de se obter uma representação gráfica do mesmo. A função usada é

```
plot(scara, [0 theta1 theta2 d3 theta4]);
```

Onde o segundo parâmetro da função é um vetor com a situação atual das juntas do robô. A chamada dessa função cria a representação tridimensional do robô SCARA, mostrada na Figura 33.

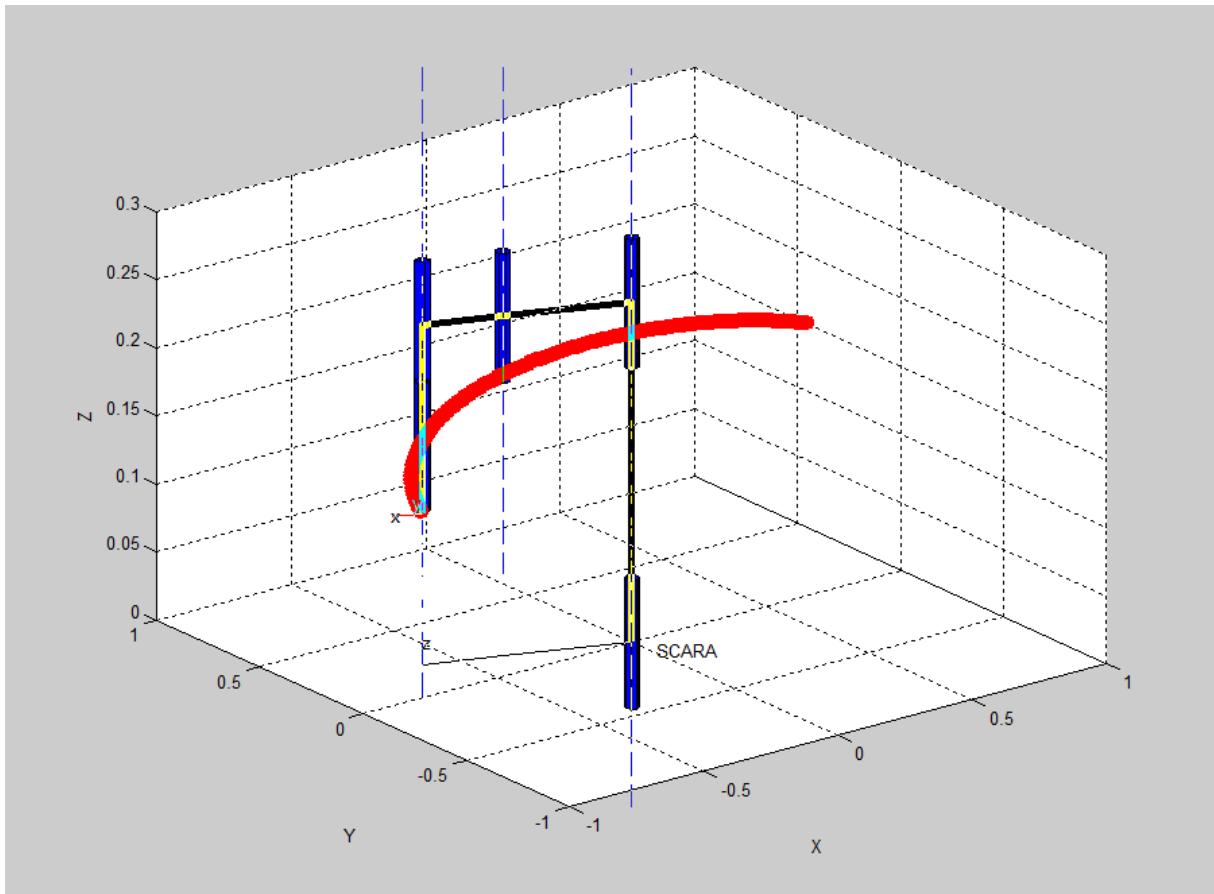


Figura 33: Representação gráfica do robô SCARA 7545

A linha vermelha de trajetória mostrada na Figura 33 foi obtida traçando no grafico tridimensional a posição do *Tool Center Point* (TCP) do robô a cada iteração do programa. A obtenção desses pontos foi realizada através da função *fkine()*, que realiza a cinemática direta do robô em questão:

```
TR = fkine(scara, [0 theta1 theta2 d3 theta4]);
```

Similar a essa função é a função *ikine()*, que calcula a cinemática inversa do robô. Porém, devido a problemas com o uso dessa função, procedeu-se com o cálculo da cinemática inversa do robô SCARA de forma geométrica e essas equações foram implementadas no código. O SCARA possui, como já mencionado, 4 GdL, que analisamos separadamente. Primeiramente, se olhado de cima, o SCARA é um pêndulo duplo, como mostrado na Figura 34, no qual o ângulo θ_1 tem seu valor positivo na rotação em sentido horário com limites rotacionais 0° e 200° . Já θ_2 é positivo no sentido anti-horário e tem seus limites rotacionais em 0° e 135° .

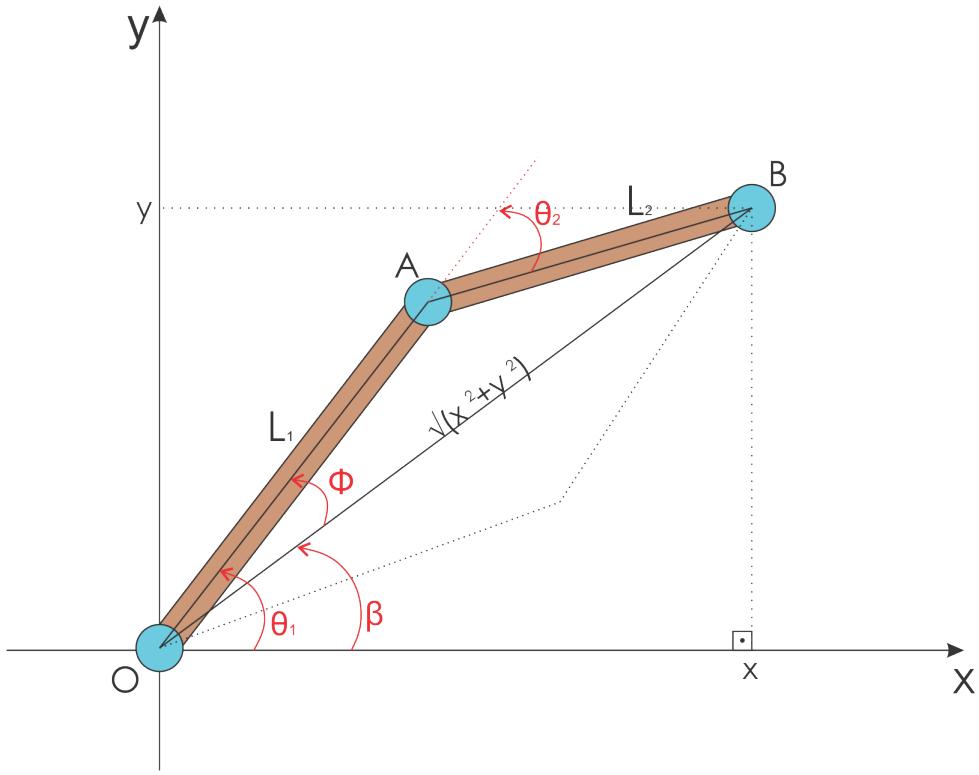


Figura 34: Visão superior do SCARA - Um pêndulo duplo

Com isso, podemos aplicar a lei dos cossenos no triângulo OAB da Figura 34:

$$\begin{aligned}
 x^2 + y^2 &= l_1^2 + l_2^2 - 2l_1l_2\cos(180 - \theta_2) \\
 x^2 + y^2 &= l_1^2 + l_2^2 + 2l_1l_2\cos(\theta_2) \\
 \cos(\theta_2) &= \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \\
 \Rightarrow \theta_2 &= \cos^{-1}\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2}\right)
 \end{aligned} \tag{4}$$

Mais além, podemos encontrar o valor de β :

$$\beta = \tan^{-1}(y/x)$$

Aplicamos novamente a lei dos cossenos em OAB para encontrar ϕ :

$$\begin{aligned}
 l_2^2 &= x^2 + y^2 + l_1^2 - 2\sqrt{x^2 + y^2}l_1\cos(\phi) \\
 \cos(\phi) &= \frac{x^2 + y^2 + l_1^2 - l_2^2}{2\sqrt{x^2 + y^2}l_1} \\
 \phi &= \cos^{-1}\left(\frac{x^2 + y^2 + l_1^2 - l_2^2}{2\sqrt{x^2 + y^2}l_1}\right)
 \end{aligned}$$

$$\Rightarrow \theta_1 = \beta + \phi \quad (5)$$

Note que ainda há uma segunda solução, na qual $\theta_1 = \beta - \phi$ e $\bar{\theta}_2 = -\theta_2$

Ainda restam dois GdL: a movimentação da junta prismática em z (altura) e a rotação da ferramenta. Esses GdL são mais fáceis de resolver, sem uso da cinemática inversa. No mapeamento de espaços de trabalho, a altura da mão do operador em sua referência cartesiana é usada como entrada direta para a altura da ferramenta do SCARA, mediante mapeamento linear. A rotação da ferramenta é tratada também diretamente, como a rotação da mão do operador.

Dessa forma, o programa da animação em MATLAB recebe, via TCP/IP, os dados de posicionamento do pulso do operador, calculados através da cinemática direta do braço humano. Com isso, realiza o mapeamento de espaço para o espaço de trabalho do SCARA e plota a animação do robô na posição desejada a cada instante.

4 Resultados e Discussão

O resultado principal do projeto é, naturalmente, o desenvolvimento da interface para teleoperação de um braço robótico baseado nos movimentos do braço humano. O projeto ganhou sigla, WITRA, que em inglês significa Wearable Interface for Teleoperation of Robot Arms. A Figura 35 exemplifica a montagem da interface com um sensor inercial. Ela mostra o sensor inercial posicionado na mão do usuário e ligado à interface serial. O adaptador bluetooth mostrado é o adaptador WiFi, que envia os dados por rede sem fio.

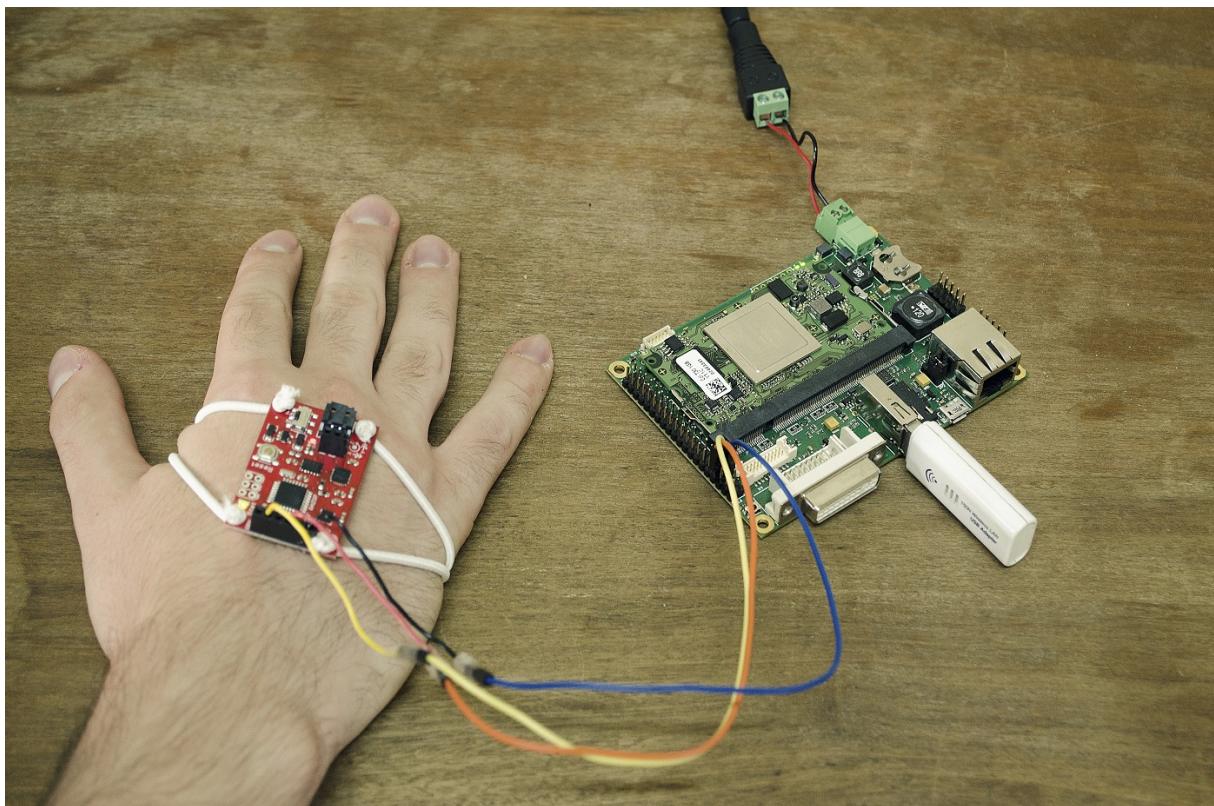


Figura 35: Sistema com um sensor inercial posicionado na mão do usuário

Com o desenvolvimento dessa interface, alguns objetivos e pontos importantes foram alcançados:

- Obteve-se um grande aprendizado no tocante aos módulos da Toradex, em especial no âmbito dos protocolos de transmissão de dados;
- Alcançou-se um know-how sobre o uso de um sistema embarcado; de um sistema operacional de tempo-real, notadamente o Windows Embedded Compact 7 e da programação desse sistema nas linguagens C++ e C#;
- Em especial, foi possível aprender sobre e implementar a transmissão de dados

via serial, usando a interface UART, bem como a transmissão de dados *wireless*, utilizando a tecnologia Ethernet atrelada ao protocolo TCP/IP;

- Com a implementação desses métodos de comunicação, foi possível realizar a aquisição de dados dos sensores inerciais através a interface serial. Paralelamente, foi possível enviar os dados de posicionamento do robô via TCP/IP;
- A aplicação (software) criada para rodar no Colibri foi implementada usando as práticas de multi-threading e de programação de sistemas de tempo real. Com isso, conseguiu-se obter uma execução do programa com tempo e recursos de processamento bem gerenciados. Mediú-se o tempo de execução da thread principal do programa, aquela que realiza a aquisição, processamento e envio de dados e obteve-se um tempo médio de 32.5ms. Esse tempo corrobora com as limitações de tempo para validação do sistema como tempo-real. Para motivos de comparação, a IMU tem capacidade de samplear dados a cada 50ms. Além disso, um parâmetro comparativo de boa qualidade é a frequência de quadros em um filme. A maioria dos filmes opera em 24fps, ou seja, com um frame a cada aproximadamente 41.67ms. Isso permite que o olho humano não perceba a transição entre um quadro e outro, de modo que figuras estáticas em sequência pareçam um movimento fluido. É pregado também que com uma taxa de 10fps, aproximadamente um quadro a cada 100ms, já se tem essa noção de fluidez. Assim, ao se alcançar um tempo total de execução até 100ms (tempo entre uma movimentação do robô para um determinado ponto no espaço até a movimentação para o ponto seguinte) é possível transmitir ao usuário a noção de fluidez no movimento. Mais além, o fato de se obter um pequeno tempo de execução de cada iteração do software, alcança-se a noção de tempo-real discutida, uma vez que não se nota atraso significativo entre o usuário mexer seu braço para uma nova posição e o robô de fato responder se posicionando na posição desejada.
- Foi com sucesso que se implementou uma animação do robô real, o SCARA 7545, de modo que o software pudesse ser validado previamente à sua implementação no robô, que é um passo a ser concluído no futuro.

Com os tópicos acima abordados, é possível se ter uma visão global do projeto e de como ele articula seus recursos para alcançar o objetivo proposto.

5 Próximas Etapas

Dado o desenvolvimento presente do projeto, estabelece-se como fase posterior do desenvolvimento do projeto a adição de mais funcionalidades ao equipamento, como feedback visual por meio de vídeo, de modo a possibilitar o afastamento entre o usuário e o robô; a adição de maior número de graus de liberdade de controle; a remodelagem do mapeamento entre os envelopes de trabalho do usuário e do robô, entre outras características que possam advir da pesquisa e desenvolvimento atrelados ao projeto.

6 Conclusão

Com o projeto concluído até esta etapa, obteve-se uma metáfora de teleoperação do robô SCARA mediante o uso dos movimentos naturais do braço humano. A intuitividade dessa metáfora, comprovada em [1], vem a contribuir com o ramo de desenvolvimento de interfaces homem-máquina. É possível, através do uso de tal interface, reduzir o tempo de aprendizado de um usuário leigo, de modo que seja mais ágil no uso em suas aplicações inerentes. O uso de sensores inerciais para a captação dos movimentos do braço do operador foi uma solução bastante viável, pois estes englobam em um só pequeno encapsulamento três sensores e podem ser acoplados ao braço do usuário de maneira simples, rápida e que não interfira nos movimentos do braço deste.

Dessa forma, o projeto se presta de maneira flexível para aplicações como operação de robôs em profundidades marítimas, manipulação de objetos em ambientes perigosos (com alto índice radioativo, por exemplo) e até aplicações industriais de programação de tarefas a serem realizadas por robôs em linhas de montagem, sem a necessidade de escrita de código específico.

É com grande satisfação que se encerra essa etapa do projeto. Dado o alto objetivo proposto no início, conclui-se que a parte essencial para sua realização foi completa. Os resultados obtidos com o projeto, como todo o know-how obtido e detalhado neste relatório, contribuíram não somente para a formação pessoal, mas certamente contribuem com o desenvolvimento do Laboratório de Mecatrônica e servem como referência para trabalhos futuros na área.

Apêndices

A Cinemática Direta do Braço Humano

O objetivo é obter a matriz de transformação que fornece a posição (x, y, z) do pulso do operador em relação a uma base fixa no ombro do mesmo, bem como a orientação da mão do operador. Para alcançar tais cálculos, usamos os valores de rotação em cada junta do braço humano, que são obtidos por meio dos sensores iniciais discutidos anteriormente. A matriz de transformação em questão é da forma

$$A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

Lida-se aqui, com uma geometria simplificada do braço humano, mas bastante funcional, que considera os 7 GdL principais e ignora o movimento da escápula e a pronação do antebraço. Portanto, os GdLs considerados são: 3 no ombro, que age como uma junta esférica; 1 no o cotovelo; e 3 no pulso, também uma junta esférica. A disposição dos eixos de rotação podem ser observados na Figura 36

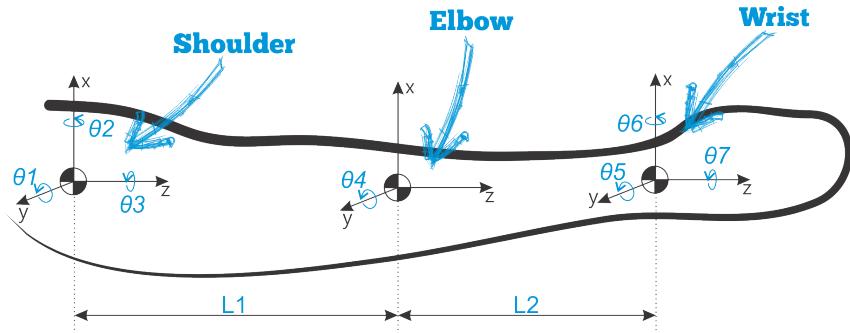


Figura 36: Human Arm Model.

Cada rotação (GdL) é representada por uma matriz 4×4 , A_i , onde $i = 1, \dots, 7$. Além disso, a base de referência x_0, y_0, z_0 é colocada no ombro. Portanto as 7 matrizes são

$$A_1 = R_y(\theta_1) = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

$$A_2 = R_x(\theta_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_2 & -s_2 & 0 \\ 0 & s_2 & c_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

$$A_3 = R_z(\theta_3)T(0, 0, L_1) = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (9)$$

$$A_4 = R_y(\theta_4)T(0, 0, L_2) = \begin{bmatrix} c_4 & 0 & s_4 & s_4 L_2 \\ 0 & 1 & 0 & 0 \\ -s_4 & 0 & c_4 & c_4 L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

$$A_5 = R_y(\theta_5) \begin{bmatrix} c_5 & 0 & s_5 & 0 \\ 0 & 1 & 0 & 0 \\ -s_5 & 0 & c_5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

$$A_6 = R_x(\theta_6) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_6 & -s_6 & 0 \\ 0 & s_6 & c_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

$$A_7 = R_z(\theta_7) \begin{bmatrix} c_7 & -s_7 & 0 & 0 \\ s_7 & c_7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

A matriz resultante é dada por

$$A_{wrist} = A_1 A_2 A_3 A_4 A_5 A_6 A_7$$

Essa matriz, A_{wrist} tem a mesma estrutura que a matriz mostrada em (6), onde

$$r_{11} = s_7(s_6(s_45(s_1s_2s_3 + c_1c_3) + s_1c_2c_{45}) + c_6(s_1s_2c_3 - c_1s_3)) + c_7(c_{45}(s_1s_2s_3 + c_1c_3) - s_1c_2s_{45}) \quad (14)$$

$$r_{12} = c_7(s_6(s_45(s_1s_2s_3 + c_1c_3) + s_1c_2c_{45}) + c_6(s_1s_2c_3 - c_1s_3)) - s_7(c_{45}(s_1s_2s_3 + c_1c_3) - s_1c_2s_{45}) \quad (15)$$

$$r_{13} = c_6s_{45}(s_1s_2s_3 + c_1c_3) + s_6(c_1s_3 - s_1s_2c_3) + s_1c_2c_6c_{45} \quad (16)$$

$$r_{21} = c_2(s_7(s_3s_6s_{45} + c_3c_6) + s_3c_7c_{45}) + s_2(c_7s_{45} - s_6s_7c_{45}) \quad (17)$$

$$r_{22} = c_2(s_3(s_6c_7s_{45} - s_7c_{45}) + c_3c_6c_7) - s_2(s_6c_7c_{45} + s_7s_{45}) \quad (18)$$

$$r_{23} = c_2(s_3c_6s_{45} - c_3s_6) - s_2c_6c_{45} \quad (19)$$

$$r_{31} = s_7(c_1(s_6(s_2s_3s_{45} + c_2c_{45}) + s_2c_3c_6) + s_1(s_3c_6 - c_3s_6s_{45})) - c_7(c_{45}(s_1c_3 - c_1s_2s_3) + c_1c_2s_{45}) \quad (20)$$

$$r_{32} = c_7(c_1(s_6(s_2s_3s_{45} + c_2c_{45}) + s_2c_3c_6) + s_1(s_3c_6 - c_3s_6s_{45})) + s_7(c_{45}(s_1c_3 - c_1s_2s_3) + c_1c_2s_{45}) \quad (21)$$

$$r_{33} = c_1(s_2(s_3c_6s_{45} - c_3s_6) + c_2c_6c_{45}) - s_1(c_3c_6s_{45} + s_3s_6) \quad (22)$$

$$p_x = s_1c_2(L_1 + c_4L_2) + s_4L_2(s_1s_2s_3 + c_1c_3) \quad (23)$$

$$p_y = c_2s_3s_4L_2 - s_2(L_1 + c_4L_2) \quad (24)$$

$$p_z = c_1c_2(L_1 + c_4L_2) + c_1s_2s_3s_4L_2 - s_1c_3s_4L_2 \quad (25)$$

B Firmware e Programação da IMU Razor 9Dof

B.1 Upload do Firmware para o sensor

A IMU Razor 9DoF, como mencionado em 2.1.2, possui um firmware para seu processador interno ATMega 328. Os passos para sua programação foram realizados segundo [20]. A programação deve ser realizada usando um cabo ou placa FTDI. Para realizar o upload do Firmware, realizada através da Arduino IDE [37], é preciso seguir os seguintes passos:

- Conectar a IMU ao computador através de uma interface FTDI (cabô ou placa);
- Abrir o arquivo "Arduino/Razor_AHRS/Razor_AHRS.ino" do pacote de Firmware na Arduino IDE;
- Após ter aberto o arquivo, há a seção "USER SETUP AREA", onde o usuário pode configurar alguns parâmetros;
- Em "HARDWARE OPTIONS", deve-se selecionar o hardware que está sendo usado (no caso a versão SEN-10736 do sensor):

```
/*
***** USER SETUP AREA! Set your options here! *****/
// HARDWARE OPTIONS
// Select your hardware here by uncommenting one line!
#ifndef HW_VERSION_CODE 10125 // SparkFun "9DOF Razor IMU" version "SEN-10125" (HMC5843 magnetometer)
#define HW_VERSION_CODE 10736 // SparkFun "9DOF Razor IMU" version "SEN-10736" (HMC5883L magnetometer)
#ifndef HW_VERSION_CODE 10183 // SparkFun "9DOF Sensor Stick" version "SEN-10183" (HMC5843 magnetometer)
#define HW_VERSION_CODE 10321 // SparkFun "9DOF Sensor Stick" version "SEN-10321" (HMC5843 magnetometer)
#ifndef HW_VERSION_CODE 10724 // SparkFun "9DOF Sensor Stick" version "SEN-10724" (HMC5883L magnetometer)
```

Figura 37: Selecionando o Hardware no Código

- Deve-se selecionar qual tipo de arduino corresponde ao processador da IMU, no caso, "Arduino Pro or Pro Mini (3.3v, 8mhz w/ ATmega328)". Essa seleção é feita em "Tools->Board":

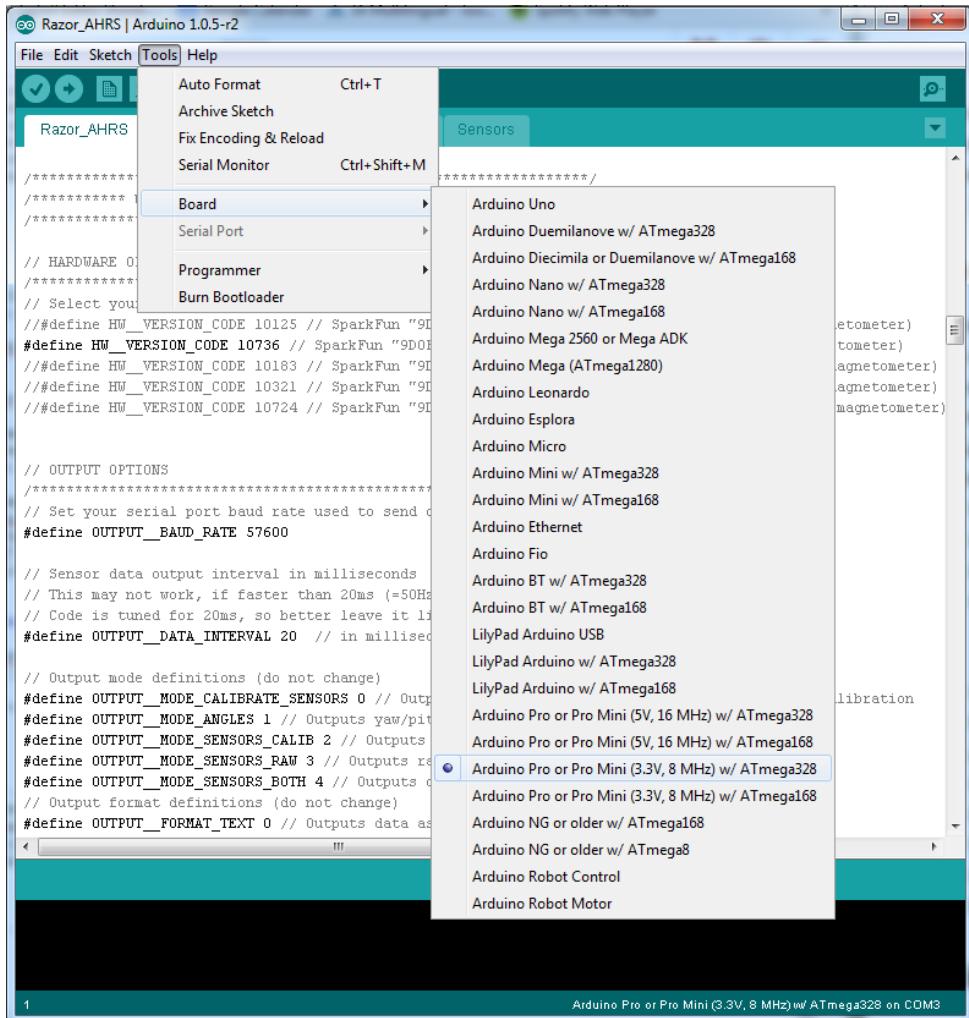


Figura 38: Selecionando o tipo de arduino

- Deve-se também selecionar a porta serial à qual a IMU está conectada (na realidade a porta à qual o cabo/placa FTDI está conectado). Essa seleção é feita em "Tools->Serial Port";
- Para fazer o upload do programa para a IMU, basta clicar no botão de upload (seta horizontal branca mostrada a seguir).

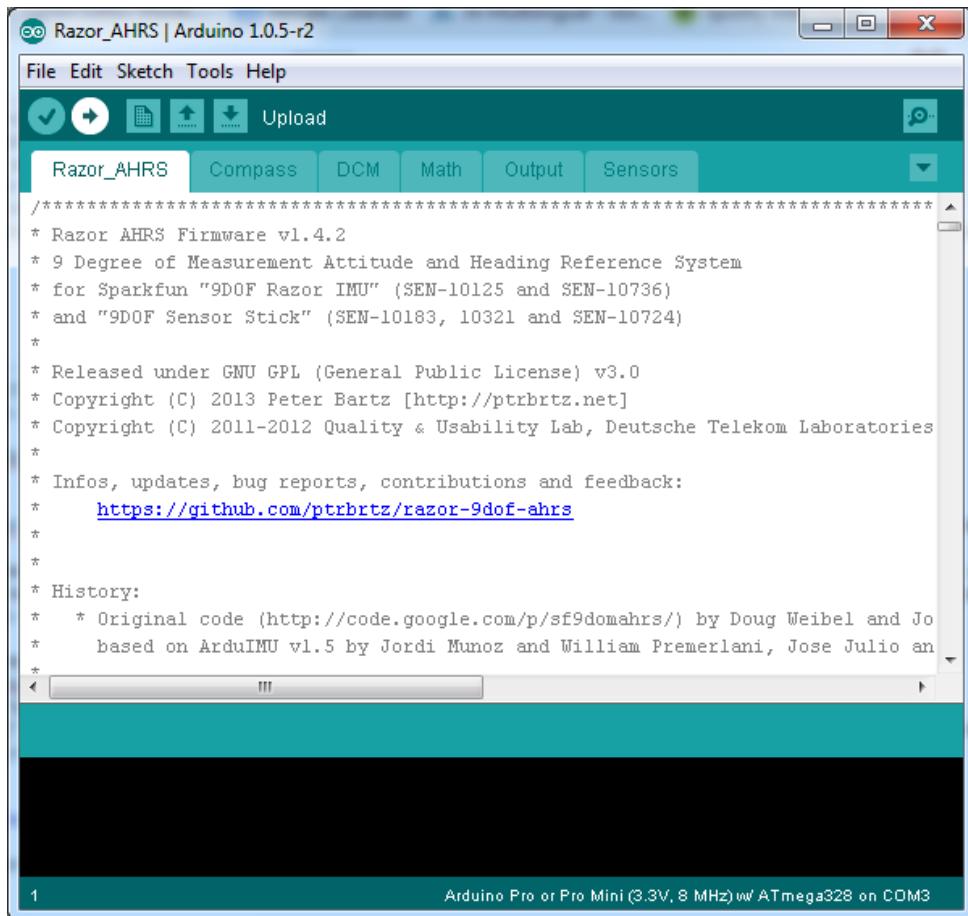


Figura 39: Upload do código para a IMU

É possível, então visualizar os dados de saída da IMU. Para tal, uma maneira simples é, tendo a IMU conectada ao PC e com as configurações anteriores para a Arduino IDE, abrir o monitor serial (Serial Monitor) com as opções "No line ending" e "57600 baud" selecionadas.

Como o formato de saída padrão é texto (pode ser mudado para binário), a saída padrão da IMU do tipo "YPR=-117.85,1.70,-61.61". Por motivos de manipulação de dados no projeto, esse formato de saída foi alterado para -117.85 1.70 -61.61". Essa mudança pode ser realizada alterando-se o código presente em "Output.ino".

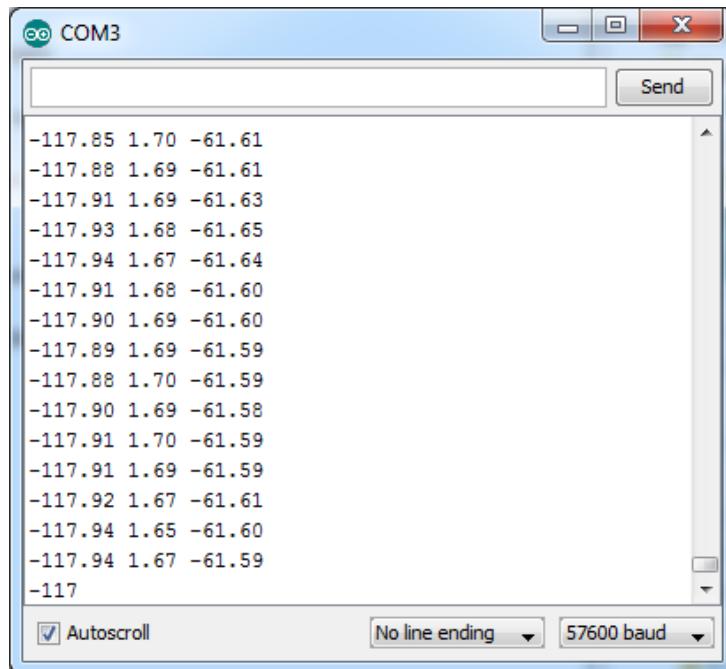


Figura 40: Amostra de saída de dados

B.2 Animação gráfica em Processing

Mais além, há uma visualização gráfica da IMU, utilizando um programa em *Processing*, ferramenta de programação gráfica em Java. O código para essa aplicação está também presente no pacote do firmware da IMU. O software Processing (qualquer versão acima de 2.x) deve ser baixado e instalado [38]. Assim, os passos seguintes são:

- Abrir o arquivo Processing/Razor_AHRS_test/Razor_AHRS_test.pde
- No Processing, ir em "Sketch" e então clicar em "Run"

Uma animação da IMU é mostrada na tela, como mostra a figura a seguir:

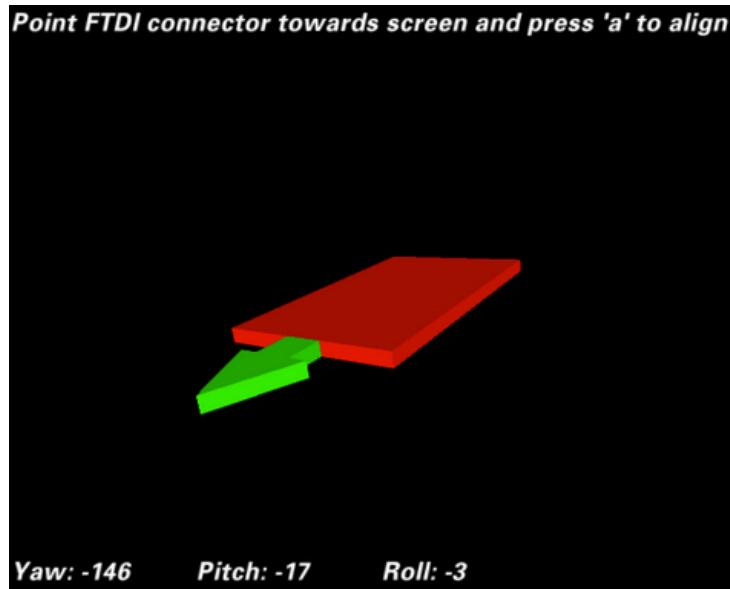


Figura 41: Animação da IMU em Processing

B.3 Calibrando os sensores internos

Apesar de ser um ótimo sensor inercial, a IMU Razor 9DoF, como qualquer outro sensor inercial, conta com imperfeições em seus sensores internos (Acelerômetro, Giroscópio e Magnetômetro). Para superar tais imperfeições, é possível calibrar os sensores para um funcionamento melhorado da IMU. Alguns problemas que podem surgir do fato de não calibrar os sensores internos são uma variação indesejada (drift) no valor de *yaw* quando *roll* é aplicado à IMU e não se ter o sensoriamento correto quando se aponta a IMU para cima.

É importante ressaltar, ainda, que a definição de eixos para cálculo da orientação espacial da IMU não segue o desenho impresso nela. O firmware usa as seguintes definições:

- Eixo-x aponta para frente (na direção do lado menor da IMU onde há os furos de conexão FTDI)
- Eixo-y aponta para a direita
- Eixo-z aponta para baixo

B.3.1 Calibrando o acelerômetro

Antes de calibrar os sensores, recomenda-se ligar a IMU por alguns minutos antes, para que os sensores possam aquecer um pouco. Calibrar o acelerômetro é, entre os três sensores internos, o processo mais delicado. Os passos são os seguintes:

- Abrir o arquivo "Arduino/Razor_AHRS/Razor_AHRS.ino" na Arduino IDE e descer o código até a área "USER SETUP AREA" / "SENSOR CALIBRATION". Essa área é onde se escreve os valores de calibragem;
- Deve-se conectar a IMU ao PC usando a interface FTDI e abrir o monitor serial;
- Para entrar em modo de calibragem, deve-se enviar ao firmware a string `#oc` no campo *Send* do monitor serial. Isso gerará uma forma de saída de dados do tipo

"accel x,y,z (min/max) = -5.00/-1.00 25.00/29.00 225.00/232.00";

- Agora, é preciso encontrar os valores máximos e mínimos de leitura do acelerômetro em cada um dos três eixos. Como o acelerômetro mede acelerações, **é muito importante que a calibragem deste seja feita com movimentos lentos e delicados da IMU**;
- Aqui começa a calibragem em si. Primeiramente deve-se (devagar) apontar o eixo-x para baixo, deixando-o vertical. Enquanto isso, o valor de x máximo vai crescendo. Quando estiver na posição, deve-se segurar a IMU sem mexê-la e enviar novamente o comando "#oc" para resetar a medida;
- Agora, é preciso mexer, devagar, a IMU um pouco e todas as direções até o valor de x máximo não aumentar mais. Esse valor deve ser anotado;
- O mesmo procedimento deve ser feito para o outro sentido do eixo-x, agora apontando para cima, o que levará ao valor mínimo de x;
- O processo se repete para os eixos y e z.
- Ao final, ter-se-á os seis valores de mínimo e máximo para cada eixo. Esses valores devem ser colocados na seção de calibragem do código "Razor_AHRS.ino"

```
// SENSOR CALIBRATION
// ****
// How to calibrate? Read the tutorial at http://dev.qu.tu-berlin.de/projects/sf-razor-9dof-ahrs
// Put MIN/MAX and OFFSET readings for your board here!
// Accelerometer
// "accel x,y,z (min/max) = X_MIN/X_MAX Y_MIN/Y_MAX Z_MIN/Z_MAX"
#define ACCEL_X_MIN ((float) -286)
#define ACCEL_X_MAX ((float) 322)
#define ACCEL_Y_MIN ((float) -297)
#define ACCEL_Y_MAX ((float) 334)
#define ACCEL_Z_MIN ((float) -296)
#define ACCEL_Z_MAX ((float) 257)
```

Figura 42: Adicionando ao código os valores de calibragem do acelerômetro

B.3.2 Calibrando o Giroscópio

Esse procedimento é mais simples. Deixe a IMU totalmente parada em cima da mesa. Na sequência, para mudar do modo de calibragem do acelerômetro, é preciso enviar duas vezes o comando "#on", uma para passar do acelerômetro para o magnetômetro (calibrado posteriormente) e outra para passar do magnetômetro à calibragem do giroscópio. Agora, espere por cerca de 10 segundos, **sem mexer a IMU**. O programa coleta o ruído médio do giroscópio em cada um dos três eixos. A saída de dados será do tipo

```
"gyro x,y,z (current/average) = -58.00/-58.83 49.00/51.11 7.00/7.57";
```

Esses valores devem ser inseridos no código "Razor_AHRS.ino", assim como foi feito para o acelerômetro.

```
// Gyroscope
// "gyro x,y,z (current/average) = .../OFFSET_X .../OFFSET_Y .../OFFSET_Z
#define GYRO_AVERAGE_OFFSET_X ((float) -58.83)
#define GYRO_AVERAGE_OFFSET_Y ((float) 51.11)
#define GYRO_AVERAGE_OFFSET_Z ((float) 7.57)
```

Figura 43: Adicionando ao código os valores de calibragem do giroscópio

B.3.3 Calibrando o Magnetômetro

Por fim, procedemos à calibragem do magnetômetro. O firmware oferece uma calibragem padrão que compensa apenas erros relativos a ferro duro. A calibragem extendida, porém, compensa também os efeitos de ferro macio e, portanto, é adotada. É muito importante que a **IMU seja calibrada no ambiente em que será usada**, pois a calibragem leva em conta o efeito magnético dos objetos próximos. A seguir, os passos são:

- Feche todas as aplicações que leiam dados do sensor. Para poder usar o arquivo, em Processing, que faz a calibragem do magnetômetro, é necessário instalar a biblioteca EJML. Abra o arquivo "Processing/Magnetometer_calibration", onde instruções para instalar a biblioteca são detalhadas. Após a instalação da biblioteca, o programa pode ser executado.
- No modo de calibragem, deve-se tentar mexer a IMU de forma que ela atinja todas as possíveis orientações espaciais. Isso pode ser verificado pelos pontos coloridos que vão sendo plotados na tela do programa.

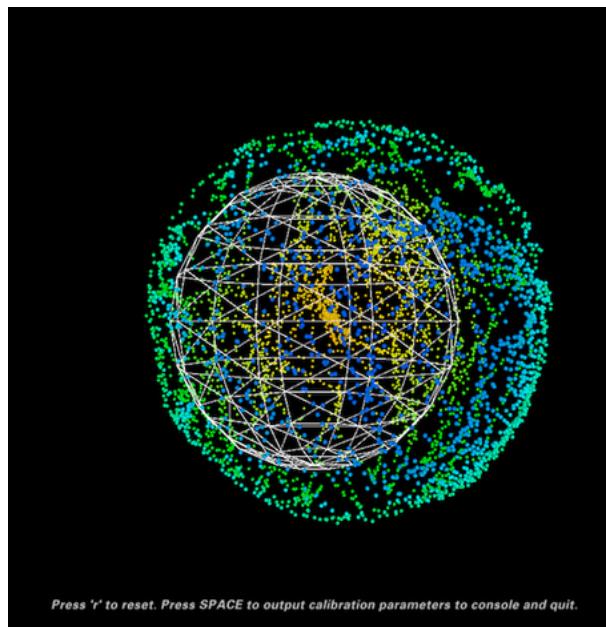


Figura 44: Execução do programa de calibragem do magnetômetro

- Ao adquirir uma quantidade razoável de pontos, como na figura 44, basta apertar a barra de espaço e é gerada a linha de código que se deve colocar no código do firmware.

```
// Magnetometer (extended calibration mode)
// Uncommend to use extended magnetometer calibration (compensates hard & soft iron errors)
#define CALIBRATION_MAGN_USE_EXTENDED true
const float magn_ellipsoid_center[3] = {133.044, -40.5460, -54.8962};
const float magn_ellipsoid_transform[3][3] = {{0.946767, 0.00924318, 0.00298294}, {0.00924318, 0.964504, 0.0169215}, {0.00298294, 0.0169215, 0.990859}};
```

Figura 45: Adicionando ao código os valores de calibragem do magnetômetro

C Configuração do Modem Bluetooth Mate Silver

Em 2.1.3, foi mencionado que é possível, além de trabalhar com a IMU conectada ao PC, ler seus dados através de uma interface bluetooth. Essa interface pode ser qualquer modem bluetooth que permita a si a conexão dos pinos FTDI da IMU. O modem escolhido foi o *Bluetooth Mate Silver*. Nesta seção é mostrado os detalhes sobre a programação desse dispositivo e sua integração com a IMU. Para configurar o modem bluetooth, pode-se conectar-lo ao PC usando a mesma interface FTDI usada para a IMU e usar um programa de terminar como o TeraTerm [26].

A configuração padrão do modem é:

- Bluetooth Slave Mode
- PIN: 1234
- Baud rate de 115200 na porta serial, com 8 bits, sem paridade e 1 stop bit
- Flow control da porta serial desabilitado
- Modo de baixa potência desligado

Com o modem conectado ao PC em uma porta COM específica, e com o TeraTerm (ou similar) aberto, pode-se iniciar a configuração. Alguns pontos importantes:

- Digite \$\$\$ no terminal para entrar em *Command Mode*
- O módulo retorna CMD que indica que a conexão e as configurações do terminal estão corretas.
- Se um comando é válido, o módulo retorna AOK.
- Se for inválido, retorna ERR e se for comando não reconhecido, ?.
- Digite "h <cr >" para ver a lista de comandos (<cr >= carriage return)
- Digite "X <cr >" para ver as configurações atuais.
- Sair do command mode: digite "--- <cr >" ou reseste o módulo e reconecte.

O modem possui sete modos de operação, a saber:

- Slave Mode (SM,0) - padrão. Outros dispositivos bluetooth podem descobrir e conectar ao módulo.
- Master Mode (SM,1) - conexão de baixa velocidade. As conexões são feitas quando um comando de conexão, C, é recebido. Modo usado quando se quer que o módulo inicie conexões ao invés de receber-las. O módulo não pode ser encontrado e não se pode conectar a ele.

- Trigger Mode (SM,2) - conexão de baixa velocidade. Conexões feitas automaticamente quando um caractere é recebido na porta serial (UART).
- Auto-Connect Master Mode (SM,3) - a conexão é feita automaticamente quando o módulo é ligado e refeita se a conexão é perdida. Alta velocidade.
- Auto-Connect DTR Mode (SM,4) - deve ser configurado por comando. Funciona como o anterior.
- Auto-Connect ANY Mode (SM,5) - deve ser configurado por comando.
- Pairing Mode (SM,6) - nesse modo, o módulo tenta conectar com o dispositivo remoto que corresponde ao endereço armazenado. Para configurar o endereço remoto use o comando SR.

O Bluetooth Mate Silver possui três tipos básicos de comandos: SET, GET e CHANGE. A seguir, são mostrados alguns dos mais importantes:

SET COMMANDS

- SM,<value>- security mode. Explicado acima
- SN,<string>- definir o nome do dispositivo
- SP,<string>- definir o código PIN. Default: 1234
- SR,<hex value>- armazena o endereço remoto.
 - SR,Z apaga todos os endereços armazenados
 - SR,I escreve o último endereço observado
 - Exemplo: SR,00A053112233 configura o endereço remoto de Bluetooth para 00A053112233
- ST,<value>- configura o timer de configuração remota, onde value é um valor de 0 a 255 que define o tempo em que a configuração remota por bluetooth após o power up em Slave Mode.
- SU,<value>- configura a baud rate: 1200, 2400, 4800, 9600, 19.2, 28.8, 38.4, 57.6, 115K, 230K, 460K, 921K. Só é preciso especificar os primeiros 2 dígitos do baud rate
 - Exemplo: SU,57 configura o baud rate par 57600.

GET COMMANDS

- D - mostra configurações básicas como endereço, nome, configurações UART, segurança, PIN, bonding e endereço remoto.

- E - mostra configurações extendidas
- GB - retorna o endereço de bluetooth do dispositivo
- GF - retorna o endereço o endereço de bluetooth do dispositivo atualmente conectado
- GK - retorna o status de conexão: 1,0,0 indica conectado e 0,0,0 indica desconectado
- GR - retorna o endereço remoto
- G<char>- mostra as configurações para um set command onde <char>é o set command

CHANGE COMMANDS

- \$\$\$ - entra no command mode
- --- sai do command mode
- C - tenta conectar ao endereço remoto armazenado
- C,<address>- conectar ao endereço especificado e entrar em fast data mode.
- CFI - causa a conexão e entra em fast data mode usando o último endereço encontrado no comando de inquiry
- CFR - causa a conexão e entra em fast data mode usando o endereço remoto armazenado.
- H - mostra o help com a lista de comandos
- IQ - scan por dispositivos e retorna seus RSSI
- K, - desconecta da conexão atual.
- R,1 - reboot

Para o uso específico com a IMU, é bastante simples configurar o modem bluetooth. Basta entrar no modo de configuração, como descrito anteriormente e enviar os comandos seguintes:

- \$\$\$ para entrar em command mode
- SU,57 para baud rate de 57600
- SO,#
- --- para sair do command mode.

Essa configuração habilita o recebimento dos dados da IMU pelo modem bluetooth. Para que esses dados sejam enviados para um computador, é necessário que ambos os dispositivos (PC e modem bluetooth) estejam pareados. O processo de pareamento (PIN padrão do modem: 1234) cria duas portas COM no PC para a conexão de dados: uma *incoming* e outra *outgoing*. Ou seja, a primeira lida com os dados que chegam ao PC e a segunda, com os que saem. A leitura dos dados emitidos pelo modem bluetooth pareado é feita da mesma forma que uma conexão serial.

D Stack Bluetooth do Windows Embedded Compact

O stack do protocolo Bluetooth do Windows Embedded Compact é responsável por permitir que dispositivos bluetooth sejam visíveis e visualizados, podendo estabelecer uma conexão. A Figura 46 mostra como estão organizadas as camadas desse stack.

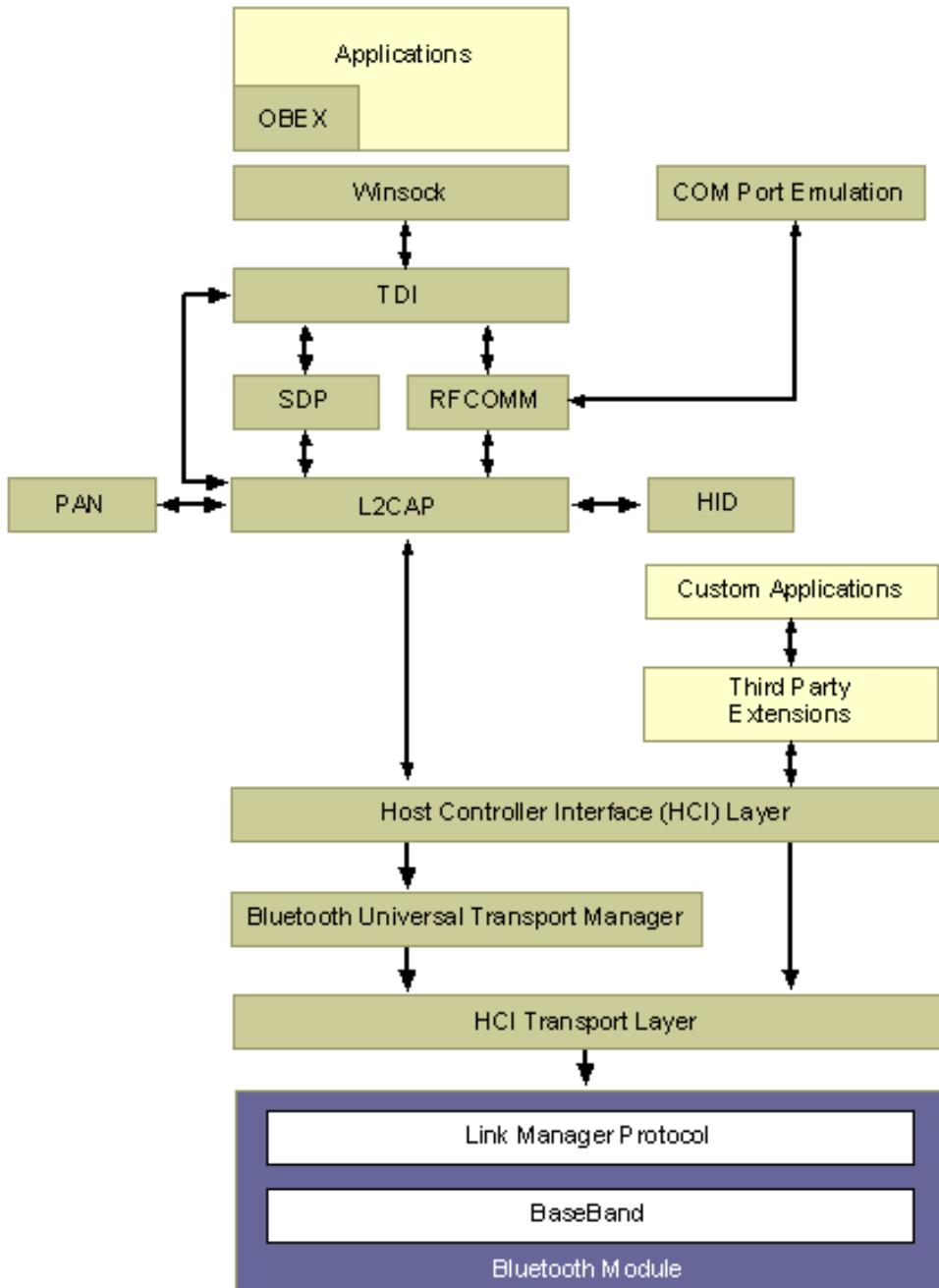


Figura 46: Camadas do stack bluetooth do Windows Embedded Compact. Retirado de [32]

Dessas camadas, algumas merecem destaque por sua relação com o que foi pesquisado nesse projeto:

- **COM Port Emulation:** permite que portas COM virtuais sejam criadas através de canais RFCOMM. Assim, pode-se criar uma porta COM virtual associada a um dongle bluetooth conectado ao módulo. Mais informações a esse respeito são encontradas em [33]. Essa utilidade está inclusa em *Btd.dll*;
- **SDP (Service Discovery Protocol):** protocolo de descoberta de dispositivos bluetooth que lida com a descoberta e publicação dos serviços que estão disponíveis no topo do stack bluetooth. O serviço é diferenciado para cliente e server. Para cliente, está incluso em *Btdrt.dll* e para server, em *Btd.dll*;
- **RFCOMM (Serial Cable Emulation Protocol):** incluso em *Btd.dll*, fornece os recursos para emular a transmissão de dados via serial;
- **HID (Human Interface Device):** define os procedimentos para dar suporte a dispositivos de interface com usuário, como mouse e teclado;
- **HCI (Host Controller Interface):** incluso em *Btd.dll*, é a interface responsável por lidar com o hardware, gerenciando o controle, estabelecendo ligações e manutenção. Durante a sequência de inicialização da interface bluetooth, a HCI criar threads de escrita e leitura, realiza uma conexão com o bluetooth transport e reseta o dispositivo e em seguida lê o tamanho de buffer do dispositivo. Após isso, entra em um estado inicializado, no qual pode receber conexões.
- **HCI Transport Layer:** é a camada de transporte que leva os comandos HCI ao hardware bluetooth.

Referências

- [1] V. B. P. Fernandes, J. A. Frank, and V. Kapila, *A Wearable Interface for Intuitive Control of Robotic Manipulators without User Training*. Proc. 12th Biennial ASME Conf. on Engineering Systems Design and Analysis (ESDA2014), Copenhagen, Denmark, June 2014.
- [2] Toradex, (2014). Disponível em: <http://toradex.com> [Acessado em 30 Junho de 2014].
- [3] FERNANDES, Guilherme. Exploração de ambientes não estruturados através de manipulador robótico implementando controlador de impedância com parâmetros variáveis. 2013. Dissertação (Mestrado em Dinâmica das Máquinas e Sistemas) - Escola de Engenharia de São Carlos, University of São Paulo, São Carlos, 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/18/18149/tde-22072014-174251/> [Acessado em 26 Outubro de 2014]
- [4] Robotics Toolbox. Disponível em: http://www.petercorke.com/Robotics_Toolbox.html [Acessado em 26 Outubro de 2014]
- [5] John J. Craig, *Introduction to Robotics - Mechanisms and Control*. Pearson Education International, New Jersey, 3rd edition, 2005.
- [6] Toradex Developer, (2014). SDK. Disponível em: <http://developer.toradex.com/software-resources/arm-family/windows-ce/development-tools/sdk> [Acessado em 17 Outubro de 2014].
- [7] Toradex Product Download Section - Platform Builder Works-space. Disponível em: <http://developer1.toradex.com/files/toradex-dev/uploads/media/Colibri/WinCE/PBWorkspace/Betas/> [Acessado em 20 Outubro de 2014]
- [8] Toradex Product Download Section - Tegra BSP. Disponível em: <http://developer1.toradex.com/files/toradex-dev/uploads/media/Colibri/WinCE/BSP/CE7/Betas/> [Acessado em 20 Outubro de 2014]
- [9] Toradex - Frequent Downloads. Disponível em: http://developer.toradex.com/frequent-downloads#Windows_CE_Images_Tegra [Acessado em 20 Outubro de 2014].
- [10] Toradex - Txx Recovery Mode. Disponível em: <http://developer.toradex.com/knowledge-base/txx-recovery-mode> [Acessado em 20 Outubro de 2014].

- [11] Toradex - Getting Started with Colibri Modules. Disponível em: <http://developer.toradex.com/knowledge-base/getting-started-with-colibri-modules> [Acessado em 21 Outubro 2014]
- [12] Toradex - Bluetooth. Disponível em: <http://developer.toradex.com/knowledge-base/bluetooth> [Acessado em 21 Outubro de 2014]
- [13] Toradex - USB Bluetooth Flash Installers. Disponível em: <http://developer.toradex.com/knowledge-base/usb-bluetooth-flash-installer> [Acessado em 21 Outubro de 2014]
- [14] Toradex - Wifi. Disponível em: <http://developer.toradex.com/knowledge-base/wifi> [Acessado em 22 Outubro de 2014]
- [15] Toradex - Ambicom Wireless Wifi Dongle Drivers. Disponível em: <http://developer1.toradex.com/files/toradex-dev/uploads/media/Misc/AmbiCom/> [Acessado em 22 Outubro de 2014]
- [16] Toradex - How to send and receive data over Ethernet. Disponível em: <http://developer.toradex.com/knowledge-base/how-to-send-and-receive-data-over-ethernet> [Acessado em 22 Outubro de 2014]
- [17] Toradex - Remote Display. Disponível em: <http://developer.toradex.com/knowledge-base/remote-display> [Acessado em 22 Outubro de 2014]
- [18] Toradex - Iris Carrier Board. Disponível em: <http://developer.toradex.com/product-selector/iris-carrier-board> [Acessado em 30 Outubro de 2014]
- [19] Perso.wanadoo.es, (2014). Robot Manipulador SCARA. Disponível em: <http://perso.wanadoo.es/e/alimapp/scara/> [Acessado em 30 Junho 2014].
- [20] GitHub, (2014). Tutorial - Building an AHRS using the SparkFun "9DOF Razor IMU" or "9DOF Sensor Stick". Disponível em: <https://github.com/ptrbrtz/razor-9dof-ahrs/wiki/Tutorial> [Acessado em 16 Outubro de 2014]
- [21] Wikipedia, (2014). SCARA. Disponível em: <http://en.wikipedia.org/wiki/SCARA> [Acessado em 30 Junho 2014].
- [22] Wikipedia, (2014). Gimbal Lock. Disponível em: http://en.wikipedia.org/wiki/Gimbal_lock [Acessado em 16 Outubro de 2014]
- [23] Wikipedia, (2014). Round-robin scheduling. Disponível em: http://en.wikipedia.org/wiki/Round-robin_scheduling [Acessado em 20 Outubro de 2014]

- [24] Wikipedia. Universal Asynchronous receiver/transmitter. Disponível em: http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter [Acessado em 22 Outubro de 2014]
- [25] Wikipedia - Internet protocol suite. Disponível em: http://en.wikipedia.org/wiki/Internet_protocol_suite [Acessado em 22 Outubro de 2014]
- [26] Ayera Technologies, INC. (2002-2014). TeraTerm 3.1.3. Disponível em: <https://www.ayera.com/teraterm/> [Acessado em 16 Outubro de 2014]
- [27] Sparkfun, (2014). Polymer Lithium Ion Battery - 850mAh. Disponível em: <https://www.sparkfun.com/products/341> [Acessado em 16 Outubro de 2014].
- [28] Sparkfun, (2014). USB LiPoly Charger - Single Cell. Disponível em: <https://www.sparkfun.com/products/12711> [Acessado em 16 Outubro de 2014].
- [29] Microsoft, (2014). Visual Studio 2008 update for Windows Embedded Compact 7. Disponível em: <http://www.microsoft.com/en-us/download/details.aspx?id=11935> [Acessado em 17 Outubro de 2014]
- [30] Microsoft, (2014). Windows Embedded Compact 7 ATL Update for Visual Studio 2008 SP1. Disponível em: <http://www.microsoft.com/en-us/download/details.aspx?id=27729> [Acessado em 17 Outubro de 2014]
- [31] Microsoft - Windows Embedded Compact. Disponível em: [http://msdn.microsoft.com/en-us/library/ee504813\(v=winembedded.70\).aspx](http://msdn.microsoft.com/en-us/library/ee504813(v=winembedded.70).aspx) [Acessado em 21 Outubro de 2014]
- [32] Bluetooth Stack Architecture (Windows CE 5.0). Disponível em: <http://msdn.microsoft.com/en-us/library/ms890956.aspx> [Acessado em 29 Outubro de 2014]
- [33] Creating a Connection to a Remote Device Using a Virtual COM Port (Windows CE 5.0). Disponível em: <http://msdn.microsoft.com/en-us/library/ms881004.aspx> [Acessado em 29 Outubro de 2014]
- [34] Connectify - Turn your PC into a WiFi Hotspot. Disponível em: <http://www.connectify.me/> [Acessado em 22 Outubro de 2014]
- [35] HW group - Hercules SETUP Utility. Disponível em: http://www.hwgroupp.com/products/hercules/index_en.html [Acessado em 22 Outubro de 2014]
- [36] What Is a Socket?. Disponível em: <http://docs.oracle.com/javase/tutorial/networking/sockets/definition.html> [Acessado em 22 Outubro de 2014]

- [37] Arduino - Software. Disponível em: <http://arduino.cc/en/main/software> [Acessado em 23 Outubro de 2014]
- [38] Processing.org. Disponível em: <https://www.processing.org/> [Acessado em 23 Outubro de 2014]
- [39] N. I. Badler, D. Tolani. *Real-Time Inverse Kinematics of the Human Arm*. University of Pennsylvania Scholarly Commons, 1999.
- [40] MathWorks - MATLAB and Simulink. Disponível em: <http://www.mathworks.com/> [Acessado em 30 Outubro de 2014]