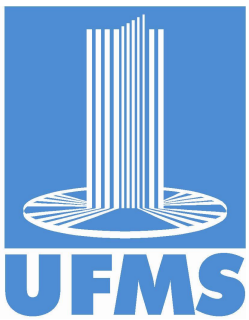
Plano de Trabalho

Controle de velocidade utilizando P.I.D.

Kelvim Rodrigues de Oliveira

Orientação: Prof. Doc. Gedson Faria

Bacharelado em Sistemas de Informação



Universidade Federal de Mato Grosso do Sul
Campus de Coxim

13 de Maio de 2017

Controle de velocidade utilizando P.I.D.

Plano de Trabalho

Coxim, 13 de Maio de 2017.

Capítulo 1

Introdução

Bla bla bla

1.1 Justificativa

Bla bla bla

1.2 Objetivos

1.3 Objetivo Geral

O objetivo deste trabalho é estudar e implementar controle de velocidade entre os motores esquerdo e direito do robô jogador.

1.4 Objetivos Específicos

- Criar uma interface simples e intuitiva para que qualquer pessoa sem alto nível de conhecimento no pacote *Vaucanson-G* possa utilizá-lo.
- Desenvolver o sistema de transição livre.
- Desenvolver o sistema de transição dupla.

1.5 Organização da Monografia

Capítulo 2

Revisão da Literatura

Capítulo 3

Futebol de robôs

Este capitulo destina-se alguns conceitos, informações e história do futebol de robôs, suas principais modalidades, dinâmica de jogo e movimentos.

3.1 Uma breve história sobre futebol de robôs

Com duas organizações de referencia mundial para o estudo e organização quanto ao futebol de robôs, são elas a FIRA, sigla em inglês para Federation of International Robot-soccer Association e a RoboCup associações que tem como objetivo promover pesquisas nas áreas de robótica e inteligência artificial, colaborando na divulgação científica através atividades afim de estimular o interesse dos participantes a resolver problemas.

A FIRA, a qual foi fundada no ano de 1995, pelo professor Kim Jong-Hwan, realizou seu primeiro campeonato mundial na Coréia, em 1997, dois anos após sua fundação, com o objetivo de levar aos leigos e as gerações jovens o espírito da ciência e tecnologia aplicada à robótica. Desde então a FIRA vem realizando diversos eventos em vários países do mundo, incluindo nossa pátria, o que fez com que o futebol de robôs obtivesse reconheci-mento mundial. Então o Brasil, que teve a oportunidade de sediar a Copa do Mundo em agosto de 1999, realizada entre os dias 4 a 8, na cidade Campinas no estado de São Paulo.

A RoboCup Criada no Japão por meio da iniciativa de um grupo de pesquisadores que tinham como objetivo em comum, o jogo de futebol, afim de promover o crescimento da ciência e tecnologia. De forma independente, os professores Minoru Asada,da Universidade de Osaka, e a Professora Manuela Veloso junto a seu aluno Peter Stone da Universidade Carnegie Mellon, EUA, estavam também trabalhando em robôs jogadores de futebol. O primeiro campeonato da Robocup junto a primeira conferência, foram realizados em 1997. Onde compareceram mais de 40 equipes e mais de 5.000 espectadores. Infelizmente no Brasil

até o momento não foram realizados eventos da Robocup com âmbito mundial.

3.2 Modalidade

Assim como o futebol jogado por humanos, o futebol de robôs também tem suas regras de jogo bem definidas. A FIRA Cup é um evento organizado com competições várias categorias, destacam-se as categorias Micro-Robot Soccer Tournament (MiroSot) e a Humanoid Robot Soccer Tournament (HuroSot), ambas são disputadas sob o olhar de um árbitro humano, já a categoria Simulated Robot Soccer Tournament (SimuroSot), que como o próprio nome sugere, trata-se de uma modalidade de simulação, quando não se tem a disposição de os recursos de hardware referentes aos robôs.

Este trabalho tem como foco o estudo das velocidades dos motores da categoria MiroSot. O acrônimo MiroSot, vem da denominação Micro Robot Soccer Tournament, cuja categoria é voltada para partidas com pequenos robôs que se movimentam através de dois motor, um motor para cada roda, realizando movimentos rápidos, enviados a partir de um computador disposto com uma inteligencia artificial com suas estratégias. Sendo este o ambiente principal do trabalho. A partida deve ser jogada por duas equipes, com três robôs cada, um robô pode atuar como goleiro de acordo com as estratégias, já fora de campo o time contem três membros humanos, o "gerente", o "técnico"e um "instrutor"os quais só tem acesso ao palco. Cada equipe possui seu computador o qual é principalmente dedicado ao processamento de visão, identificação e de localização. O tamanho dos robôs desta categoria é limitado a 7,5 cm x 7,5 cm x 7,5 cm, altura da antena não deve ser considerada na medição do tamanho de um robô, é utilizado uma bola de golfe de cor laranja afim de facilitar o processo de visão computacional. Esta categoria conta com duas ligas, a "Large league"(Liga grande) considerado grande pois o seu campo possui as dimensões de 400cm x 280cm e a "Middle league"(Liga média) que tem seu campo menor com as dimensões de 220cm x 180cm.

======== INSERIR IMAGEM DAS DIMENSÕES DO CAMPO =========

Capítulo 4

O robô

O time de futebol de robôs do campus de Coxim é participante da modalidade MiroSot confome a seção 3.2 esta limitado as dimensões de 7,5 cm x 7,5 cm x 7,5 cm, dotado de um Arduino nano o qual realiza o controle de todo o robô, um bluetooth do modelo **HC-06** que possui apenas o modo *slave*(apenas recebe conexão, não busca a mesma) cujo função principal é estabelecer a conexão entre o robô e o computador da equipe, um driver para motor de corrente continua do modelo **TB6612FNG** capais de controlar dois motores utilizados para realizar os movimentos, dois sensores ópticos do modelo **TCRT1000** os quais serão utilizados como encoder afim de determinar as velocidades das rodas.

4.1 Arduino Nano

4.2 Sensor Optico TCRT1000

O sensor optico utilizado do modelo TCRT1000 é um sensor refletivo o qual inclui um emissor infravermelho e um phototransistor dentro de um invólucro de chumbo o qual bloquei a luz visível, um sensor de pequenas dimensões medindo apenas 7mm x 4mm x 2.5mm e de grande precisão.

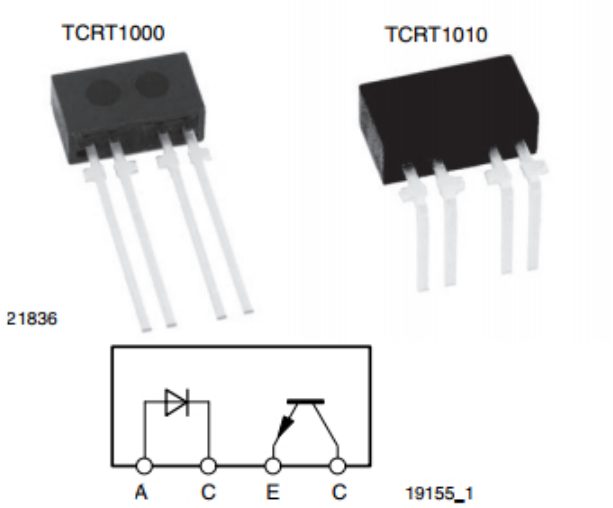


Figura 4.1: Sendor optico TCRT1000

4.3 Roda

O robô Modu possui apenas um par de rodas de XXmm e uma faixa de marcadores, utilizados juntamente com os sensores opticos 4.2 transformando o conjunto em um encoder 4.4 descrito na próxima sessão, tendo 17 marcadores brancos e 17 marcadores preto totalizando 34 marcadores.

4.4 Encoder

4.5 Bluetooth HC-06

O modulo foi utilizado para comunicação sem fio entre o robô e o computador, as informações são trocadas utilizando do protocolo *Serial*, o alcance do módulo segue o padrão da comunicação bluethooth, aproximadamente 10 metros, o suficiente, visto que o campo não passa pouco dos dois metros. Uma característica deste modelo (HC-06) possui apenas o modo *slave* neste modo é apenas permitido receber conexão de outros dispositivos, ou seja, não permite que este modelo busque e solicite conexão com outros dispositivos, assim como ocorre com fones de ouvido e caixas de som.

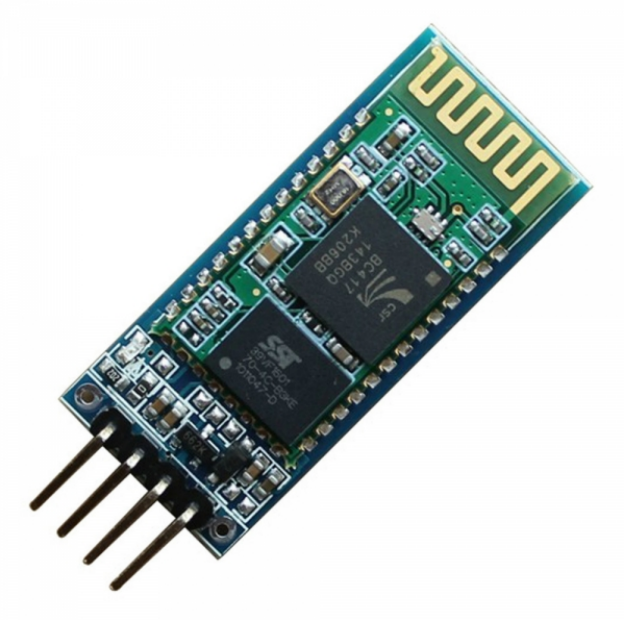


Figura 4.2: Bluetooth HC-06

Capítulo 5

Ambiente de desenvolvimento

Este capítulo aborda as ferramentas utilizadas para o desenvolvimento da programação dos robôs jogadores. O sistema operacional escolhido foi o Ubuntu Gnome 17.04 64-bit, por ser uma distribuição linux o Ubuntu já possui os drivers USB para comunicação a placas de teste, a utilizada no projeto foi a placa Arduino Nano 328, com o driver USB, como o FTDI. Sera descrito também neste capítulo a instalação e configuração dos softwares: IDE Arduino, IDE Clion 2017.1 e PlatformIO.

5.1 Arduino IDE

======== https://www.arduino.cc/en/main/software === O Arduino 1.8.3(IDE) é um software open-source para desenvolvimento de programas para Arduino. A utilização deste software torna fácil escrever códigos e subir(upload) para a placa. O Arduino IDE tem suporte para Windows, Mas OS e para Linux. Ambiente foi desenvolvido em Java baseado no Processing e outros softwares open-source, com a utilização da IDE qualquer placa Arduino pode ser utilizada com poucas configurações.

5.1.1 Intalação

Na pagina oficial de download https://www.arduino.cc/en/Main/Software foi realizado o download conforme o sistema operacional, neste caso, Linux x64, ao selecionar o download é possível fazer uma doação para os desenvolvedores da IDE, visto que é um projeto open-source e software livre, não tendo fins lucrativos, o mesmo depende dessa colaboração da comunidade para permanecer ativo e continuar expandido. Então realizado o download da versão atual, neste momento a versão 1.6.10 para linux.

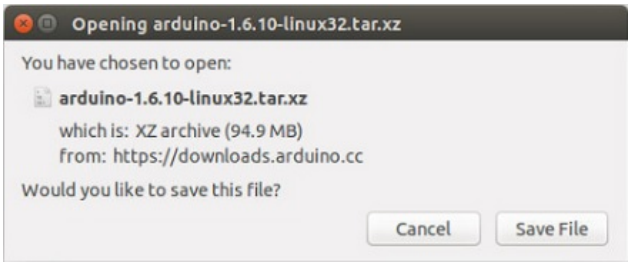


Figura 5.1: Titulo.

Ao finalizar o download do arquivo compactado, sera necessário descompactar em um diretório desejado, este ainda não sera o diretório de instalação.

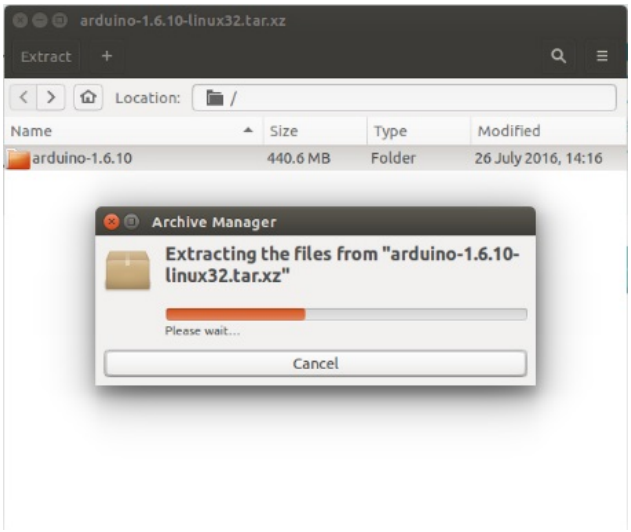


Figura 5.2: Titulo.

Abrindo o terminal navegue pelos diretórios utilizando o comando 'CD' até o diretório arduino-1.6.x o qual foi criado ao descompactar o arquivo. Execute então o arquivo 'intall.sh' utilizando o comando:

```
1 ./install.sh
```

O processo de instalação é rápido, um novo ícone foi criado na área de trabalho.

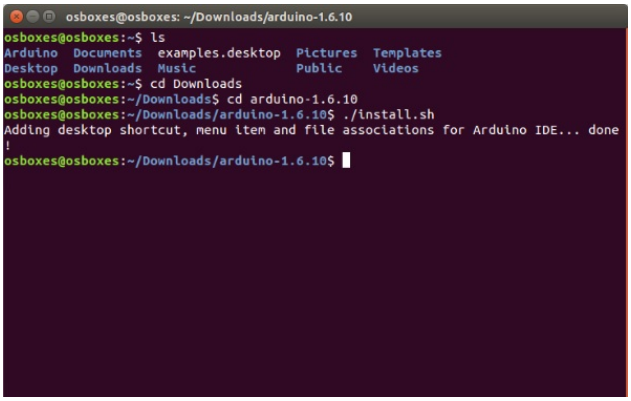


Figura 5.3: Titulo.

5.2 Clion 2017.1

======= https://www.jetbrains.com/clion/ ======= CLion é uma IDE multi-plataforma, criada pela empresa JetBrains para desenvolvimento de softwares nas linguagens C e C++, uma ferramenta poderosa. Suporte as linguagens nativas C e C++, incluindo C++11, C++14, libc++ e mais. A ferramenta desenvolvimento conta com ótimos recursos tais como: navegação, geração de código, gerenciamento de bibliotecas entre outros.

5.2.1 Instalação

Na página oficial de download do Clion "https://www.jetbrains.com/clion/download/#linux" foi realizado o download do arquivo "CLion-*.tar.gz" da versão compatível com o sistema operacional. Descompacte o arquivo "CLion-*.tar.gz", neste projeto foi descompactado no diretório /opt, sendo assim foi utilizado o seguinte comando para descompactar direto no diretório desejado.

```
1 sudo tar xf CLion-*.tar.gz -C /opt/
```

Ao finalizar a descompactação, novamente navegue até o diretório /bin, diretório este que se encontra junto aos arquivos descompactados, utilizando o comando:

```
1 cd /opt/CLion-*/bin
```

Execute o arquivo clion.sh que se encontra dentro do subdiretório /bin, com o comando:

```
1 ./clion.sh
```

5.3 PlatformIO

Diferente microcontroladores normalmente exigem diferentes ferramentas de desenvolvimento, para o Arduino temos a Arduino IDE. Outros usuários preferem ferramentas com auxílios. Tais ambientes de desenvolvimentos como o Eclipse, o qual trás recursos que ajudam a gerenciar seus projetos, bibliotecas e funções de autocompletar. As vezes é um pouco difícil manter-se na linha com diferentes microcontroladores e ferramentas. Então o ecossistema (como é chamado pelos seus desenvolvedores) open-source do PlatformIO junta tudo em uma única ferramenta. Sendo uma ferramenta multiplataforma podendo ser instalada nos sistemas operacionais Linux, Windows e MAC, dá suporte ao compilação para mais de 200 placas de teste, com mais de 15 plataformas de desenvolvimento e 10 frameworks, cobrindo então as placas mais populares do mercado, fazendo o trabalho duro de organização de centenas de projeto e bibliotecas que podem ser incluídas no projeto.

5.3.1 Instalação

A instalação ou atualização no MAC e Linux é feita pelo terminal de um modo muito fácil, apenas utilizando o comando(sudo pode ser requerido):

```
1 python -c "$(curl -fsSL https://raw.githubusercontent.com/platformio/platformio/
    master/scripts/get-platformio.py)"
```

5.4 Iniciando um novo projeto

Para criar um novo projeto primeiramente devemos abrir a IDE Clion, deve-se criar um novo projeto na linguagem C++ clicando no File, New Poject então em Create. Ao finalizar a criação do novo projeto, abra as opções de configurações utilizando o atalho de teclado “Ctrl + Alt + S”clique na aba “Plugins” no campo de texto procure por Arduino, conforme a figura abaixo. Caso não seja localizado no repositório local clique em “Search in repositories” para procurar nos repositórios da JetBrains.

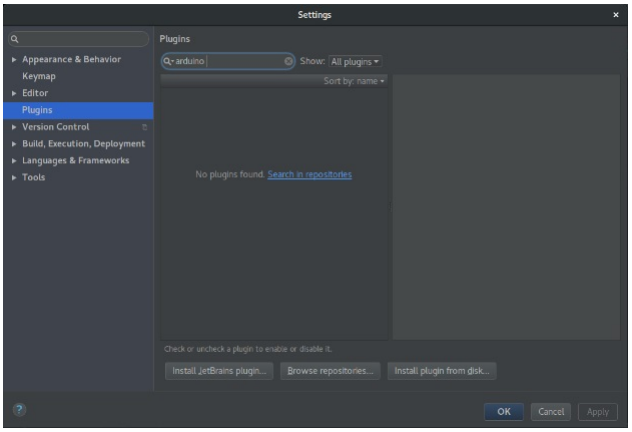


Figura 5.4: Titulo.

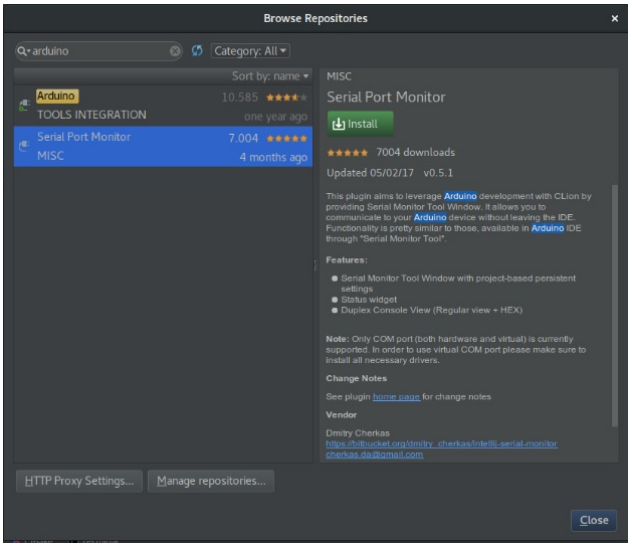


Figura 5.5: Titulo.

Após o final da instalação dos dois plugins reinicie o CLion para os plugins venham funcionar. Com a IDE aberta, no terminal, navegue com o comando "CD"até o diretório do projeto o qual acabou de ser criado. Neste projeto foi criado a pasta TCCTerceira no diretório de projetos do CLion o qual se encontra na home do linux.

```
1 cd /home/kelvimro/CLionProjects/TCCTerceira
```

Ainda no terminal é utilizado o comando "platformio" com a opção "boards", este comando lista as placas de teste suportadas pelo PlatformIO.

```
1 platformio boards
```

Ao localizar a placa desejada na lista, Arduino Nano processador Atmel 328, a qual foi escolhida para ser utilizada no desenvolvimento dos robôs de futebol do time UFMS-CPCX.

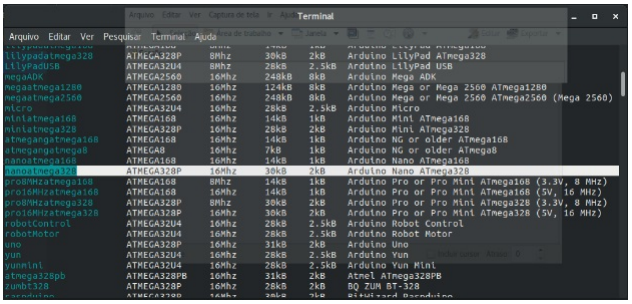


Figura 5.6: Titulo.

Localizado e copiado o nome da placa “nanoatmega328”, utilize o comando “platformio” com as opções de “init” para iniciar/instalar os recursos necessários no projeto, a opção “ide” deve-se ser utilizada o parâmetro “clion”indicando que utilizamos o framework para trabalhar junto a IDE do Clion, sendo assim o comando completo utilizado foi:

```
1 platformio init --ide clion --board nanoatmega328
```

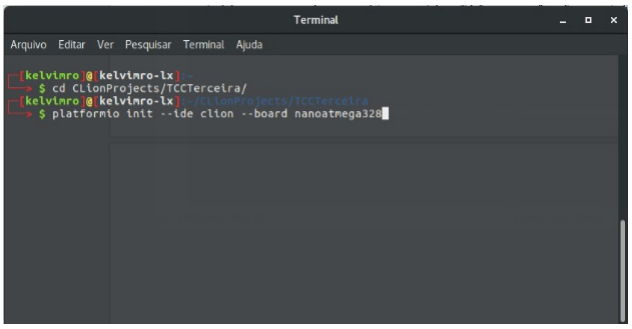


Figura 5.7: Titulo.

Finalizando a inicialização do PlatformIO será nescessário recarregar o arquivo CMake. Para isso clique com o botão direito do mouse em cima do nome do projeto(TCCTerceira) e em seguida em “Reload CMake Project”.

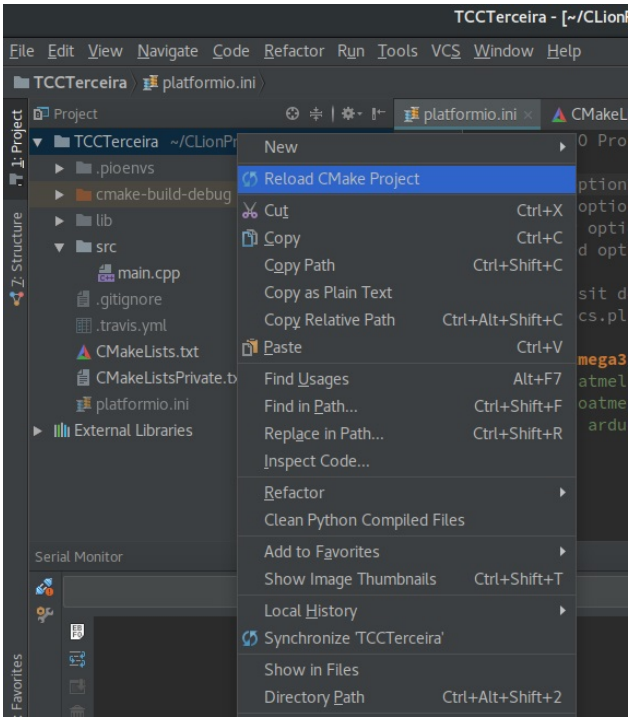


Figura 5.8: Titulo.

O PlatiformIO requer que as classes, assim como a Main.cpp, deve estar dentro da subpasta "src"pois a ferramenta é configurada nativamente para buscar e compilar os códigos que ali estão. Na classe Main.cpp deve-se fazer o importe da biblioteca "Arduino.h"por meio do "#include"a qual é a sintaxe para importar em C.

```
1 #include <Arduino.h>
```

O código a seguir foi utilizado para teste de compilação e funcionamento do ambiente como um todo no fim da configuração. O programa tem a função básica, de piscar o led embutido na placa do arduino (LED_BUILTIN) ou seja o pino da porta 13, ficando 1 segundo acesso e 1 segundo apagado.

```
1  #include <Arduino.h>
2  void setup() {
3  // inicializa o pino digital LED_BUILTIN como saida.
4  pinMode(LED_BUILTIN, OUTPUT);
5  }
6
7  // Função loop do arduino
8  void loop() {
9  digitalWrite(LED_BUILTIN, HIGH);   // liga o LED
10 delay(1000);                       // espera um segundo
```

```
11 digitalWrite(LED_BUILTIN, LOW);    // desliga o LED
12 delay(1000);                       // espera um segundo
13 }
```

Compilando o código utilizando Ctrl + F9 ou clicando no ícone no canto superior direito

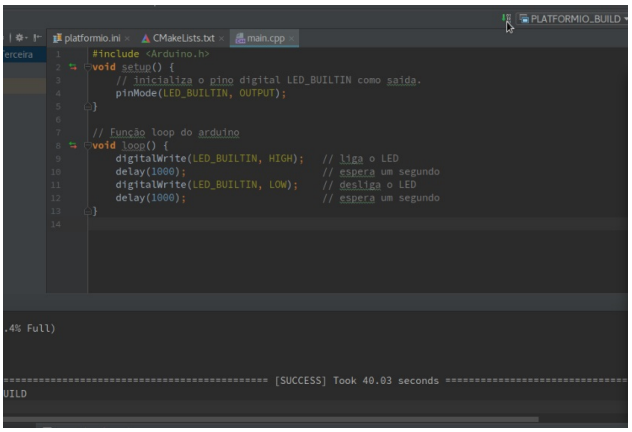


Figura 5.9: Titulo.

Ao receber a mensagem de "SUCCESS"na aba "Messages", indica que o "build"ou seja, a compilação do programa foi concluída com exito. No fim a compilação (build) é hora de subir(upload) do código para a placa arduino mudando para opção PLATFORMIO_UPLOAD (na caixa de seleção no canto superior direito) e clicando no ícone "PLAY"verde.

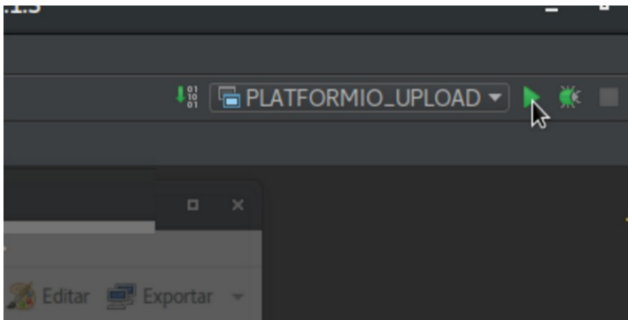


Figura 5.10: Titulo.

Aguarde a próxima mensagem de sucesso, a qual indicando que o programa já foi completamente enviado a placa, então o código já rodando. ATENÇÃO: Algumas distribuições podem solicitar a permissão de administrador para ter acesso ao dispositivo USB antes de fazer o upload do código. O privilégio pode ser concedido com a utilização do comando:

```
1 sudo chmod 777 /dev/ttyUSB0
```

Ps.: O numero “0” em ttyUSB0 deve ser trocado conforme a porta ao qual o dispositivo foi conectado.

Ps2.: Em algumas distribuições as permissões devem ser dadas sempre o dispositivo for conectado/reconectando. Finalizando assim as configurações do ambiente de desenvolvimento, que foi utilizado para gerenciar e editar os códigos neste trabalho de conclusão.

Capítulo 6

Controle Bluetooth

Um simples projeto foi desenvolvido também em Arduino. O projeto tem a finalidade de ser um controle BT(Bluetooth) com um Joystick, utilizado para controlar as direções do robô. O joystick, um dos componentes do controle, nada mais é que dois potenciômetros, um potenciômetro para os valores do eixo X e outro para os valores do eixo Y. O controle possui um componente BT para estabelecer uma troca de dados junto ao robô, o qual também possui seu componente BT, o qual será explicado no capítulo ¡TeX¿. O código atualizado pode ser encontrado no link: https://github.com/kelvimro/TCCTerceira/blob/master/lib/ControleBT/controleBT
O controle tem o intuito de simular os comandos que são gerados pela inteligência artificial no software que é responsável pelas táticas e movimento dos jogadores durante uma partida. Os comandos podem variam entre -100 até 100, tendo um valor para cada motor, ou seja cada computação realizada pela inteligência artificial é enviado dois números inteiros, onde o primeiro número é destinado ao motor esquerdo e o segundo ao direito. Utilizando-se da função map() a qual proporciona os valores do potenciômetro - dos quais variam entre 0 a 1023 - aos valores válidos para o robô.

```
1 if (cmdX >= 0 && cmdX <= 501) cmdX = map(cmdX, 0, 501, -100, 0);
2 else cmdX = map(cmdX, 502, 1023, 0, 100);
3 if (cmdY >= 0 && cmdY <= 509) cmdY = map(cmdY, 0, 509, -100, 0);
4 else cmdY = map(cmdY, 510, 1023, 0, 100);
```

O potenciômetros Y, o qual vai representar os valores de potência, em posição neutra (sem ação do controlador) encontra-se com valor analógico de 509, então mapeado para 0, ou seja, não há potência requerida. Ao movimentar o joystick para frente, aumenta-se o valor analogio do potenciômetro até o máximo de 1023 que após ter seu valor mapeado representa o valor 100. Movimentando o joystick para trás, os valores analogicos lido diminuem até 0, denotando o valor -100.

```
if (cmdY >= 0 && cmdY <= 509) cmdY = map(cmdY, 0, 509, -100, 0);
else cmdY = map(cmdY, 510, 1023, 0, 100);
```

Valores positivos no eixo Y indicam que o movimento solicitado é para frente, enquanto os valores negativos representam movimento para trás. O potenciômetros X, o qual vai representar os valores da diferença entre os motores. Em sua posição neutra tem o valor analogico lido de 502, representando 0% de diferença entre os motores. Afim de facilitar a compreensão chamaremos de A o motor do lado esquerdo e de B o motor do lado direito. Então ao movimentar o joystick para a esquerda o valor analógico lido diminui até 0, resultando no valor de -100, movimentando o joystick para a direita aumentando o valor analógico lido até 1023 que representa o valor 100.

```
if (cmdX >= 0 && cmdX <= 501) cmdX = map(cmdX, 0, 501, -100, 0);
else cmdX = map(cmdX, 502, 1023, 0, 100);
```

No eixo X os valores negativos indicam que o motor A tem de se movimentar X% mais lento em relação ao motor B, fazendo o movimento de curva para o lado de A. Já com os valores positivos de X indica que o motor B deve ser X% mais lento q o A fazendo a curva para o lado B.

Capítulo 7

Controlador PID

== https://www.citisystems.com.br/controle-pid/ === Neste capítulo, explanarei sobre o controle PID -que é abreviatura para Proporcional, Integral e Derivativo- e como utiliza-lo em aplicações de controle que requerem um controle de precisão. Começaremos pelo exemplo onde um carro tenta manter uma distancia X do carro da frente.

7.1 P - Proporcional

Ao tentar manter uma distancia X o carro de trás deve acelerar proporcionalmente quando o carro da frente acelerar e começar a distanciar, tentando alcançar o mesmo. No entanto, caso o carro de trás venha acelerar mais que o carro da frente a distancia será menor que a desejada, X, então a nova correção é frear para aumentar a distancia buscando a distancia desejada de X. Novamente caso a correção, que desta vez é a frenagem(desaceleração), for muito, a sua distancia se será superior a distancia desejada de X, então novamente terá de acelerar. Isso ocorrera indefinitivamente caso a aceleração não proporcional não esteja correta, ou seja, acelerando de mais e freando de menos e não atingindo o ponto alvo(*set point*). Então esta aceleração é chapada de ganho ou proporcional(P) em um controle PID.

7.2 I - Integral

A integral, neste exemplo dos carros, seria como recuperar a distancia X a cada instante, caso o veiculo da frente venha a acelerar, o veiculo de trás acelera gradualmente afim de atingir a distancia X com mais suavidade, este procedimento pode ser comparado a integral de um controle PID.

7.3 D - Derivativo

A derivada é utilizada para eliminar erros acumulados na integral, neste exemplo dos carros, seria quando o carro de trás percebe que a distancia desce ou decresce muito rapidamente em relação a desejada X, deixando as correções da integral ainda mais suaves, diminuindo a oscilação das correções em volta do Setpoint.

7.4 Utilização do PID

O controle do tipo PID foi empregado para controlar a velocidade da roda mais rápida em relação a mais lenta, como pode ser visto até o então, o controle PID se utiliza de dados obtidos durante a sua execução, no caso deste trabalho o tempo de um ciclo foi de 200 milissegundos(ms), ou seja, a cada 200 ms o controle obtém a velocidade de rotação de ambas as rodas por meio do encoder4.4 utilizando a velocidade do motor mais lento como *set point* do motor mais rápido, calculando a proporcional, a integral e derivando, utilizando o resultado como o novo valor do PWM do motor mais rápido o deixando com a velocidade de rotação mais próxima do motor de referencia.

7.4.1 Problemas de ciclo

Durante o emprego do controle PID para equalizar as velocidades dos motores foi percebido que o tempo de ciclo para o controle se tornar eficiente é superior o intervalo de comandos recebidos pelo computador de controle o qual envia novos comandos a cada 200ms. O controle do tipo PID não consegue equalizar as velocidades na primeira correção, então ao se aproximar da segunda ou terceira correção o robô recebe a próxima instrução mudando todas as referencias de velocidades.

7.4.2 Problemas devido interrupções

Como pode ser visto o encoder4.4 é o conjunto da roda4.3 provida de marcadores e dos sensores ópticos4.2, para realizar a contagem de marcadores os sensores ópticos estão conectados nos pinos digitais 2 e 3, se utilizando da função de interrupção para contagem dos marcadores e determinas as velocidades, veremos no próximo capitulo, mais precisamente na seção de rotinas de interrupção8.1, que ao gerar uma interrupção todo o sistema "para"temporariamente, trazendo discrepância nas contagens devido a sobreposição de interrupções. Concluindo então que a utilização de controle do tipo PID do modo implementado

não é recomendado.

Capítulo 8

Interrupções

O processador do Arduino Nano, o qual foi utilizado neste projeto é do modelo Atmega328 e nesta seção discorreremos sobre o funcionamento da interrupção neste processador, utilização no projeto, exemplos e !!efeitos colaterais!!. Em sua maioria os processadores possuem interrupções dos quais permitem executar algo após determinado evento externo, sendo chamado de interrupções externas. Nas referencias online presentes no site oficial do Arduino, para mais informações a cerca de interrupções é indicado as notas de Nick Gammon!!(https://gammon.com.au/interrupts)!!, de um modo didático e simples ele faz um paralelo ao cotidiano, ao fazer o jantar e ter de cozinhar as batatas, é colocado um despertador com o tempo de 20 minutos, indo faze outra atividade durante o tempo de cozimento, ao despertar o alarme vc "interrompe"sua atividade e então termina o jantar. Apesar do exemplo utilizado ser quase uma mudança de estado, não é recomendado invocar funções grandes e complexas pois além de sair da linha principal de execução, as funções não contem argumentos, utilizamos então variáveis do tipo voláteis. Logo recomenda-se as interrupções para trocas de *flag*, disparo de funções emergenciais, contagens precisas entre outros a interrupção é o "botão vermelho", veremos a seguir que *reset* é a interrupção de maior prioridade.

8.1 Rotinas de serviço de interrupção (ISR)

As rotinas de serviço de interrupção conhecidas pela siga ISR que vem do inglês *Interruption Service Rotines* é uma rotinha a qual se encontra fora da função **loop**, ou sejá, é uma função a qual será executada quando ocorrer uma interrupção, interrompendo a execução do código **loop** onde quer que ele esteja, retornando ao ponto de "pausa"ao fim da ISR. Algo de suma importância é citado por Simon Monk em seu livro, Programação com Arduino vol. 2

> "Quando uma ISR está sendo executada, as interrupções são automaticamente desligadas. Isso evita a confusão que poderia ser causada se as ISRs interrompessem umas as outras."

Ou seja, enquanto uma ISR estiver ativa outras não serão executadas, exceto uma interrupção de maior prioridade como o *reset*.

8.2 Modos de Interrupção

Os chamados modos de interrupção é como gatilho, irá disparar a ISR, ou seja, uma ISR é uma função a ser executada sempre quando o gatilho é acionado. O Atmega328 possui quatro modos de interrupção, são eles: LOW - Dispara uma interrupção continuamente enquanto o pino selecionado estiver em nível baixo (*low*). RISING - Dispara uma interrupção sempre que o pino passar de baixo (*low*) para alto (*high*). FALLING - Dispara uma interrupção sempre que o pino passar de alto (*high*) para baixo (*low*). CHANGE - Dispara uma interrupção sempre que o pino mudar de nível, em ambos os sentidos. Vale ressaltar que o modelo Arduino Due tem um quinto modo de interrupção, o *HIGH*. HIGH - Dispara uma interrupção continuamente enquanto o pino estiver em nível alto (*high*), assim como o modo *LOW*.

8.3 Interrupção no Atmega328

Assim como em outros processadores o Atmega328 possui sua lista de interrupções, logo a baixo podemos ve-la em ordem de prioridade, tendo em primeiro lugar -maior prioridade- a interrupção de *reset*.

Como podemos na tabela acima as interrupções externas nomeadas de INT0_vect e INT1_vect que correspondem aos pinos digitais 2 e 3 respectivamente, no Arduino Nano(os pinos de interrupção podem mudar de acordo com a placa) sendo as maiores prioridades seguidas do *reset*, aproveitamos para observar as interrupções de 7 a 17, as quais são interrupções de tempos e a interrupção de prioridade 25 a qual é a interrupção para uso da interface serial.

8.4 Utilizando interrupção no Arduino

Nesta seção foi como exemplos os códigos utilizados no robô no desenvolvimento do trabalho, conforme visto na seção anterior 8.3 os pinos de interrupção são os pinos digitais

1	Reset	
2	External Interrupt Request 0 (pin D2)	(INT0_vect)
3	External Interrupt Request 1 (pin D3)	(INT1_vect)
4	Pin Change Interrupt Request 0 (pins D8 to D13)	(PCINT0_vect)
5	Pin Change Interrupt Request 1 (pins A0 to A5)	(PCINT1_vect)
6	Pin Change Interrupt Request 2 (pins D0 to D7)	(PCINT2_vect)
7	Watchdog Time-out Interrupt	(WDT_vect)
8	Timer/Counter2 Compare Match A	(TIMER2_COMPA_vect)
9	Timer/Counter2 Compare Match B	(TIMER2_COMPB_vect)
10	Timer/Counter2 Overflow	(TIMER2_OVF_vect)
11	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)
12	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)
13	Timer/Counter1 Compare Match B	(TIMER1_COMPB_vect)
14	Timer/Counter1 Overflow	(TIMER1_OVF_vect)
15	Timer/Counter0 Compare Match A	(TIMER0_COMPA_vect)
16	Timer/Counter0 Compare Match B	(TIMER0_COMPB_vect)
17	Timer/Counter0 Overflow	(TIMER0_OVF_vect)
18	SPI Serial Transfer Complete	(SPI_STC_vect)
19	USART Rx Complete	(USART_RX_vect)
20	USART, Data Register Empty	(USART_UDRE_vect)
21	USART, Tx Complete	(USART_TX_vect)
22	ADC Conversion Complete	(ADC_vect)
23	EEPROM Ready	(EE_READY_vect)
24	Analog Comparator	(ANALOG_COMP_vect)
25	2-wire Serial Interface (I2C)	(TWI_vect)
26	Store Program Memory Ready	(SPM_READY_vect)

Tabela 8.1: Lista de interrupções para o Atmega328.

2 e 3. Declaramos então no código as variáveis das quais irá representar os pinos, utilizando o código.

```
1 const int encodPinA1 = 2;                    // encoder A pino 2
2 const int encodPinB1 = 3;                    // encoder B pino 3
```

Dentro da função *setup* !!intanciamos!! os pinos como *INPUT* pois estes pinos farão a entradas de informação ao pino de interrupção.

```
pinMode(encodPinA1, INPUT);
pinMode(encodPinB1, INPUT);
```

Segundo as refecias disposta na documentação do site oficial do Arduino temos três sintaxes diferentes para acoplar/iniciar, porem dentre elas apenas uma é recomendada pelos desenvolvedores, sendo assim foi utilizado a sintaxe recomendada.

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

A função *digitalPinToInterrupt(pin)* é utilizada para traduzir o pino digital para o numero especifico da interrupção. ISR8.1 é o nome da função a ser chamada quando a interrupção ocorrer, lembrando que esta função não deve conter argumentos. Mode define o modo de quando a interrupção deve disparar, conforme descrito na seção 8.2. Seguindo então a sintaxe recomendada para iniciar nossa interrupção com o pino e função desejada.

```
attachInterrupt(digitalPinToInterrupt(encodPinA1), rencoderA, CHANGE);
attachInterrupt(digitalPinToInterrupt(encodPinB1), rencoderB, CHANGE);
```

Como pode ser visto no código acima, a função com nome de rencoderA (ISR) será executada sempre que seu valor do pino encodPinA1 mudar, pois seu modo foi defino como *CHANGE*, o mesmo acontece com o pino encodPinB1 chamando sua respectiva ISR nomeada como rencoderB.

```
    void rencoderA() {
    countA++;
    }
    void rencoderB() {
    countB++;
    }
```

Do mesmo modo que podemos acoplar uma interrupção a um pino especifico, também podemos desacopla-lo utilizando o código a seguir sempre que achar necessário.

```
detachInterrupt(digitalPinToInterrupt(encodPinA1));
detachInterrupt(digitalPinToInterrupt(encodPinB1));
```

O código exemplificado acima tem a função de desativar a interrupção do pino “encodPinA1”e do pino “encodPinB1”exclusivamente.

Existe um modo alternativo de ativar e ativar/desativar as interrupções, mas estas funções ativam/desativam todas a interrupções -exceto *rest*-

```
interrupts();       // Ativa interrupções
noInterrupts();     // Desativa interrupções
```

Lembrando que ao desativando as interrupções utilizando-se da função acima citada, as interrupções citadas na tabela8.1como a *Serial* também são desativada. Sendo muito útil quando nos trechos de códigos sensíveis ao tempo, no código a seguir desativamos as interrupções para evitar que os valores da variável contadora sejam alterados durante a execução, evitando erros de leitura quanto as velocidades das rodas.

```
    noInterrupts();             // Desarma interrupts
    calibA.push(countA);        // Adiciona a ultima contagem a fila de amostras
    countA = 0;                 // Zera contadores de marcos do encoder
    calibMillis = millis();     // Zera o contador de tempo
    interrupts();               // Arma interrupsts
```

Capítulo 9

Metodologia

(A) Desenvolvimento e entrega do plano de trabalho;

(B) Escrita da introdução, levantamentos dos dados teóricos e bibliográficos.

(C) Revisão da literatura.

(D) Implementar a aplicação

(E) Escritas do trabalho de conclusão do curso.

Capítulo 10

Considerações Finais

Prof. Doc. Gedson Faria
Orientador

Kelvim Rodrigues de Oliveira
Acadêmico

Referências Bibliográficas

[1] DA SILVA, F. C. Planta didÁtica – controle pid digital para motor dc. Dezembro 2008.

[2] DA SILVA ALVES, J. D. An interactive system for automata manipulations.