THE BIG BOOK OF ROBOTA

A number of fascinating and life-changing works presented in a clear and useable way

Florianópolis, 2013–inf

Edited by

Felippe Schmoeller
Martin Vincent
Patrick J.P
The University of Santa Catarina

$\mathcal{PL}$
2013
ROBOTA

Conteúdo

Capítulo 1

Introdução

1.1 Introdução ao livro.

Esse livro tem como objetivo armazenar todo o conhecimento do grupo ROBOTA, ajudando novos membros e colaborando com o ambiente acadêmico.

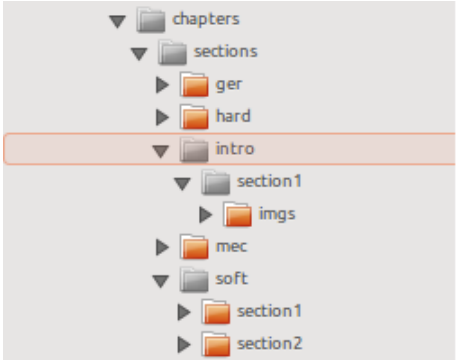
1.1.1 Sobre THE BIG BOOK OF ROBOTA.

Esse livro teve inicio no fatídico dia (04/11/2013) para maior organização do grupo, unir toda a documentação e evitar o retrabalho.

Este livro esta separado em 5 capítulos:

1. Introdução
2. Software
3. Hardware
4. Mecânica
5. Gerencial

Cada capitulo possui uma pasta e cada pasta possui outra para as seções, como mostrado na fugira 1.2.



1.2 Estrura de pastas.

Essa estrutura pode ser detalhada da seguinte forma:
A pasta raiz do THE BIG BOOK OF ROBOTA contem a pasta include, que por sue vez contem a pasta chapters, cada aquivo tex dentro desta pasta possui os caminhos para sections.

Assim, caso alguém queira adicionar mais um arquivo ao grandioso e inigualável THE BIG BOOK OF ROBOTA, é só seguir o que é dito na proxima subsection (1.2.1).

1.2.1 Adicionando sua contribuição.

Bem, vamos lá, primeiramente podes utilizar essa própria section como exemplo, ela está em:
root/include/chapters/sections/intro/section1
Podendo encontrar o arquivo tex, esse seria como um ambiente de trabalho, onde existe o seu tex e uma pasta para imagens da sua section.
Após criar seu tex, vamos adicionar ele ao nosso livro.
Primeiramente vá até: root/include/chapters/
Entre no tex do capitulo que gostaria adicionar seu tex, agora é só adicionar a seguinte linha ao documento:
input{./include/chapters/sections/XXX/sectionX/section.tex}
Onde XXX seria o nome da pasta do capitulo e X o numero da section.
Simplificando seria os seguintes paços:

1. Crie seu .tex
2. Dentro da pasta sections, entre na pasta do chapter que gostaria de adicionar e crie uma pasta sectionX para adicionar seu .tex
3. Após adicionar seu arquivo, inclua o caminho dele dentro da arquivo .tex como exemplificado dentro de intro.tex

4. Não esqueça de quando for compilar, sempre compilar o arquivo main.tex, não o arquivo .tex da section !!

Capítulo 2

Software

Parte de Software, com algumas informações sobre controle de versão, servidores remotos de versão, Arduino e documentação Doxygen.

2.1 Controle de versão

2.1.1 Git

O GIT é um sistema de controle de versão, de código aberto, tendo como um dos desenvolvedores o conhecido Linus Torvalds.

> Eu sou um egoísta degenerado, batizo todos os meus projetos com meu nome. Primeiro Linux, agora Git. — Linus Torvalds

Lembrando que Git é um gíria para cabeça dura, contudo podendo ter outros sinônimos como: Global information tracker, ou, quando ele trava, "Goddamn idiotic truckload of sh*t".

Uma ferramente de controle de versão possui uma série de periféricos para possibilitar sua utilização, nesta parte iremos introduzir algumas delas.

1. Criando
   Para criar um repositório local: git init
   Para clonar um repositório remoto: git clone /caminho/para/o/repositório
   Para clonar um repositório remoto: git clone username@host:/caminho/para/o/repositório ou git clone
   https://github.com/username/repositório.git

2. Adicionando e removendo
   Para adicionar arquivos no repositório a ser verificado: git add nome_do_arquivo
   Para adicionar tudo ao repositório: git add * Para remover um arquivo do repositório: git rm nome_do_arquivo

3. Commit (comentar as mudanças) e sincronizar
   Commitar mudanças: git commit -m “mensagem de commit”
   Atualizar o seu repositório: git pull
   Atualizar o repositório remoto: git push
   Conectar repositório local com um remoto: git remote add origin server

4. Branches (ramos no repositório)
   Criar um novo branche: git checkout -b nome_do_branch
   Trocar do branch para o master: git checkout master
   Deletar branch: git branch -d nome_do_branch

Dar um push do branch para o repositório local: git push origin branch

5. Merges (incorporar mudanças)
   Merge as mudanças de algum branch: git merge nome_do_branch
   Vê as mudanças entre dois branches: git diff branch_fonte branch_alvo

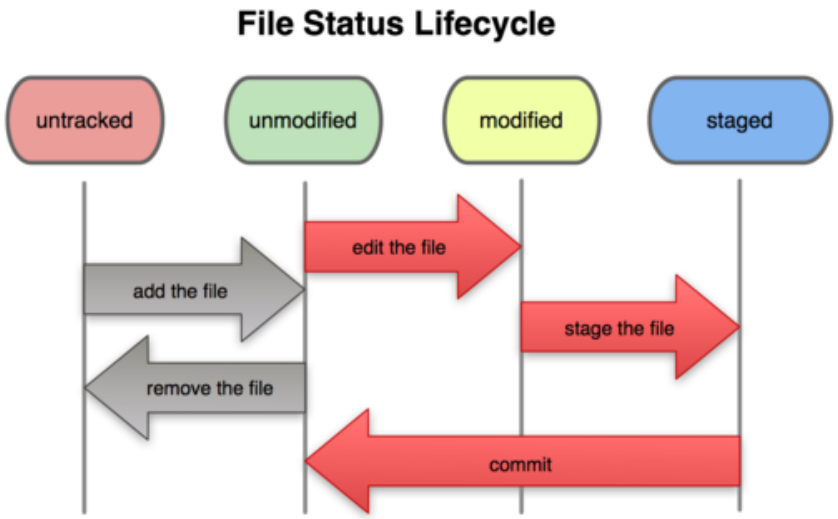


Figura 2.1: Ciclo GIT

A figura 2.1, mostra numa visão muito simplificada a utilização do git. Tendo como início, o arquivo não rastreado, "untracked", temos que iniciar o mesmo no repositório, para isso utilizando o comando:
$ git add nome_do_arquivo, $ git add * ou até mesmo $ git add −−all.

Tendo em mãos o arquivo modificado, organize e comente o mesmo para o commit, fazendo assim uma atualização do arquivo para o gerenciador de versão, desta forma o arquivo ira constar como atualizado na sua maquina para ultima versão, desta forma, basta utilizar o seguinte comando:
$ git commit -m "mensagem", $ git commit e escrever a mensagem em seguida.

Fazendo tudo isso na sua maquina e tendo o arquivo como não modificado, "unmodified", está na de atualizar o servidor remoto, utilizando desta forma o comando:
$ git push origin master

2.1.2 Primeiros passos

1. Configurando.
   Após a instalação do git.
   Vamos fazer algumas configurações do usuario.
   username:
   $ git config –global user.name “seu nome”
   Email:
   $ git config –global user.name “seu_mail@exemplo.com”

   Esses dados são uteis, pois iram junto com seus commits.
   Caso esteja utilizando o GitHub, é altamente aconselhável utilizar o mesmo e-mail para ambos.
   Pode visualizar as modificações utilizando:
   $ git config –list

2. Conseguindo um repositório git.

   a) Inicializando um repositório de um diretório existente.
   $ git init
   Este comando cria um subdiretório .git, que ira conter todos os arquivos necessários para o git.
   Após isso é necessário adicionar os arquivos que você gostaria de serem adicionados ao nosso repositório.
   $ git add README
   Após adicionar nossos arquivos, vamos agora comentar as modificações.
   $ git commit -m “primeiro commit, adicionado os arquivos”
   Terminando isso seu repositório Git esta pronto e atualizado, podendo ser visualizado pelo comando:
   $ gitk

   b) Iniciando um repositório utilizando um existente.
   Em sua essência utilizamos o comando:
   $ git clone url
   Que, por sua vez, clona um repositório ja existente.
   Ex: $ git clone git://github.com/live/4ever.git
   Como o “$ git init”ele também cria uma pasta .git que possui todo o histórico e modificações do repositório.

3. Salvando modificações.
   Para checar o status dos arquivos, utilizamos o comando:
   $ git status
   Se o retorno for: nothing to commit (working directory clean),significa que seu repositório esta atualizado.
   Caso for diferente, devemos atualizar o repositório e comentar as mudanças.

Para rastrear arquivos dentro do diretório, utilizamos o comando:
$ git add
Caso todos os arquivos dentro da pasta estejam no diretório, é recomendado utilizar:
$ git add –all
Arquivos específicos:
$ git add README
Arquivos com extensão especifica:
$ git add *.c

Para poder ver a situação atual do repositório utilizando o comando status do Git, "$ git status", permitindo desta forma visualizar modificações depois do ultimo commit. Utilizando o comando de status, poderemos ver:

Algumas extensões são ignoradas pelo comando "$ git add", geralmente são arquivos de compilação e temporários:
*.o
*.a
*~

Visualizando modificações.
A principal ferramenta para visualizar as modificações do ultimo commit é o comando "$ git diff", mostra os arquivos e linhas modificadas. A visualização é bem instintiva e de grande ajuda para os commits realizados no futuro.

Commitando as modificações.
Após adicionar o arquivo e modificar, vamos agora commitar as modificações, esta ação atualiza todos os arquivos rastreados pelo git,fazendo dessa forma um update do repositório local comentado.
O comando base:
$ git commit
(Este comando utiliza da variável Shell $EDITOR, vim,emacs,nano. Caso não esteja configurado no seu OS, utilize o comando "$ git config –global core.editor").

Abrindo tela, é possível deixar uma mensagem para comentar as atualizações.
Para tornar este comando mais pratico, existe a possibilidade de adicionar a variável -m.
Ex: $ git commit -m "o arquivo README foi modificado"

4. Visualizando o histórico do commit.

Após dar um clone num repositório, podemos visualizar todas os commits realizados durante sua existência utilizando o comando "$ git log".
Esta função do git, possui inúmeras funcionalidades e variáveis para ajudar o desenvolvedor, contudo existi uma ferramenta mais amigável:
$ gitk

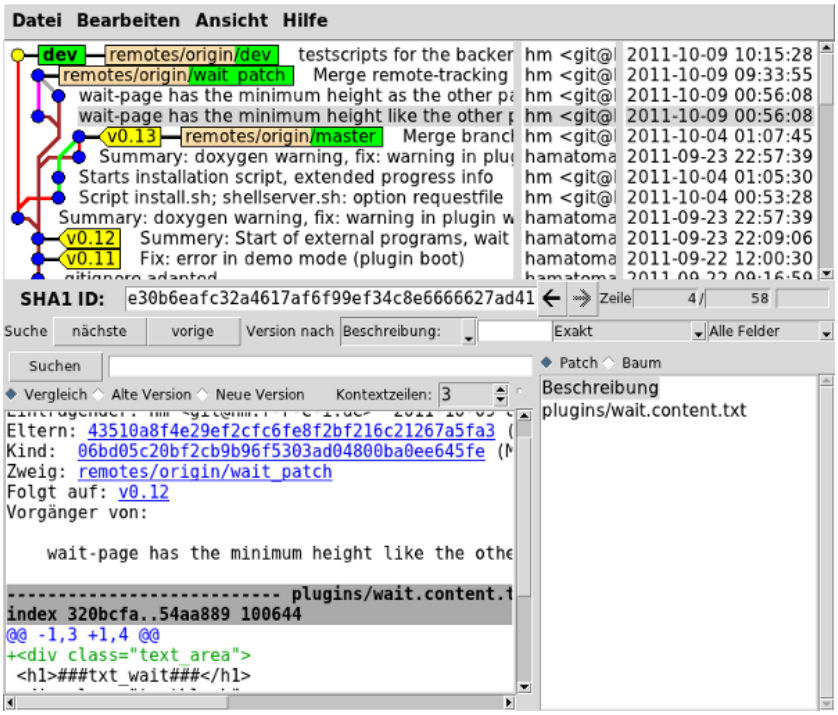


Figura 2.2: Gitk

5. Trabalhando com um repositório remoto.

   Trabalhando num repositório remoto, abre as portas para novas oportunidades e colaboração entre os desenvolvedores.
   Primeiramente, vamos clonar um repositório:
   $ git clone url
   Este comando permite fazer uma cópia do repositório remoto para seu computador.
   Você também pode adicionar a variável -v,podendo mostrar a url que foi retirada o repositório.
   $ git remote -v

   Atualizando o repositório remoto.
   Caso seu repositório local esta pronto, podemos enviar as modificações

para o servidor remoto, utilizando o comando:
$ git push origin master
(Tendo previamente definido.)

6. Marcações.

Utilizado geralmente para marcar a versão que estamos trabalhando no projeto (v1.0 e assim por diante), vamos agora dar alguns exemplos.
A listagem das tags pode ser feita utilizando o comando:
$ git tag
# v0.1
# v1.3
Este comando lista as marcas em ordem alfabética, a ordem em que eles aparecem não possui nenhuma importância.

Criando Marcações.
Pode ser feito marcações (-a) com comentários (-m):
$ git tag -a v1.4 -m 'minha versão 1.4'
$ git tag
# v0.1
# v1.3
# v1.4

Você pode visualizar as informações da marcação, com o comando:
$ git tag show v1.4
# tag v1.4
# Tagger: Scott Chacon schacon@gee-mail.com
# Date: Mon Feb 9 14:45:11 2009 -0800
#
my version 1.4
commit 15027957951b64cf874c3557a0f3547bd83b3ff6
Merge: 4a447f7... a6b4c97...
Author: Scott Chacon schacon@gee-mail.com
Date: Sun Feb 8 19:02:46 2009 -0800
#
Merge branch 'experiment'
#

Pode ser feita uma pesquisa rápida e limitada.
$ git tag-l 'v1.4.2. *'
# v1.4.2.1
# v1.4.2.2
# v1.4.2.3
# v1.4.2.4

Compartilhando marcações.
Por padrão o comando "$ git push"não transfere tags para servidores remotos.
A maneira existente para fazer isso, seria utilizar o comando no formato:
$ git push origin nome_da_tag
Ex: $ git push origin v1.5
Counting objects: 50, done.
Compressing objects: 100% (38/38), done.
Writing objects: 100% (44/44), 4.56 KiB, done.
Total 44 (delta 18), reused 8 (delta 1)
To git@github.com:schacon/simplegit.git
* [new tag] v1.5 - v1.5

Se existe um numero grande de marcações que gostaria de dar push, pode utilizar –tags como opção.
$ git push origin –tags
# Counting objects: 50, done.
# Compressing objects: 100% (38/38), done.
# Writing objects: 100% (44/44), 4.56 KiB, done.
# Total 44 (delta 18), reused 8 (delta 1)
# To git@github.com:schacon/simplegit.git
# * [new tag] v0.1 - v0.1
# * [new tag] v1.2 - v1.2
# * [new tag] v1.4 - v1.4
# * [new tag] v1.5 - v1.5

2.1.3 Ramificações ou Branchs

O Branch pode ser simplesmente definido como:

> Um branch no Git é simplesmente um leve ponteiro móvel para um desses commits. O nome do branch padrão no Git é master. Como você inicialmente fez commits, você tem um branch principal (master branch) que aponta para o último commit que você fez. Cada vez que você faz um commit ele avança automaticamente. — git-scm

A figura 2.3 exemplifica como funciona na pratica, o branch da inicio no ultimo commit e quando terminado o objetivo dele, as modificações são aplicadas ao master.

Para criar um Branch basta utilizar o comando:
$ git checkout -b nome_do_branch
$ git branch nome_do_branch
Ex: $ git checkout -b nice_feature
Ex: $ git branch nice_feature

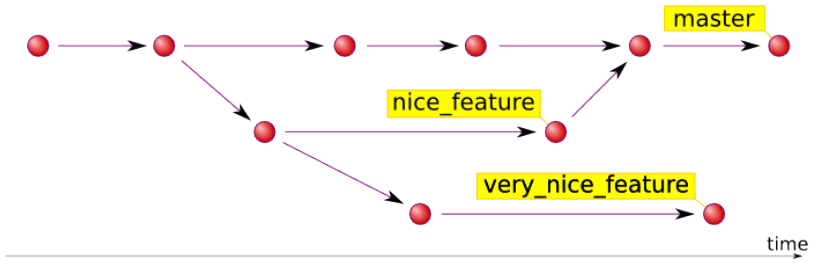


Figura 2.3: Exemplo de Branch

Para nosso diretório apontar para o novo branch, devemos utilizar o comando:
$ git checkout nome_do_branch
Ex:$ git checkout nice_feature

Como pode ser visto na figura 2.4, podemos ver que com o checkout mudamos o local do ponteiro onde estamos trabalhando, desta forma, as modificações serão feitas aonde o apontador esta localizado.

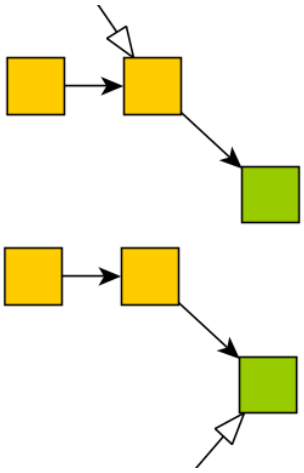


Figura 2.4: Exemplo checkout

Após as modificações serem realizadas, esta na hora de integrar as modificações com o branch master.

Primeiramente, devemos estar com o nosso apontador no master:
$ git checkout master

Em seguida, fazer o merge e atualizar nosso master com a nice_feature:
$ git merge - -no-ff nice_feature
# Merge made by recursive.
# README — 1 +
# 1 files changed, 1 insertions(+), 0 deletions(-)

É recomendado que utilize o comando "$ git merge - -no-ff nice_feature"com a opção - -no-ff, pois:

> Isso evita a perda de informações sobre a existência histórica de um ramo e suas características, junto com todos os commits. — nvie

A figura 2.5 exemplifica bem a diferença.

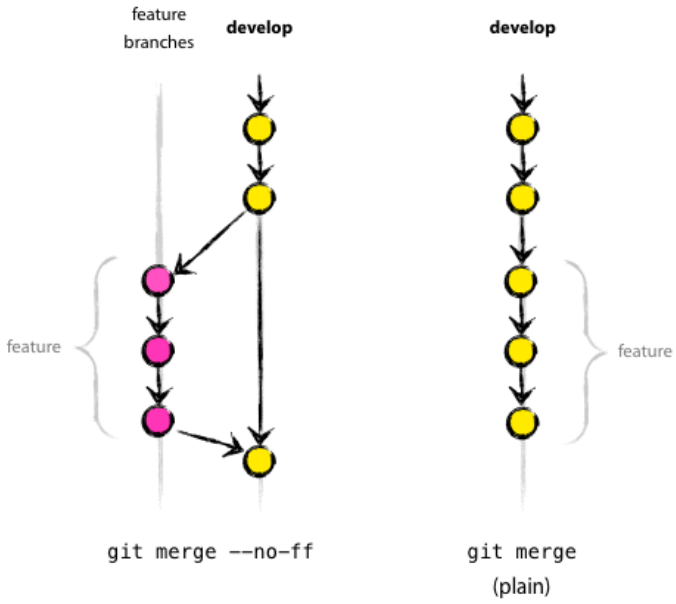


Figura 2.5: Exemplo da opção - -no-ff

Com a atualização do seu master, não existe mais a necessidade de ter ainda em paralelo o branch, assim podemos deletar o mesmo, utilizando o comando:
$ git branch -d nome_do_branch
Ex: $ git branch -d nice_feature

Conflitos de Merge.

Como visto na figura 2.4, temos em paralelo o branch nice_feature com o branch very_nice_feature. Contudo, pode acontecer alguns problemas, se for alterada a mesma parte do mesmo arquivo de forma diferente nos dois branches, o Git não será capaz de executar o merge de forma clara, o conflito pode ser

demostrado com:
$ git merge very_nice_feature
# Auto-merging index.html
# CONFLICT (content): Merge conflict in index.html
# Automatic merge failed; fix conflicts and then commit the result.

Todo conflito de merge que não foi resolvido é listado como “unmerged”no comando “$ git status”no branch master.
$ git status
# index.html: needs merge
# On branch master
# Changes not staged for commit:
# (use ”git add file...”to update what will be committed)
# (use ”git checkout – file ...”to discard changes in working directory)
#
unmerged: index.html
#

Gerenciamento de Branch.

Para ter uma lista de branchs existentes no repositório, é possível utilizar o comando:
$ git branch
# nice_feature
# * master
# very_nice_feature
O simbolo mostra aonde o apontador est'łocalizado.
Para ver o último commit feito em cada branch, pode ser executado o comando:
$ git branch -v

2.2 Criando um servidor remoto de versão

2.2.1 Criando o server git

Primeiramente, tenha um server remoto.
Em seguida:

1. Adicione os usuarios que iram trabalhar.
   $ sudo adduser robota

2. Crie um grupo de desenvolvimento.
   $ sudo addgroup developers

3. Edite o grupo para adicionar os desenvolvedores.
   $ sudo nano /etc/group

Encontre a linha que contenha: "developers:XXXX". Adicionando os usuarios "developers:XXXX:robota,usuario2"

4. Agora, crie uma pasta para o projeto, de preferência na raiz da maquina.
   $ sudo mkdir git

5. Crie a pasta do projeto dentro da do git (/git/projeto.git).
   $ sudo mkdir projeto.git

6. Mude as permissões para o grupo.
   $ sudo chgrp developers projeto.git

7. Mude as permissões para escrita e leitura.
   $ sudo chmod g+rws projeto.git

8. Agora iniciamos o repositório com pro git, dentro da pasta do projeto.
   $ sudo git init –bare –shared

2.2.2 Configurando o pc do desenvolvedor

Primeiramente, tenha acesso ao pc do desenvolvedor.

1. Crie uma pasta para o git.
   $ mkdir git

2. Adicionando o master.
   $ git remote add origin robota@host:/git/projeto

Pronto, agora temos um server funcionando e um pc configurado para le utilizar.

2.3 Arduino

O Arduino definido como uma plataforma eletrônica open-source baseada em hardware e software fáceis de usar. Sua simplicidade e flexibilidade tem permitido uma enormidade de projetos saírem do papel constantemente.

2.3.1 Adicionar Bibliotecas do Arduino

Com a revolução que o Arduino participa e promove, o uso de microcontroladores para diversos fins cresce exponencialmente. Seja automação residencial,

devaneios de hobbystas ou robótica (móvel ou não), a versatilidade destes controladores é constantemente expandida.

Com certos processos de maior complexidade pode ser interessante adicionar um protocolo, criado em .h e .c (ou .cpp), como biblioteca para o Arduino, pois um grande número de funções é comum para uma larga gama de projetos que a versatilidade do microcontrolador permite criar. Quando há um protocolo bem estabelecido como o de comunicação entre Arduino e PC, por exemplo, a adição do protocolo como biblioteca torna o desenvolvimento de aplicações mais simples, rápido e compreensível.

Passos

Os seguintes passos são necessários para a adição de bibliotecas:

1. Abrir a IDE do Arduino;

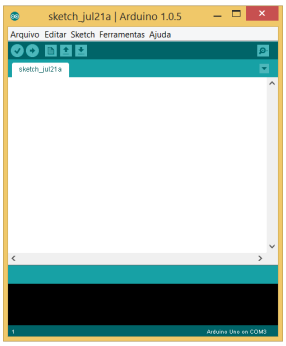


Figura 2.6: IDE Arduino

2. No Menu Sketch, selecione a opção Add Library;

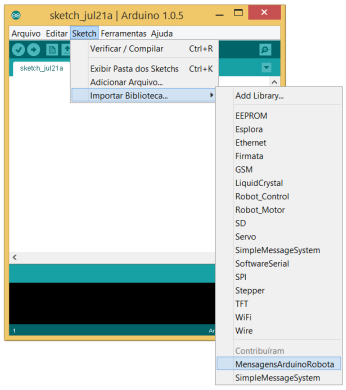


Figura 2.7: Adicionar Biblioteca

3. Pronto. A biblioteca está adicionada. Agora é possível inclui-la no projeto com um simples include, como por exemplo

```
#include <MensagensArduinoRobota.h>
```

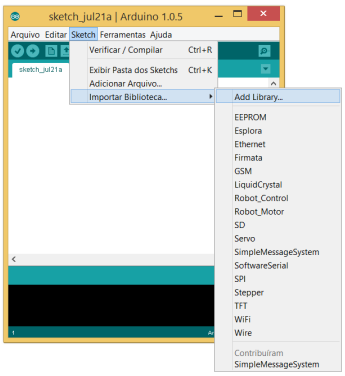


Figura 2.8: Biblioteca adicionada

Outra opção, um tanto menos sutil, é adicionar a pasta da biblioteca na pasta libraries no diretório onde o Arduino está instalado.

2.3.2 Biblioteca MensagensArduinoRobota

A Biblioteca "MensagensArduinoRobota"oferece aos usuários uma simples e intuitiva maneira de controlar um Arduino pelo computador.

Funcionalidades

As seguintes funcionalidades estão disponíveis:

- Ler pinos digitais do Arduino;
- Ler pinos analógicos do Arduino;
- Chavear pinos digitais do Arduino;
- Escrever valores nos pinos de PWM do Arduino;

Comandos

Os comandos funcionam da seguinte forma:

O Arduino recebe uma string, pega a primeira letra de cada palavra para identificar o comando desejado e executa os comandos.

Os comandos resumem-se a:

- r $\rightarrow$ leitura

- w → escrita
- a → analógico
- d → digital
- t → todos
- p → pino especifico para leitura

Exemplos de comandos aceitos

Alguns exemplos de comandos aceitos pela biblioteca:

- r a t → Le todos os pinos Analogicos
- r a p [pin] → Le valor do pino analogico especificado
- r d t → Le todos os pinos digitais
- r d p [pin] → Le valor do pino digital especificado
- w d [pin] [value] → write digital pin
- w a [pin] [value] → write analog pin

Bibliografia

Para a criação da presente biblioteca, os seguintes protocolos foram utilizados como exemplo:

- Protocolo Firmata
- Protocolo CmdMessenger
- Protocolo SimpleMessagingSystem

2.3.3 Visual C++ e Portas Seriais

A utilização do Visual C++ para usos com o Arduino não é a maneira mais convencional encontrada, mas é uma possibilidade para alguns usuários. Existem diversas implicações negativas na escolha dessa linguagem de programação, mas existem alguns pontos positivos também. Apesar de não se ter a liberdade e praticidade que se tem ao trabalhar com periféricos via USB em S.O.s como o Ubuntu, o Visual C++ é uma alternativa funcional para utilizar C++, periféricos USB e o sistema Windows.

Exemplo Visual C++ ⇒ Porta Serial

A seguir, exemplos simples do C++ no Windows para o controle de um Arduino conectado em uma porta USB de um PC.

O primeiro passo é adicionar uma biblioteca bastante importante para o uso de C++ e o Windows.

```
// Biblioteca da malvada Microsoft para fazer tudo funcionar
#include "stdafx.h"
```

Com a biblioteca adicionada, é interessante definir os namespaces utilizados.

```
using namespace System;
using namespace System::IO::Ports;
```

Agora, é possível definir a porta utilizada. No caso, a porta serial será chamada de Arduino para facilitar o processo.

```
// Configuracao do Arduino
        int baudRate = 9600;
        portName = "COM3";
        SerialPort^ arduino;
        arduino = gcnew SerialPort(portName, baudRate)
```

É importante definir limites de tempo para a leitura e escrita de dados do Arduino.

```
        arduino->ReadTimeout = 1500;
        arduino->WriteTimeout = 1500;
```

Para escrever algo na porta serial.

```
        arduino->WriteLine("Algo");
```

Para ler algo da porta serial.

```
        arduino->ReadLine();
```

2.4 Documentação Doxygen e o Doxywizard

O Doxygen é um gerador de documentação, ou seja, uma ferramenta utilizada para a redação de documentos acerca de um software. O conteúdo da documentação é escrito no corpo do código, o que facilita o referenciamento de partes do código no documento.

2.4.1 Uso

Além de aceitar a sintaxe do Javadoc, o Doxygen suporta marcações de documentação usadas no Qt toolkit e a sua saída pode ser em HTML, CHM, RTF, PDF, LaTeX, PostScript ou man pages (páginas de manual).

2.4.2 Doxywizard

Doxywizard é uma interface para a configuração e o uso do Doxygen. Ao ser inicializado, o Doxywizard mostrará sua janela principal. Algumas diferenças visuais poderão ser notadas nos diferentes Sistemas Operacionais utilizados (caso suportados).

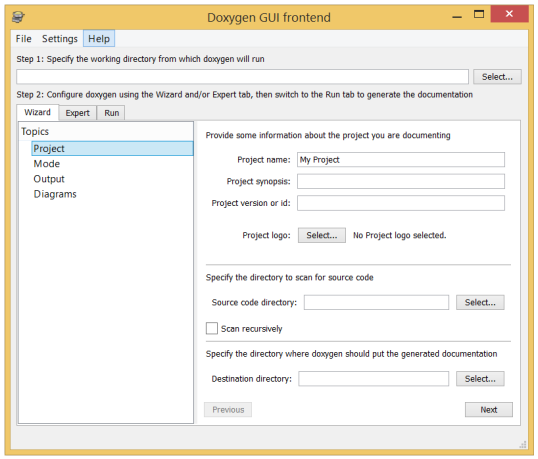


Figura 2.9: Janela Principal

Esta janela mostra os passos para realizar a configuração e a utilização do Doxygen. O primeiro passo é escolher uma das opções de configuração:

- Wizard: Utilizado para configurar rapidamente as opções mais importantes e manter o restante das opções em seus padrões;
- Expert: Utilizado para ter acesso a todas as opções configurações;
- Run: Utilizado para gerar a documentação após realizadas as configurações.

É possível (e comum) realizar uma configuração básica utilizando o Wizard e realizar ajustes específicos no modo Expert.

Wizard

A aba Wizard comummente está selecionada ao iniciar-se o Doxygen, podendo ser vista na Figura 2.9.

Project

Quando selecionada, a aba Wizard permite acesso às seguintes opções:

- Determinar informações como nome, uma pequena sinopse, versão do código e logotipo do projeto;

- Especificar o diretório do código;
- Especificar o diretório alvo do documento gerado.

Modes

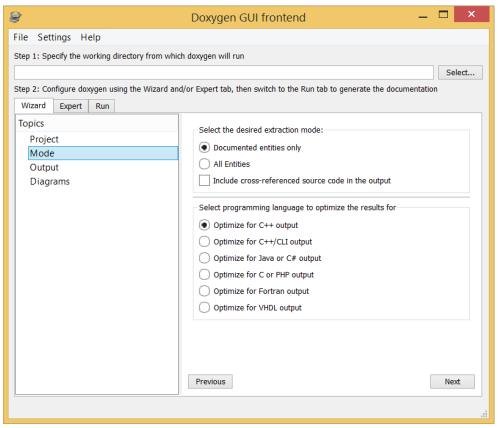


Figura 2.10: Modes

No tópico Modes é possível configurar algumas opções referentes à linguagem de programação utilizada.

Output

Tópico onde são realizadas configurações (como formato, cores e tipos) do documento gerado.

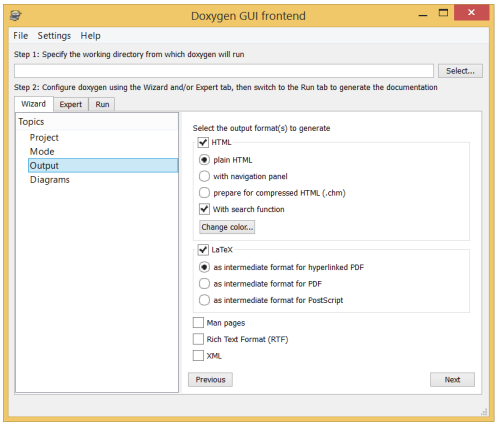


Figura 2.11: Outputs

Diagrams

Tópico relacionado à diagramas.

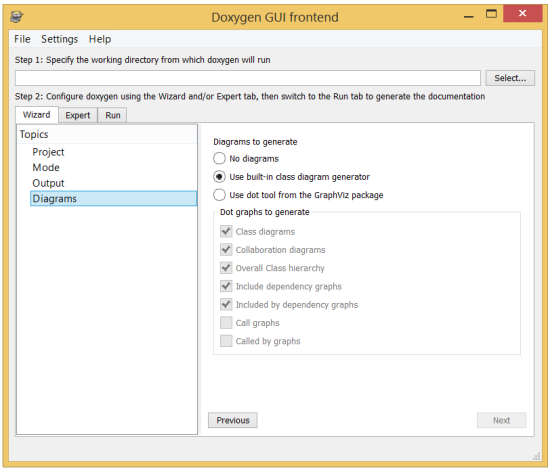


Figura 2.12: Diagramas

Expert

A aba Expert oferece opções de configuração mais detalhadas e avançadas.

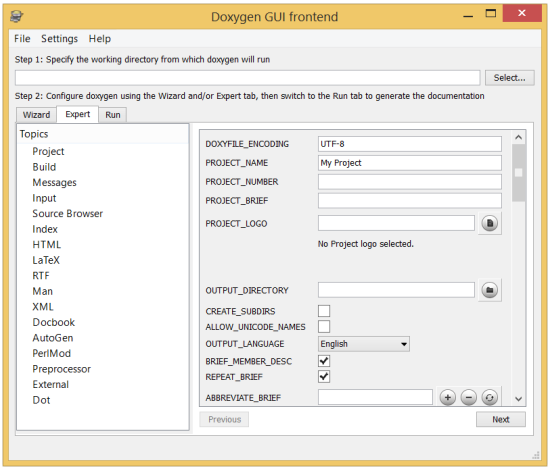


Figura 2.13: Expert

Run

A aba Run permite a geração da documentação.

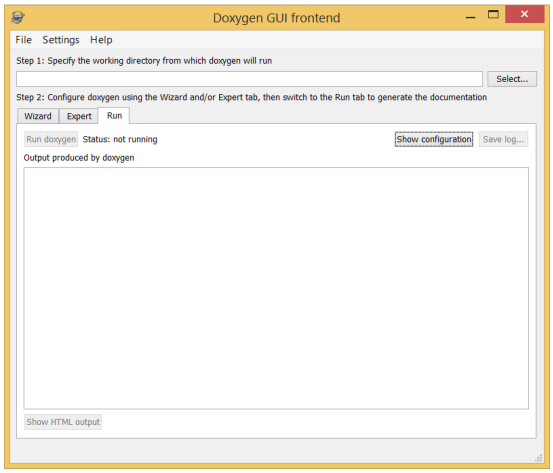


Figura 2.14: Run

Capítulo 3

Hardware

Aqui seria a parte de Hardware

3.1 Básico

Aqui serão apresentados alguns dos conceitos básicos necessários para a compreensão das principais aplicações de eletrônica em robótica móvel.

3.1.1 Elementos básicos

1. Resistor: O elemento mais utilizado na confecção de circuitos eletrônicos, é um componente chave pois permite limitar a corrente elétrica que passa por elementos em série, e possibilita a aplicação de uma tensão desejada através do conceito de divisor de tensão.
   Por definição é um componente que oferece uma oposição à passagem de corrente elétrica, dissipando energia através do efeito Joule.

2. Capacitor: O capacitor é um elemento que armazena energia na forma de campo elétrico, criando uma diferença de potencial entre os seus terminais. Apesar de aparentemente possuir aplicação apenas em circuitos em corrente alternada, é de extrema importância nos circuitos de corrente contínua. Sua aplicação mais utilizada é na filtragem dos sinais, ajudando a evitar ripples e interferências indesejadas. Nesse âmbito pode-se citar os capacitores de desacoplamento, que geralmente ficam em paralelo com a alimentação dos chips eletrônicos, e evitam variações de tensão ou interferências sobre o mesmo. Um valor típico de capacitor de desacoplamento é de 100nF.

3. Diodo: O diodo é um elemento composto de cristal semicondutor, tipicamente de germânio ou silício, sendo o mais simples dos elementos semicondutores, por ser composto apenas por um cristal N e um cristal P. Suas aplicações mais importantes estão relacionadas com sua característica de

conduzir apenas quando for polarizado de forma direta, ou seja, a corrente passa apenas em um sentido pelo diodo (o que explica a sua simbologia, uma seta, que aponta para o sentido da corrente passante). Outra característica importante é a queda de tensão que ocorre entre os seus terminais, que é tipicamente de 0,7V (no cristal de silício, o mais utilizado), e que varia dependendo do material semicondutor empregado.
Uma aplicação clássica é a ponte de diodos, que permite que uma tensão alternada passe a ter apenas tensões positivas (faz um rebatimento dos semi-ciclos negativos no eixo x). Esse sinal pode ser então regulado e filtrado (utilizando reguladores da família 78XX e capacitores, por exemplo). Esse é o princípio de funcionamento de fontes lineares (que convertem a tensão da rede em tensão contínua).

4. Transistor: O transistor é um componente eletrônico que revolucionou a eletrônica por volta dos anos 60, por apresentar uma alternativa mais compacta às válvulas e relés, além de possuir uma autonomia muito maior por ter seu funcionamento baseado nos efeitos de campo do seu material semi-condutor. Pode ser utilizado como chave (deve estar trabalhando na zona de saturação) ou como amplificador (deve estar trabalhando na zona linear).

3.1.2 Aplicações em Robótica Móvel

A eletrônica se mostra de extrema importância no âmbito da robótica móvel, estando presente em praticamente todos os módulos que juntos compõe um robô. Logo abaixo estão listados alguns exemplos de aplicação de elementos e circuitos eletrônicos em um robô.
Primeiramente haverá uma abordagem sobre sensores e atuadores, que basicamente são elementos que possibilitam todo tipo de interação do robô com o ambiente, cada vez mais necessária à medidada em que a "inteligência"dos robôs aumenta. Logo após serão abordados os microcontroladores e computadores que são cada vez mais embarcados nos robôs (e em tudo o que se possa pensar no mundo), e são como um cérebro do robô (talvez não seja o termo mais adequado, já que com a diminuição nos preços de microcontroladores o processamento de dados está sendo mais distribuido em processadores próximos aos diversos elementos do robô).

1. Sensores: Sensores são dispositivos sensíveis à fenômenos físicos ou químicos, transformando-os em uma grandeza mensurável. Nos seres vivos notamos a existência de diversos sensores, permitindo a interação do organismo com o meio exterior (a visão, por exemplo utiliza sensores para interpretar as cores através da frequência dos fótons de luz, os cones, e outros sensores para a luminosidade através da quantidade de fótons de luz, os bastonetes). Quando o sensor converte esse fenômeno em um sinal elétrico (de tensão ou corrente), ele é chamado de transdutor. Em suma, todos os sensores utilizados em robótica móvel são transdutores, logo necessitam

de um circuito eletrônico para interpretar a grandeza medida, podendo ser apenas um comparador analógico utilizando amplificadores operacinais ou um circuito microcontrolado para tratar o sinal de saída do sensor. Atualmente a segunda opção, microcontrolada é amplamente utilizada, devido à redução dos preços dos microcontroladores, e às vantagens relacionadas, como a transmissão de dados de forma digital (sinais alanógicos são mais sucetíveis a ruídos e interferências). Uma tendência já amplamente aplicada é a de sensores inteligentes, que são sensores com um microcontrolador embarcado, agregando funcionalidades interessantes, como a digitalização dos dados, e possibilitando outras, nem sempre utilizadas, como a calibração do sensor, supervisão de valores limite, compactação de dados e eliminação de redundâncias, filtragem digital do sinal, correção automática de erros sistemáticos (não linearidade, dependências de temperatura, etc).[**?**]

2. Atuadores:

   > Atuador é um elemento que produz movimento, atendendo a comandos que podem ser manuais, elétricos ou mecânicos. Como exemplo, pode-se citar atuadores de movimento induzido por cilindros pneumáticos (pneumática) ou cilindros hidráulicos (Hidráulica) e motores (dispositivos rotativos com acionamento de diversas naturezas).
   > - http://pt.wikipedia.org/wiki/Atuador

   Dentre estes atuadores citados, os que envolvem diretamente conceitos e circuitos eletrônicos são os motores, do tipo elétrico, sendo também os mais utilizados no âmbito da robótica móvel.

3. Microcontroladores: Um microcontrolador é um pequeno computador com diversos periféricos dentro de um único chip. Enquanto um microprocessador (como os que encontramos dentro dos PC's) apresenta apenas a ULA (unidade de lógica aritmética) e registradores, o microcontrolador apresenta memórias EEPROM e RAM, conversores analógico-digital, interfaces de entrada-saída, clock interno e outros elementos. Obviamente por apresentar vários elementos dentro do mesmo chip, eles não são tão eficientes quanto um microprocessador, e trabalham em frequência muito mais baixa, na ordem de MHz, porém são extremamente práticos por não necessitarem de periféricos externos para funcionar, serem de baixo custo e permitirem a gravação dos programas diretamente no chip.
   Existem microcontroladores com diversas capacidades de processamento, o que também influencia seu preço. Primeiramente temos os processadores de 8 bits, utilizados em aplicações que não exigem muito poder de processamento, como o tratamento de um sinal analógico (de um sensor, por exemplo) ou o controle de um motor. Nessa categoria os microcontroladores mais populares são os PICs da família 16F e 18F, que são microcontroladores de arquitetura RISC (reduced instruction set computing), que

funcionam em frequências de cerca de 4MHz a 20Mhz e possuem diversos periféricos como conversores AD, protocolos de comunicação implementados em hardware (USART, I2C, SPI e no caso de alguns 18Fs, USB), PWM (pulse width modulation), etc.

4. Computadores:

3.2 Placa de Circuito Impresso

Nesa seção será abordado um tema de extrema importância para a eletrônica, que é relativo às placas de circuito impresso, que a partir de agora serão chamadas de PCB, do inglês 'printed circuit board'. Saber confeccionar uma PCB significa transformar um circuito desenhado no papel em realidade, e envolve uma vasta gama de conhecimento, que será (à medida do possível) condensado e resumido aqui.

3.2.1 O que é?

A PCB é uma placa, geralmente de fenolite, podendo ser também de fibra de vidro ou de poliéster, com uma fina camada de cobre que é corroída de forma a criar caminhos para conectar eletricamente componentes eletrônicos (chamados de trilhas), formando assim o circuito eletrônico desejado. A PCB foi criada para substituir a ponte de terminais, que era utilizada anteriormente, e que impedia a implementação de circuitos mais complexos, devido aos diversos problemas de interferência e pela própria disposição física dos componentes, que muitas vezes beiravam o caos. Outro problema encontrado nessa técnica arcaica era a produção em série dos circuitos, um processo difícil de ser mecanizado e bastante meticuloso.
Inicialmente o desenho do circuito na placa (layout) era feito manualmente, com o auxílio de réguas que continham os moldes dos principais componentes utilizados, e as trilhas eram desenhadas à mão com uma caneta permanente ou com o auxílio de fitas adesivas. Obviamente hoje em dia existem softwares de computador, chamados CADs (computer aidded design) que possibilitam esse projeto. Após possuir o layout pronto, pode-se mandar produzir as placas em alguma empresa especializada, que utiliza ferramentas avançadas possibilitando placas complexas e com componentes pequenos, ou pode-se utilizar métodos artesanais para produzir essa placa, que acaba limitando a complexidade do projeto, porém é suficiente para várias aplicações interessantes.

3.2.2 Elementos básicos

Aqui serão apresentados alguns conceitos e termos utilizados de forma a possibilitar o entendimento dos tópicos subsequentes.

1. Trilha: As trilhas são 'caminhos' de cobre que fazem as ligações elétricas necessárias nas placas. Sua largura está relacionada com a corrente que

deve passar por ela, devendo ser gradativamente mais larga ao se esperar maiores valores de corrente, e esta deve ser uma preocupação do responsável pelo layout. O risco de subdimensionar a largura da trilha é de, quando uma corrente muito grande passar, o cobre esquentar muito e descolar da placa, podendo até mesmo fundir e romper. Isso ocorre pois a resistência elétrica associada é inversamente proporcional à largura da trilha, e pela lei de Joule o calor dissipado é maior com o aumento da resistência (mantendo-se a corrente passante).

2. Camadas (layers): As camadas de cobre presente em uma PCB são chamadas layers. As placas mais simples possuem apenas uma camada de cobre e são chamadas single-layer, sendo as mais utilizadas por hobbistas que utilizam processos caseiros de fabricação das placas. Quando a placa possui duas camadas de cobre é chamada de dual-layer, e pode também ser confeccionada com métodos caseiros, porém com mais dificuldade, pois deve-se projetar o layout para os dois lados e depois sincronizá-los para que os furos coincidam. Quando a placa possui mais de duas camadas é chamada multi-layer, e só pode ser feita através de processos industriais, com a utilização de materiais e máquinas específicas para tal.

3. Componentes Through-Hole: São componentes que se utilizam de pinos metálicos que devem atravessar a placa através de um furo, podendo ser metalizado ou não, sendo soldados no lado oposto ao componente.

4. Componentes SMD: Os componentes SMD (surface mount devices, ou dispositivos de montagem superficial), são componentes que são soldados diretamente sobre a face de cobre da PCB, sem utilizar furos para sua fixação mecânica.

5. Ilhas (pads): As ilhas são superfícies de cobre onde os componentes são soldados. Geralmente são redondas para componentes through-hole e quadradas para os SMD.

3.2.3 CADs

Os CADs (computer aidded design) são softwares que auxiliam no projeto e desenho técnico nas mais diversas áreas, como no projeto de construções, de peças mecânicas e de PCB's, substituindo a necessidade de utilizar os processos manuais para o desenho das plantas, vistas e layouts, respectivamente. Os principais softwares do mercado oferecem ferramentas que facilitam o processo, reduzindo drasticamente o tempo de projeto e de eventuais reparos em um design já existente.

3.3 Softwares

Nesta seção serão apresentados os principais softwares utilizados em aplicações para eletrônica. Tais softwares são extremamente úteis para a realização de um

projeto, principalmente para o projeto de placas de circuito. Esses softwares são chamados CAD's (computer aided design, ou desenho assistido em computador em português) e geralmente permitem o projeto tanto do esquemático quanto do layout da placa, também chamado PCB (printed circuit board, ou placa de circuito impresso em português) design. Outros softwares são utilizados pra realizar a simulação de circuitos eletrônicos, acabando com a necessidade de realizar certos cálculos manualmente, e em alguns casos agregando aspectos práticos dos elementos, como indutâncias, capacitâncias e resistências que não existem nos modelos ideais. Alguns softwares agregam todas essas funcionalidades, como é o caso do Proteus. Abaixo estão listados alguns dos principais softwares utilizados, com uma descrição de cada um.

3.3.1 Proteus

Desenvolvido pela empresa inglesa Labcenter Electronics, o Proteus é provavelmente o software mais utilizado por hobistas ao redor do mundo, principalmente pela possibilidade de simulação de diversos microcontroladores, como os PICs da microchip e Atmegas da Atmel, além de vários ARMs. O Proteus Design Suite é dividido em dois softwares, o ISIS e o ARES. O primeiro é o programa responsável pelo desenho dos esquemáticos, além das simulações de circuitos eletrônicos.

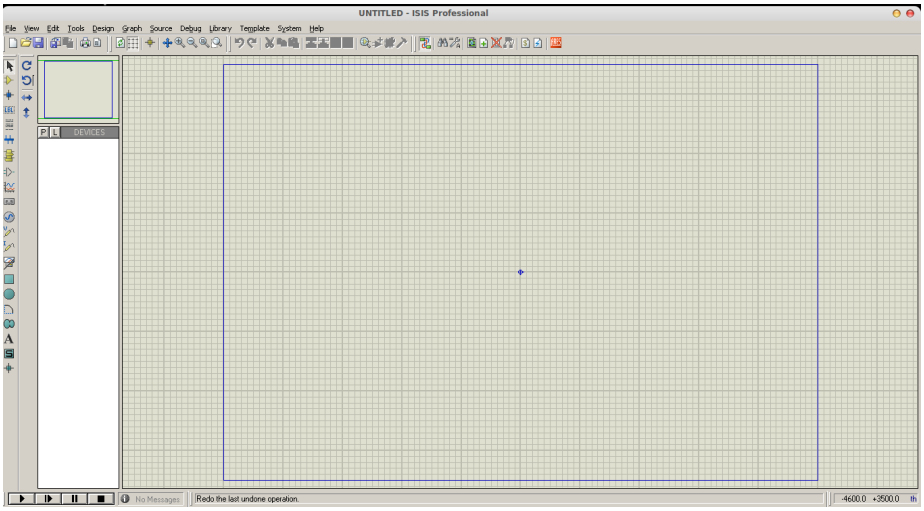


Figura 3.1: Tela inicial do ISIS

Já o Ares é utilizado para a criação do layout da PCB. Possui um auto-router, que roteia as trilhas automaticamente, utilizando as conexões do esquemático criado no ISIS. Outra função interessante do Ares é a possibilidade de uma visualização em 3D da placa em desenvolvimento, desde que os componentes utilizados possuam um modelo em 3D.

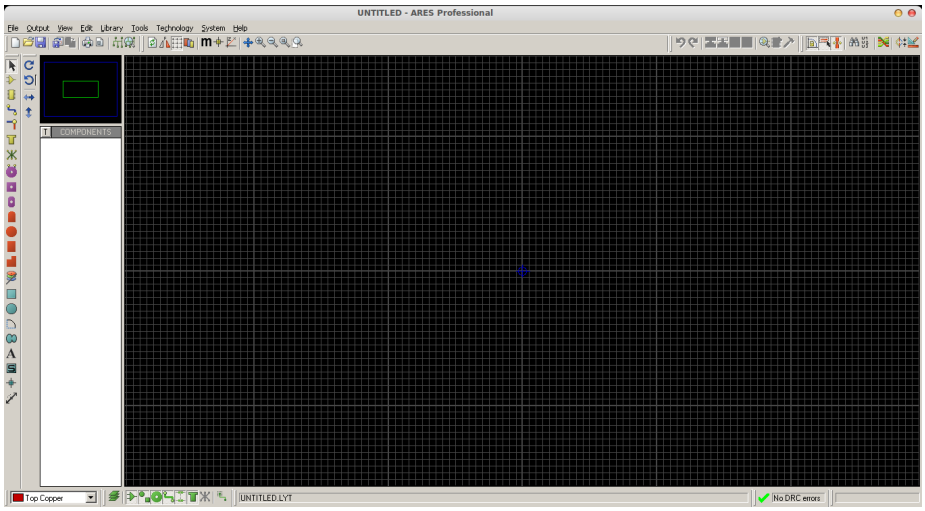


Figura 3.2: Tela inicial do Ares

3.3.2 Eagle

O Eagle (sigla de Easily Applicable Graphical Layout Editor) é um CAD desenvolvido pela empresa CadSoft Computer com suas primeiras edições jurássicas feitas para rodar no DOS, ainda como um aplicativo de 16 bits. Atualmente possui versões para Windows, Linux e Mac OS. Possui um editor para desenhar os esquemáticos de cirucitos eletrônicos, um editor para criar o layout das placas de circuito impresso, auto-router além de diversas outras ferramentas.
Apesar de ser um programa pago, é o software mais utilizado entre usuários adeptos ao open source, pois sua versão freeware possui apenas uma limitação quanto ao tamanho do projeto (que não é problema em aplicações usuais), mas principalmente por possuir uma gigantesca biblioteca com os footprints dos mais variados componentes na rede, construída pelos inúmeros hobistas que utilizam o programa.
É importante ressaltar que o Eagle é um programa que possui uma curva de aprendizado grande, porém por possuir muitos adeptos é fácil encontrar material na internet que possa ajudar a utilizá-lo.

3.3.3 Altium Designer

Possivelmene um dos mais completos programas para desenvolvimente de projetos eletrônicos, sendo desenvolvido pela empresa Australiana Altium Limited, é baseado no famoso Protel, um software muito utilizado até o começo dos anos 2000's. Possui a mais variada miríade de funcionalidades e ferramentas para o desenvolvimento de placas de circuito impresso e FPGA's. É recomendado para projetos complexos, lidando bem com placas com múltiplas camadas e componentes SMD. Possui um poderoso visualizador 3D (na verdade foi o primeiro

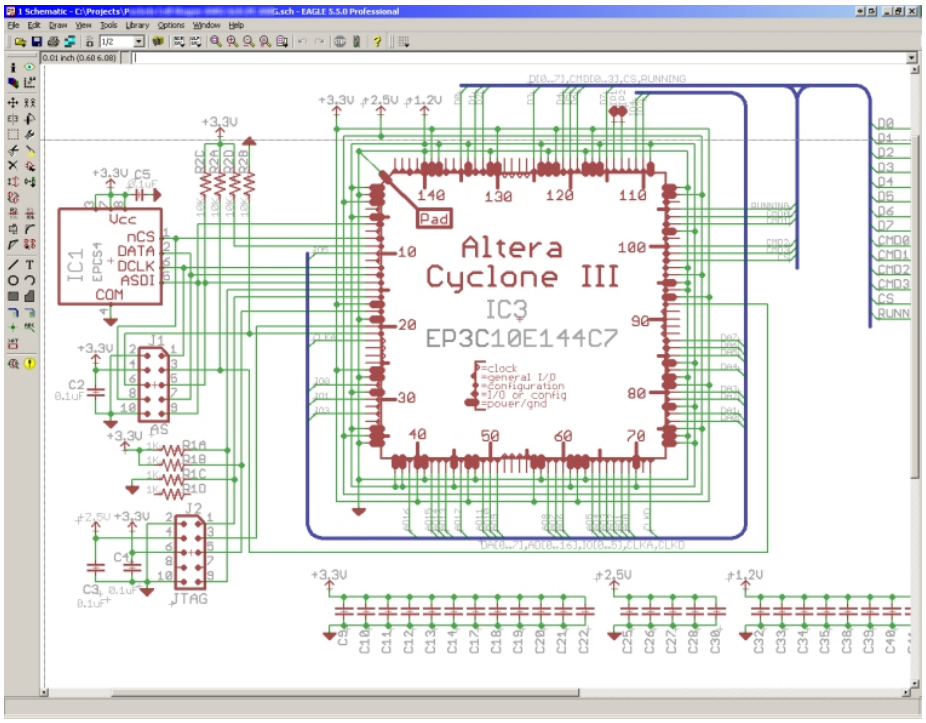


Figura 3.3: Editor de esquemáticos do Eagle

software a apresentar essa ferramenta, em 2007), preciso e muito útil para averiguar corriqueiros conflitos mecânicos que possam ocorrer no roteamento da placa. Software utilizado por diversas empresas que produzem placas com nível de complexidade elevado, interessante para quem deseja ter contato com um software de nível profissional.

3.3.4 Psim

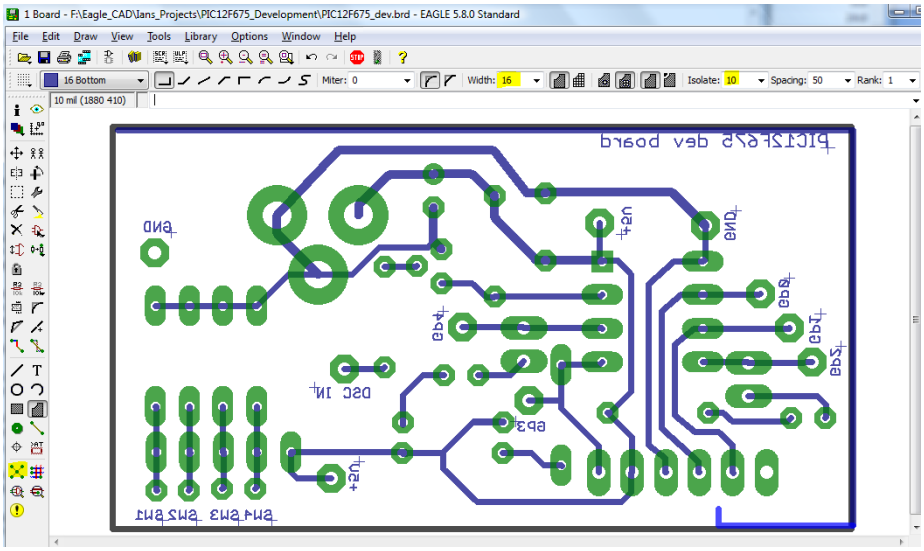


Figura 3.4: Editor de layout do Eagle

Capítulo 4

Mecânica

Aqui seria a parte mecânica do doc

Capítulo 5

Gerencial

Aqui seria a parte de Gerenciamento