



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Vinicius Delatore Martins de Andrade

DESENVOLVIMENTO DE UM DISPOSITIVO PARA
MEDIÇÃO DE SINAIS DE TRANSDUTORES DE
TORQUE

Florianópolis

2025

Vinicius Delatore Martins de Andrade

Desenvolvimento de um dispositivo para medição de
sinais de transdutores de torque

Trabalho de Conclusão de Curso - Graduação em
Engenharia Elétrica pela Universidade Federal de
Santa Catarina.

Orientador: Cristian Franzoi Mazzola, MSc. Eng.

Florianópolis
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

de Andrade, Vinicius Delatore Martins
Desenvolvimento de um dispositivo para medição de sinais
de transdutores de torque. / Vinicius Delatore Martins de
Andrade ; orientador, Cristian Franzoi Mazzola, 2025.
113 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia Elétrica, Florianópolis, 2025.

Inclui referências.

1. Engenharia Elétrica. 2. Eletrônica. 3. Transutor de
torque. I. Mazzola, Cristian Franzoi. II. Universidade
Federal de Santa Catarina. Graduação em Engenharia
Elétrica. III. Título.

Vinicius Delatore Martins de Andrade

**DESENVOLVIMENTO DE UM DISPOSITIVO PARA MEDIÇÃO DE SINAIS DE
TRANSDUTORES DE TORQUE**

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Engenharia Elétrica” e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 11 de fevereiro de 2025.



Prof. Miguel Moreto, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:



Eng. Cristian Franzoi Mazzola, MSc. - Orientador
Universidade Federal de Santa Catarina (UFSC)



Prof. Jean Viane Leite, Dr. - Avaliador
Universidade Federal de Santa Catarina (UFSC)



Prof. Roberto Coelho, Dr. - Avaliador
Universidade Federal de Santa Catarina (UFSC)



Prof. Cristhian Becker, Dr. - Avaliador
Universidade de Santiago - Chile (USACH)

Este trabalho é dedicado à minha mãe, cuja insistência, amor e apoio incansáveis foram fundamentais. Te amo imensamente!

AGRADECIMENTOS

Primeiramente, agradeço imensamente ao meu orientador, Cristian Franzoi Mazzola, que foi mais que um guia acadêmico, mas um verdadeiro amigo. Sua paciência, dedicação e persistência foram fundamentais para a realização deste trabalho. Sou eternamente grato por tudo o que fez por mim.

Também deixo um agradecimento imenso à UFSC, que foi o palco de uma das maiores transformações da minha vida. Mudar de cidade e começar essa nova jornada foi um desafio, mas também uma oportunidade de crescimento. Apesar das dificuldades e momentos de dúvidas, aprendi mais do que imaginei, e tive o privilégio de conhecer pessoas que se tornaram amigos para a vida. Sou eternamente grato por tudo o que essa instituição representa e pela chance de fazer parte dessa história.

*"O verdadeiro aprendizado é aquele que nos torna
mais humildes diante do vasto desconhecido."*

RESUMO

Delatore, Vinicius. **Desenvolvimento de um dispositivo para medição de sinais de transdutores de torque**. 113 p. Trabalho de conclusão de curso (TCC) – Universidade Federal de Santa Catarina (UFSC), 2025.

ESTE Trabalho de Conclusão de Curso apresenta o desenvolvimento e a implementação de um dispositivo de baixo custo para a medição de sinais provenientes de transdutores de torque instalados nas bancadas didáticas de conversão eletromecânica de energia do Laboratório de Máquinas e Acionamentos Elétricos (LABMAQ) da Universidade Federal de Santa Catarina (UFSC). Dessa forma, foi projetado um sistema alternativo, que também possui caráter didático, proporcionando ao aluno a experiência de desenvolver um dispositivo funcional para leitura e interpretação de sinais de transdutores. O sistema é composto por um módulo conversor analógico-digital (ADC), um microcontrolador e circuitos de condicionamento de sinal. Além do *hardware*, foi implementado um algoritmo de operação e funcionamento do dispositivo, bem como o desenvolvimento do *software* responsável pela aquisição e processamento dos dados. O dispositivo foi validado em uma bancada de ensaios e os resultados demonstraram excelente correlação das medidas com os valores de referência, garantindo a precisão e a confiabilidade da medição. O custo final dos insumos utilizados para a implementação do dispositivo foi de aproximadamente \$45,00 (dólar americano).

Palavras-chaves: amostrador de torque, transdutor de torque, medição de potência mecânica, conversão de sinais.

ABSTRACT

Delatore, Vinicius. **Development of a Device for Measuring Signals from Torque Transducers.** 113 p. Final Undergraduate Project (TCC) – Federal University of Santa Catarina (UFSC), 2025.

THIS Final Undergraduate Project presents the development and implementation of a low-cost device for measuring signals from torque transducers installed in the electromechanical energy conversion test benches at the Laboratory of Electrical Machines and Drives (LABMAQ) of the Federal University of Santa Catarina (UFSC). Commercial solutions available for this purpose are costly. Thus, an alternative system was designed, which also has an educational purpose, providing the student with the experience of developing a functional device for reading and interpreting transducer signals. The system consists of an analog-to-digital converter (ADC) module, a microcontroller, and signal conditioning circuits. In addition to the hardware, an operational algorithm was implemented, along with the development of the software responsible for data acquisition and processing. The device was validated on a test bench, and the results demonstrated an excellent correlation between the measurements and the reference values, ensuring measurement accuracy and reliability. The final cost of the device was approximately \$45.00 (american dollar).

Keywords: torque sampler, torque transducer, mechanical power measurement, signal conversion.

LISTA DE ILUSTRAÇÕES

Figura 2.1 – Bancada de ensaios de máquinas elétricas do LABMAQ.	26
Figura 2.2 – Transdutor de torque T22 - HBM.	27
Figura 2.3 – Pinagem do transdutor de torque T22 - HBM.	28
Figura 3.1 – Placa Arduino Nano - V3.	31
Figura 3.2 – Módulo de <i>display</i> LCD 16x2 para Arduino.	32
Figura 3.3 – Módulo conversor analógico digital - ADS1115.	33
Figura 3.4 – Módulo conversor de sinal de corrente em tensão HW-685.	34
Figura 3.5 – Fonte Ajustável de 12 V para 3,3 V ou 5 V - MB102.	35
Figura 3.6 – Conector Mike para a ligação entre o medidor e o transdutor de torque.	36
Figura 3.7 – Gabinete do dispositivo de medição.	36
Figura 3.8 – Diagrama esquemático de alimentação dos componentes do dispositivo.	37
Figura 3.9 – Diagrama esquemático de ligações do dispositivo.	38
Figura 3.10 – Fluxograma do algoritmo.	41
Figura 4.1 – Adaptação dos terminais para conexão com o transdutor de torque da bancada.	46
Figura 4.2 – Conectores Mike fêmea e peças de suporte.	46
Figura 4.3 – Estrutura do transdutor de torque da bancada após a adaptação.	47
Figura 4.4 – Testes de condicionamento e calibração do conversor de sinal HW-685.	49
Figura 4.5 – Parte interna do dispositivo de medição após a conclusão da montagem.	50
Figura 6.1 – <i>Setup</i> da bancada para a realização dos ensaios de validação.	59
Figura A.1 – Tela de instalação da plataforma - Em ordem numérica: (1) Janela de extensões; (2) Barra de pesquisa de extensão com o nome da plataforma; (3) Lista de resultados da pesquisa; (4) Opção de instalação na página de extensão <i>PlatformIO</i> IDE.	71
Figura A.2 – <i>Home</i> da <i>PlatformIO</i> com opção de “Novo Projeto”, em vermelho.	72
Figura A.3 – Árvore de arquivos criados pela <i>PlatformIO</i> no <i>Explorer</i> do <i>VSCode</i>	72
Figura A.4 – Arquivo <i>platfomio.ini</i> com as configurações iniciais do projeto do Amostrador de Torque.	73
Figura A.5 – Arquivo <i>main</i> genérico criado por <i>default</i> pela <i>PlatformIO</i>	73

Figura A.6–Serviços referentes ao projeto disponibilizados pela extensão <i>PlatformIO</i> com as opções de <i>upload</i> sinalizadas.	74
Figura A.7– <i>Logs</i> gerados pela <i>PlatformIO</i> ao executar o <i>Build</i> com sucesso.	74
Figura A.8– <i>Logs</i> gerados pela <i>PlatformIO</i> ao executar o <i>Upload</i> com sucesso.	75

LISTA DE ABREVIATURAS E SIGLAS

LABMAQ	Laboratório de Máquinas e Acionamentos Elétricos
DEEL	Departamento de Engenharia Elétrica e Eletrônica
UFSC	Universidade Federal de Santa Catarina
TCC	Trabalho de Conclusão de Curso
CPU	Unidade Central de Processamento
USB	Porta Serial Universal
UART	Receptor/Transmissor Assíncrono Universal
ASCII	Código Padrão Americano para Intercâmbio de Informação
I2C	Circuito Inter-Integrado
SCL	<i>Clock</i> Serial
SDA	Dado Serial
ADC	Conversor Analógico-Digital
LDO	Regulador de Baixa Queda
IDE	Ambiente Integrado de Desenvolvimento

LISTA DE SÍMBOLOS

V	Volt
mA	Miliampere
Ω	Ohm
mm	Milímetro
W	Watt
rpm	Rotações por minuto
%	Porcentagem
\pm	Mais ou menos
P_{joule}	Perdas por efeito Joule
R_a	Resistência de armadura
I_a	Corrente de armadura
T	Torque
P_m	Potência mecânica
$v_{rad/s}$	Velocidade de rotação
P_{rot}	Perdas rotacionais
P_{nu}	Perdas no núcleo
P_{in}	Potência de entrada
P_{out}	Potência de saída

SUMÁRIO

1	INTRODUÇÃO	23
1.1	OBJETIVOS METODOLÓGICOS	23
1.2	ORGANIZAÇÃO ESTRUTURAL DO TRABALHO	24
2	CONTEXTUALIZAÇÃO	25
2.1	BANCADAS DE ENSAIOS DE MÁQUINAS ELÉTRICAS	25
2.2	TRANSDUTORES DE TORQUE INSTALADOS NAS BANCADAS .	26
2.2.1	Especificações técnicas	27
2.2.2	Conexões elétricas	28
2.3	ESCOPO DO TRABALHO	28
3	PROJETO DO DISPOSITIVO AMOSTRADOR DE TORQUE	29
3.1	<i>HARDWARE</i> PRINCIPAL	29
3.1.1	Microcontrolador	29
3.1.1.1	Plataforma de prototipagem - Arduino	30
3.1.1.2	Arduino Nano	30
3.1.2	Display LCD	31
3.1.3	Módulo conversor analógico digital - ADS1115	32
3.1.4	Módulo conversor de sinal de corrente em tensão - HW 685	33
3.1.5	Fontes de Alimentação	34
3.2	<i>HARDWARE</i> PERIFÉRICO	35
3.2.1	Cabos e conectores	35
3.2.2	Gabinete	36
3.2.3	Componentes acessórios	36
3.3	DIAGRAMA ESQUEMÁTICO DO PROJETO	37
3.4	REQUISITOS DE <i>SOFTWARE</i>	38
3.4.1	Ambiente de desenvolvimento	39

3.4.2	Framework de Desenvolvimento	39
3.4.3	Estrutura do Código	39
3.5	FLUXOGRAMA DO ALGORITMO	40
3.5.1	Rotina de calibração	41
3.5.2	Rotina de medição	42
3.6	CONSIDERAÇÕES FINAIS DO CAPÍTULO 3	43
4	IMPLEMENTAÇÃO DO PROJETO - MONTAGEM DO HARDWARE E ADAPTAÇÕES DA BANCADA	45
4.1	ADAPTAÇÃO DOS TERMINAIS DE CONEXÃO DOS TRANSDUTORES	45
4.2	CONDICIONAMENTO E CALIBRAÇÃO DO CONVERSOR DE SINAL HW-685	47
4.2.1	Testes de condicionamento e calibração do conversor de sinal	48
4.3	MONTAGEM DO DISPOSITIVO DE MEDIÇÃO	50
4.4	CONSIDERAÇÕES FINAIS DO CAPÍTULO 4	50
5	IMPLEMENTAÇÃO DO PROJETO - DESENVOLVIMENTO DO SOFTWARE	51
5.1	AMBIENTE DE PROGRAMAÇÃO	51
5.2	DESENVOLVIMENTO DO CÓDIGO-FONTE	51
5.3	CONSIDERAÇÕES FINAIS DO CAPÍTULO 5	55
6	ENSAIOS DE VALIDAÇÃO DO DISPOSITIVO	57
6.1	ESTRATÉGIA ADOTADA PARA A VALIDAÇÃO DO TRABALHO	57
6.2	ENSAIOS DE VALIDAÇÃO	58
6.2.1	Setup da bancada	59
6.2.2	Resultados obtidos	60
6.3	CONSIDERAÇÕES FINAIS DO CAPÍTULO 6	62
7	CONCLUSÃO E SUGESTÕES PARA TRABALHOS FUTUROS	63

REFERÊNCIAS	65
APÊNDICES	65
APÊNDICE A – GUIA PRÁTICO PARA A UTILIZAÇÃO DAS FERRAMENTAS DE DESENVOLVIMENTO DO PROJETO	69
A.1 – GUIA PRÁTICO PARA A UTILIZAÇÃO DAS FERRAMENTAS DE DESENVOLVIMENTO DO PROJETO	70
A.2 – INSTALAÇÃO DO VS CODE E PERIFÉRICOS	70
A.3 – CRIAÇÃO DO PROJETO	71
A.4 – CONEXÃO COM O ARDUINO PARA INSERÇÃO DO <i>FIRMWARE</i> DESENVOLVIDO	73
APÊNDICE B – CÓDIGO-FONTE DO DISPOSITIVO	77
ANEXO A – TUTORIAL DE UTILIZAÇÃO (RANDOM TUTORIALS) - VS CODE E PLATFORMIO IDE	85

CAPÍTULO 1

INTRODUÇÃO

Neste Trabalho de Conclusão de Curso (TCC) é apresentado o desenvolvimento e a implementação de um dispositivo para a medição de sinais provenientes dos transdutores de torque das bancadas didáticas de conversão eletromecânica de energia do Laboratório de Máquinas e Acionamentos Elétricos (LABMAQ) da Universidade Federal de Santa Catarina (UFSC). As bancadas do LABMAQ, utilizadas para ensaios de máquinas elétricas, contam com transdutores de torque capazes de medir a potência mecânica transferida nos eixos das máquinas. No entanto, os sistemas de medição estão incompletos, não havendo a disponibilidade de dispositivos para a leitura dos sinais dos transdutores de torque. Durante a realização deste TCC, foi proposta uma alternativa de baixo custo para o desenvolvimento de um dispositivo capaz de realizar e amostrar tais medições de torque.

1.1 OBJETIVOS METODOLÓGICOS

O objetivo deste trabalho é desenvolver um dispositivo de baixo custo para medição de sinais dos transdutores de torque das bancadas didáticas do LABMAQ. Para a realização do trabalho, foram seguidos os seguintes objetivos metodológicos:

1. Analisar as especificações técnicas dos transdutores de torque disponíveis nas bancadas e identificar seus parâmetros de operação;
2. Definir o *hardware* mais apropriado para a leitura e a amostragem dos valores medidos conforme as especificações dos sinais de saída dos transdutores, buscando o melhor custo-benefício em termos de precisão, operabilidade e investimento financeiro;
3. Projetar um sistema de adaptação das conexões dos transdutores para que se tenha acesso aos terminais de alimentação e de sinais;
4. Definir uma plataforma de desenvolvimento para a programação do sistema;

5. Implementar o dispositivo de medição;
6. Realizar ensaios e testes de validação;

1.2 ORGANIZAÇÃO ESTRUTURAL DO TRABALHO

Este TCC está estruturado da seguinte maneira: no Capítulo 1, apresenta-se a introdução do trabalho; no Capítulo 2, faz-se a contextualização, com a descrição das bancadas didáticas e dos sensores de torque disponíveis; no Capítulo 3, aborda-se o desenvolvimento do projeto do dispositivo de medição dos sinais de torque, sendo definidos os requisitos de *hardware* e *software* necessários; no Capítulo 4, descreve-se o processo de implementação e montagem do *hardware* do dispositivo e as adaptações realizadas na bancada; no Capítulo 5 é apresentado o desenvolvimento do *software*, bem como é descrita a estrutura do código-fonte; no Capítulo 6, apresentam-se os ensaios e testes de validação realizados; e, no Capítulo 7, são expostas as conclusões do trabalho, além de sugestões para trabalhos futuros.

Ademais, neste trabalho são fornecidos também dois apêndices e um anexo. No Apêndice A é apresentado um guia prático de utilização das ferramentas de desenvolvimento empregadas no projeto; no Apêndice B é apresentado, em sua integralidade, o código-fonte desenvolvido; e no Anexo A é apresentada uma cópia de um tutorial de utilização das ferramentas de desenvolvimento, fornecido por um grupo chamado "*Random Nerd Tutorials*", que auxilia alunos e engenheiros em projetos diversos.

CAPÍTULO 2

CONTEXTUALIZAÇÃO

Neste capítulo, apresenta-se a contextualização do trabalho, destacando as bancadas de ensaios de máquinas elétricas do LABMAQ e suas principais características. Além disso, discorre-se sobre o transdutor de torque, presente em cada uma das bancadas, que é o foco deste TCC. São descritas as características nominais do transdutor, seu princípio de funcionamento, e as alternativas possíveis para realizar a medição de torque com base na instalação já existente, buscando a melhor relação custo-benefício possível.

2.1 BANCADAS DE ENSAIOS DE MÁQUINAS ELÉTRICAS

O LABMAQ é um laboratório de ensino onde são ministradas disciplinas relacionadas à conversão eletromecânica de energia e à eletrotécnica em geral. Entre diversos equipamentos, o LABMAQ dispõe de três bancadas didáticas para ensaios com motores e geradores, conforme mostra a Figura 2.1. Cada bancada é composta por três máquinas rotativas, interligadas em um único eixo: um motor trifásico de rotor bobinado de 5 kW, um gerador síncrono de polos salientes de 5 kVA, e um motor de indução de 5,5 kW controlado por um conversor de frequência, com ajuste de velocidade. O transdutor de torque está localizado no segmento do eixo que conecta mecanicamente o gerador síncrono ao motor de indução controlado pelo conversor de frequência.

Entre os modos de operação da bancada, é possível realizar ensaios com o motor de rotor bobinado, utilizando o motor de indução controlado pelo conversor de frequência como carga mecânica. Nessa condição, o conversor opera como controle de carga, sendo ajustado manualmente por meio do painel de comando, onde também são feitas as medições elétricas [1]. O motor de rotor bobinado é acionado por um varivolt trifásico, e a partida conta com o auxílio de um reostato. Com o uso de um sistema de leitura e amostragem de torque, seria possível medir a potência mecânica transferida do motor para a carga, que é algo essencial para análise de desempenho e rendimento do motor. Entretanto, devido à ausência de tal instrumentação, esse tipo de ensaio não é realizado, limitando algumas das aplicações da bancada.

Figura 2.1 – Bancada de ensaios de máquinas elétricas do LABMAQ.



Fonte: Autor.

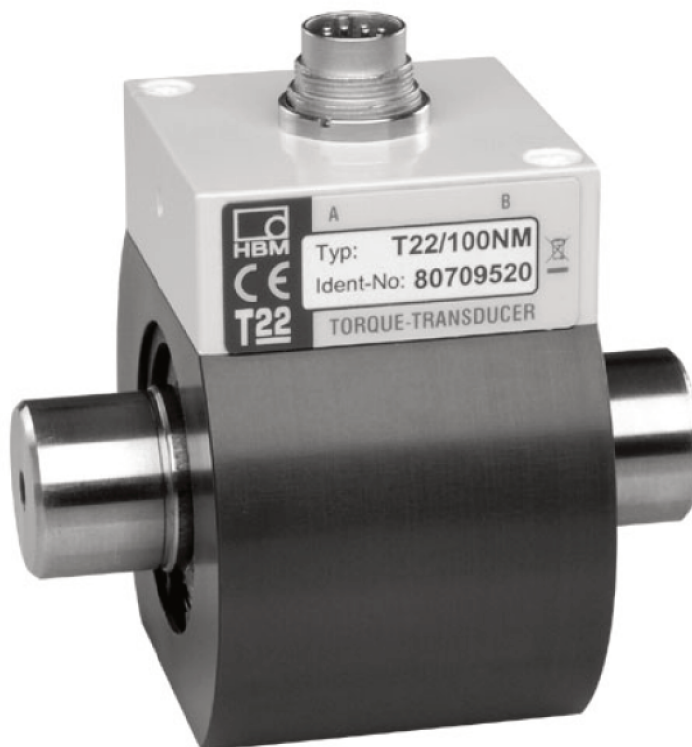
Nos ensaios com o gerador síncrono, o motor com controle de velocidade atua como a máquina primária. Por meio do painel de comando, são realizados os acionamentos necessários, incluindo a excitação do campo do gerador e as medições elétricas da potência gerada. Nessa configuração, a medição de torque pelo transdutor permitiria determinar a potência mecânica transferida da máquina primária para o gerador. Essa informação é útil para obter características de eficiência da máquina síncrona e serve também como um parâmetro didático valioso para exemplificar numericamente o processo de conversão eletromecânica de energia.

2.2 TRANSDUTORES DE TORQUE INSTALADOS NAS BANCADAS

Os transdutores de torque instalados nas bancadas de ensaios são do modelo T22 da fabricante HBM (Figura 2.2). Este transdutor é um dispositivo eficiente para medições de torque, tanto dinâmicas quanto estáticas, utilizando tecnologia de transmissão sem contato. Essa característica torna o T22 uma solução viável para a medição precisa de

torque em componentes rotativos e não-rotativos, sendo indicado para uma ampla gama de aplicações laboratoriais. Além disso, o T22 se destaca por sua fácil instalação, mesmo em espaços restritos. Uma de suas principais vantagens é a eletrônica integrada, que elimina a necessidade de amplificadores externos. Ele conta com duas saídas analógicas, uma de corrente e outra de tensão, o que facilita a integração com sistemas de monitoramento e medição.

Figura 2.2 – Transdutor de torque T22 - HBM.



Fonte: HBM T22 Datasheet [2].

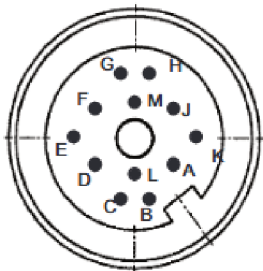
2.2.1 Especificações técnicas

O Sensor T22 do fabricante europeu HBM tem por características técnicas uma alimentação CC que pode variar de 11,5 V a 30 V com consumo inferior a 2,4 W. Os transdutores das bancadas didáticas do LABMAQ são capazes de medir até 50 N.m de torque rotacional, girando em ambos os sentidos de rotação. Os sinais de saída dos transdutores podem ser de tensão ou de corrente, operando nas faixas de 0 ± 5 V e de 10 ± 8 mA, respectivamente. A classe de precisão do transdutor é de 0,5%. Em torque zero, tem-se $0 \pm 0,2$ V na saída de tensão e $10 \pm 0,2$ mA na saída de corrente.

2.2.2 Conexões elétricas

O transdutor T22 possui um conector de 12 vias para o cabeamento da alimentação e dos sinais de medição. A relação da pinagem do conector é mostrada na Figura 2.3.

Figura 2.3 – Pinagem do transdutor de torque T22 - HBM.

	Pin	Assignment	Wire color
	A	No function	bk
	B	No function	rd
	C	Torque measurement signal, voltage output $\pm 5\text{ V}$	br
	D	Ground torque, voltage output	wh
	E	Ground supply and current output	ye
	F	Supply voltage $+11.5 \dots 30\text{ V}$	vt
	G	No function	gn
	H	No function	pk
	J	No function	gy
	K	No function	gy/pk
	L	Torque measurement signal, current output $10 \pm 8\text{mA}$	bu/rd
	M	No function	bu

Fonte: HBM T22 - Instruções para montagem [3].

2.3 ESCOPO DO TRABALHO

O presente Trabalho de Conclusão de Curso tem como objetivo principal aplicar os conhecimentos adquiridos ao longo da graduação em Engenharia Elétrica na resolução de um problema prático identificado no Laboratório de Máquinas e Acionamentos Elétricos (LABMAQ). O problema em questão refere-se à impossibilidade de realizar medições de torque com as bancadas didáticas do laboratório mesmo que existam transdutores de torque instalados em cada uma das três bancadas. Essa limitação decorre da ausência de um dispositivo adequado para a leitura e amostragem dos sinais provenientes desses sensores.

O trabalho propõe o desenvolvimento e a implementação de um dispositivo de baixo custo que permita realizar medições confiáveis de torque utilizando os transdutores já existentes nas bancadas. Esse dispositivo visa atender às necessidades do laboratório, fornecendo uma solução prática e acessível, com potencial para aprimorar as atividades didáticas e experimentais realizadas no LABMAQ.

CAPÍTULO 3

PROJETO DO DISPOSITIVO AMOSTRADOR DE TORQUE

Neste capítulo é apresentado o projeto do dispositivo amostrador de torque. São apresentados também os requisitos de *hardware* e *software* do projeto, bem como são descritos todos os elementos e ferramentas de desenvolvimento necessárias para a realização do trabalho.

3.1 *HARDWARE* PRINCIPAL

O *hardware* do dispositivo foi projetado com base nos critérios de baixo custo, praticidade de operação e facilidade de implementação. Assim, definiu-se o *hardware* principal do dispositivo, sendo composto por: microcontrolador, *display* LCD, módulo conversor de sinal de corrente em tensão, módulo conversor analógico digital e a fonte de alimentação.

3.1.1 Microcontrolador

Microcontroladores são pequenos computadores integrados em um único chip. Eles combinam, em um mesmo sistema, diversos componentes essenciais, tais como:

1. **CPU (Unidade Central de Processamento)** - Responsável pelo processamento de dados e execução de instruções;
2. **Memórias** - Utilizadas para armazenar dados temporários ou permanentes, além de suportar o funcionamento do processador;
3. **Periféricos de entrada e saída** - Dispositivos e conexões que permitem a comunicação do microcontrolador com o ambiente externo.

Por serem programáveis, os microcontroladores são amplamente utilizados para realizar tarefas específicas, como ler sensores, controlar motores ou processar sinais de maneira confiável e econômica.

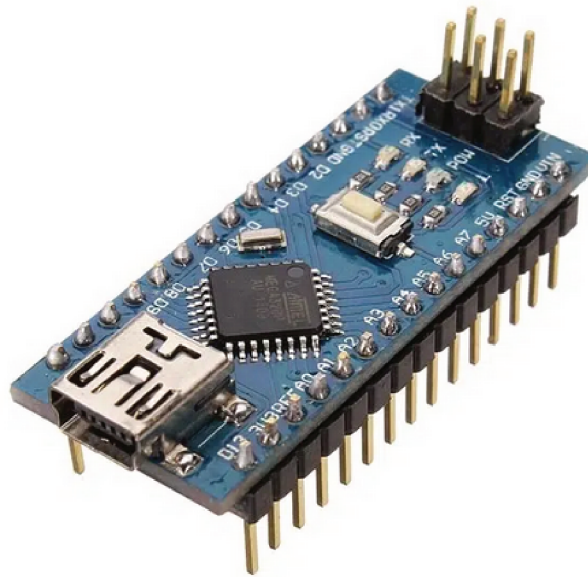
3.1.1.1 Plataforma de prototipagem - Arduino

O Arduino é uma plataforma de prototipagem eletrônica composta por placas modulares que seguem os princípios de *hardware* e de *software* de código aberto, permitindo sua livre produção, modificação e comercialização. O Arduino foi desenvolvido inicialmente como uma ferramenta educacional, com o objetivo de contribuir no aprendizado de eletrônica e programação. Após o grande sucesso das primeiras placas, a plataforma evoluiu para incluir uma ampla gama de versões e capacidades. Essas variações geralmente utilizam microcontroladores da família *AVR ATmega*, produzidos originalmente pela Atmel (hoje Microchip), em configurações de placa única com diferentes interfaces, tamanhos e conexões. Cada versão foi projetada para atender a requisitos específicos de dispositivos como LEDs, motores, sensores e outros componentes eletrônicos. Atualmente, o Arduino conta com uma robusta comunidade global que disponibiliza uma vasta quantidade de tutoriais e bibliotecas prontas. Essa característica torna a plataforma uma escolha popular para o desenvolvimento dos mais variados tipos de projetos.

3.1.1.2 Arduino Nano

Para o desenvolvimento deste trabalho, optou-se por empregar uma placa Arduino Nano, que é capaz de atender todos os requisitos necessários para a implementação do projeto, com um tamanho reduzido e custo relativamente baixo (em torno de sete dólares americanos na data de elaboração deste trabalho). O Arduino Nano é fabricado em uma placa de circuito impresso com dimensões de 45 mm por 18 mm, onde são montados três circuitos integrados: o microcontrolador ATmega328, um conversor USB (*Universal Serial Bus*) para comunicação UART (*Universal Asynchronous Receiver/Transmitter*) e um regulador de tensão linear. A placa também inclui um cristal de quartzo que serve como referência de *clock* para o microcontrolador, além de quatro LEDs indicadores: um de uso geral, um para indicar a alimentação, e dois para comunicação. Ela possui ainda botão de *reset*, conectores USB utilizados tanto para alimentação quanto para *upload* de *firmware*, e espaço para conectores tipo barra de pinos, que fornecem acesso aos contatos e às portas de entrada e saída do microcontrolador. A Figura 3.1 mostra uma imagem do Arduino Nano empregado no projeto.

Figura 3.1 – Placa Arduino Nano - V3.



Fonte: Make Hero - Arduino Nano [4].

3.1.2 Display LCD

O uso de um *display* alfanumérico é uma solução prática para exibir informações de maneira simples e clara ao usuário. Neste projeto, foi utilizado um *display* LCD 16x2 com conversor I2C, uma escolha amplamente empregada em projetos devido à sua funcionalidade e acessibilidade econômica. O modelo selecionado é apresentado na Figura 3.2. Atualmente, seu custo é de cerca de seis dólares americanos, que é um valor competitivo considerando suas vantagens em relação a alternativas mais complexas.

O *display* empregado possui uma interface de cristal líquido retroiluminada que suporta até 32 caracteres distribuídos em duas linhas e 16 colunas. Cada caractere é controlado por uma matriz de 5x8 pontos, manipulada pelo chip controlador HD44780, fabricado pela Hitachi. Esse controlador interpreta os comandos e dados enviados pelo microcontrolador, convertendo-os em informações visuais no formato ASCII.

Esta versão do *display* inclui um conversor I2C (*Inter-Integrated Circuit*) que oferece uma redução significativa na complexidade de integração. Assim, este módulo permite que as 16 conexões paralelas do *display* sejam gerenciadas por apenas dois terminais para comunicação por meio do protocolo I2C. Trata-se dos terminais SCL (*Serial Clock*) que sincroniza a transmissão de dados, e do SDA (*Serial Data*), que transporta os dados entre os dispositivos conectados.

Esse protocolo permite conectar vários dispositivos ao barramento, compartilhando os mesmos pinos. No barramento I2C, o microcontrolador atua como o dispositivo mestre, gerenciando a comunicação com os dispositivos escravos, que possuem endereços específicos. No caso do amostrador de torque, o módulo I2C do *display* traduz os comandos envia-

dos pelo protocolo serial para sinais paralelos compreensíveis pelo controlador HD44780, simplificando a comunicação entre o microcontrolador Arduino Nano e o *display*.

Figura 3.2 – Módulo de *display* LCD 16x2 para Arduino.



Fonte: Make Hero - *Display* LCD [5].

3.1.3 Módulo conversor analógico digital - ADS1115

Um conversor analógico-digital (ADC) é um componente eletrônico que converte sinais analógicos (contínuos no tempo e na amplitude) em dados digitais (discretos no tempo e em níveis). Ele é essencial para permitir que sistemas digitais, como microcontroladores, interpretem informações de sensores e dispositivos analógicos. Os ADCs funcionam coletando valores de sinais analógicos em intervalos regulares, definidos por uma taxa de amostragem. Cada amostra analógica coletada é quantizada, ou seja, mapeada para o valor digital mais próximo dentro da resolução do ADC. A resolução, medida em bits, define o número de níveis discretos possíveis. Os níveis quantizados são convertidos em valores binários para processamento digital, em geral, feito pelo microcontrolador.

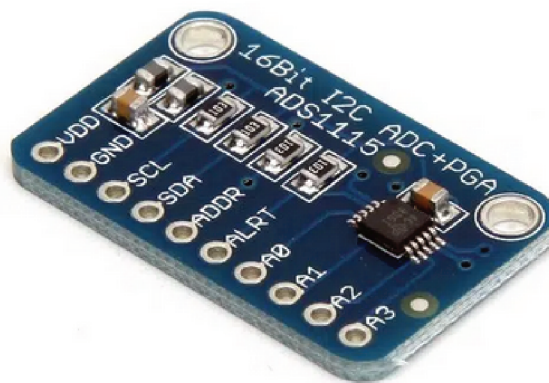
Apesar do microcontrolador ATmega328, presente no Arduino Nano, já possuir um conversor ADC interno de 10 bits, neste projeto foi empregado o ADC ADS1115 da Texas Instruments apresentado na Figura 3.3, que é um módulo comumente comercializado em pequenas placas para prototipagem. O ADS1115 oferece recursos avançados que o tornam mais adequados para sistemas que exigem maior precisão, flexibilidade, capacidade de medição e custo acessível (em torno de quatro dólares americanos). O módulo realiza conversões de sinais analógicos em digitais com 16 bits de resolução, oferecendo um baixo consumo de energia e comunicação I2C, o que facilita sua integração no projeto.

A resolução de 16 bits permite obter até 65536 níveis de quantização de tensão (2^{16}). Para comparação, conforme mencionado anteriormente, o ADC do ATmega328 oferece uma resolução de apenas 10 bits, o que corresponde a apenas 1024 níveis de tensão (2^{10}). Além disso, o ADC do ATmega328 mede apenas tensões em uma escala fixa de 5 V e não possui amplificador interno, exigindo circuitos adicionais de amplificação para sinais de

baixa amplitude. Já o ADS1115 oferece um amplificador de ganho programável (PGA) integrado, que pode ajustar a faixa de entrada para as seguintes escalas: 0,256 V; 0,512 V; 1,024 V; 4,096 V; e 6,144 V. Vale salientar que a entrada do ADS1115 aceita apenas tensões positivas, havendo uma margem de tolerância até $-0,3$ V.

Outra questão importante é o fato de que a alimentação geral do microcontrolador é a referência do seu conversor analógico-digital interno. Isto faz com que qualquer ruído nesse barramento possa influenciar nas leituras. Em geral, a alimentação também é compartilhada por outros blocos do microcontrolador e do sistema, o que amplia ainda mais as fontes de ruído. Já o ADS1115, por ser um módulo externo, oferece maior proteção contra ruídos, dado a possibilidade de isolar as partes digital e analógica do sistema.

Figura 3.3 – Módulo conversor analógico digital - ADS1115.



Fonte: Make Hero - Módulo ADS1115 [6].

3.1.4 Módulo conversor de sinal de corrente em tensão - HW 685

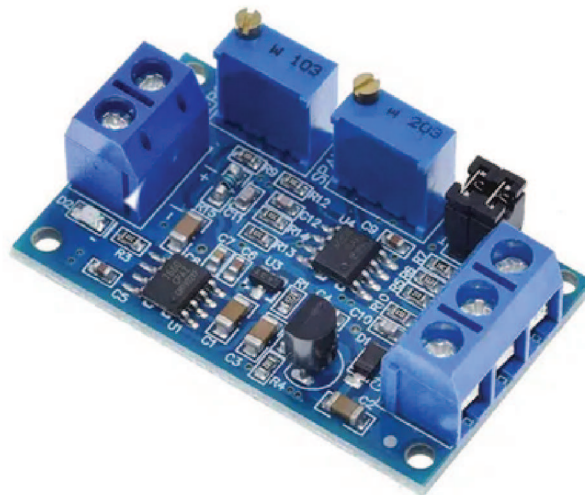
Conforme apresentado no capítulo 2, o transdutor T22 oferece duas opções de sinal proporcional ao torque medido: saída de tensão ou de corrente. Quando configurado para saída de tensão, o transdutor fornece valores que variam de -5 V a $+5$ V. Sinais positivos indicam rotação no sentido horário e sinais negativos indicam rotação no sentido anti-horário. Essa característica apresenta um desafio significativo para os sistemas de conversão analógico-digital pois, geralmente, eles operam em faixas de tensões positivas, como é o caso do ADS1115. Isso torna a saída de tensão do transdutor de torque incompatível, já que sinais negativos não podem ser processados diretamente e valores fora do intervalo suportado poderiam danificar o ADC. Para adaptar a saída de tensão do transdutor à faixa do conversor ADC, seria necessário implementar um circuito somador para o deslocamento do sinal, envolvendo amplificadores operacionais. Esse arranjo não apenas aumentaria a complexidade do sistema como também poderia interferir na precisão e confiabilidade das medições devido a fatores como ruído e consumo adicional energia pelo circuito.

A solução encontrada foi utilizar a saída de corrente do transdutor T22 e aplicar

um conversor de sinal para se obter um sinal proporcional de tensão dentro de uma faixa apropriada. Trata-se do HW-685, que é um módulo eletrônico projetado para converter sinais de corrente em sinais de tensão, amplamente utilizado em sistemas de medição e controle (Figura 3.4). O módulo HW-685 recebe o sinal de corrente do transdutor T22 pelo terminal de entrada e, internamente, o circuito converte a corrente em tensão ao fazê-la passar por um resistor de alta precisão. Essa tensão é então condicionada por amplificadores operacionais que garantem linearidade e a adaptam para o intervalo desejado, com as seguintes opções: 0 – 3,3 V, 0 – 5 V ou 0 – 10 V, dependendo das necessidades do sistema. Este conversor foi projetado para operar com sinais de corrente na faixa de 4 mA até 20 mA.

O HW-685 oferece ainda a vantagem de proporcionar ajustes finos na tensão de saída, facilitados por *trimpots* integrados, o que permite a calibração do sistema para garantir a correspondência entre os sinais do transdutor e a faixa operacional do conversor ADC. Com um *design* que inclui componentes como amplificadores operacionais, reguladores LDO (*Low Dropout Regulator*) e resistores de alta precisão, o módulo oferece versatilidade com um custo acessível (em torno de quatro dólares americanos). Para a aplicação deste conversor no projeto são necessários alguns ajustes de escalas e calibrações que são detalhadas na seção 4.2 do capítulo 4.

Figura 3.4 – Módulo conversor de sinal de corrente em tensão HW-685.



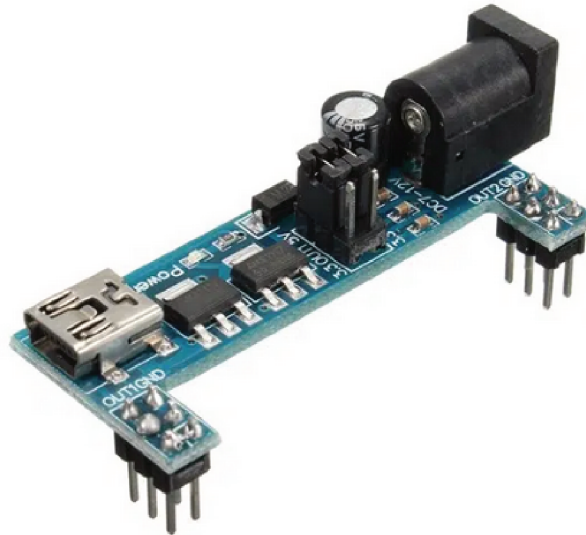
Fonte: Make Hero - Módulo HW 685 [7].

3.1.5 Fontes de Alimentação

A alimentação geral dos componentes de *hardware* do projeto é feita mediante o uso de uma fonte externa de 12 V, com capacidade de corrente até 1 A (custo aproximado de cinco dólares americanos). Para os módulos alimentados em 5 V, optou-se por empregar a fonte ajustável MB102, apresentada na Figura 3.5, que atualmente é encontrada no

mercado por aproximadamente três dólares. Esta fonte recebe a alimentação em 12 V e fornece opções de saída em 3,3 V ou em 5 V. No caso deste projeto, utiliza-se a segunda opção.

Figura 3.5 – Fonte Ajustável de 12 V para 3,3 V ou 5 V - MB102.



Fonte: Make Hero - Fonte ajustável MB102 [8].

3.2 HARDWARE PERIFÉRICO

O *hardware* periférico do projeto compreende os componentes estruturais que são utilizados para a o desenvolvimento do dispositivo amostrador de torque. São eles: cabos e conectores; gabinete; e componentes acessórios como botões, diodos e resistores.

3.2.1 Cabos e conectores

As conexões gerais dos módulos e componentes do *hardware* são feitas com cabos do tipo *jumpers*, ideais para o desenvolvimento de protótipos. A fonte de alimentação 12 V é feita via *plug P4* que se conecta no dispositivo diretamente na placa da fonte de 5 V (MB102). Desta entrada derivam as alimentações para os demais módulos que operam em 12 V. No caso do Arduino Nano, a entrada também é P4, portanto, empregou-se um conector *plug P4* macho com bornes.

A conexão entre o dispositivo amostrador de torque e o transdutor T22 é feita por meio de um cabo de sinais de três vias, blindado com malha de cobre. Este cabo fornece alimentação em 12 V para o transdutor através de duas vias e, na terceira via, percorre o sinal proporcional de corrente, cujo retorno é pelo próprio GND da fonte. As conexões são realizadas por conectores Mike de três vias, conforme mostra a Figura 3.6. Como o conector do transdutor de torque é uma entrada padrão de 12 vias, é necessário fazer uma

adaptação na pinagem para fazer o contato com os três terminais de interesse e ligá-los ao conector Mike fêmea, apresentado à esquerda na figura.

Figura 3.6 – Conector Mike para a ligação entre o medidor e o transdutor de torque.



Fonte: Autor.

3.2.2 Gabinete

Para a composição estrutural do dispositivo, empregou-se um gabinete de prototipagem disponível no laboratório, conforme mostra a Figura 3.7.

Figura 3.7 – Gabinete do dispositivo de medição.



Fonte: Autor.

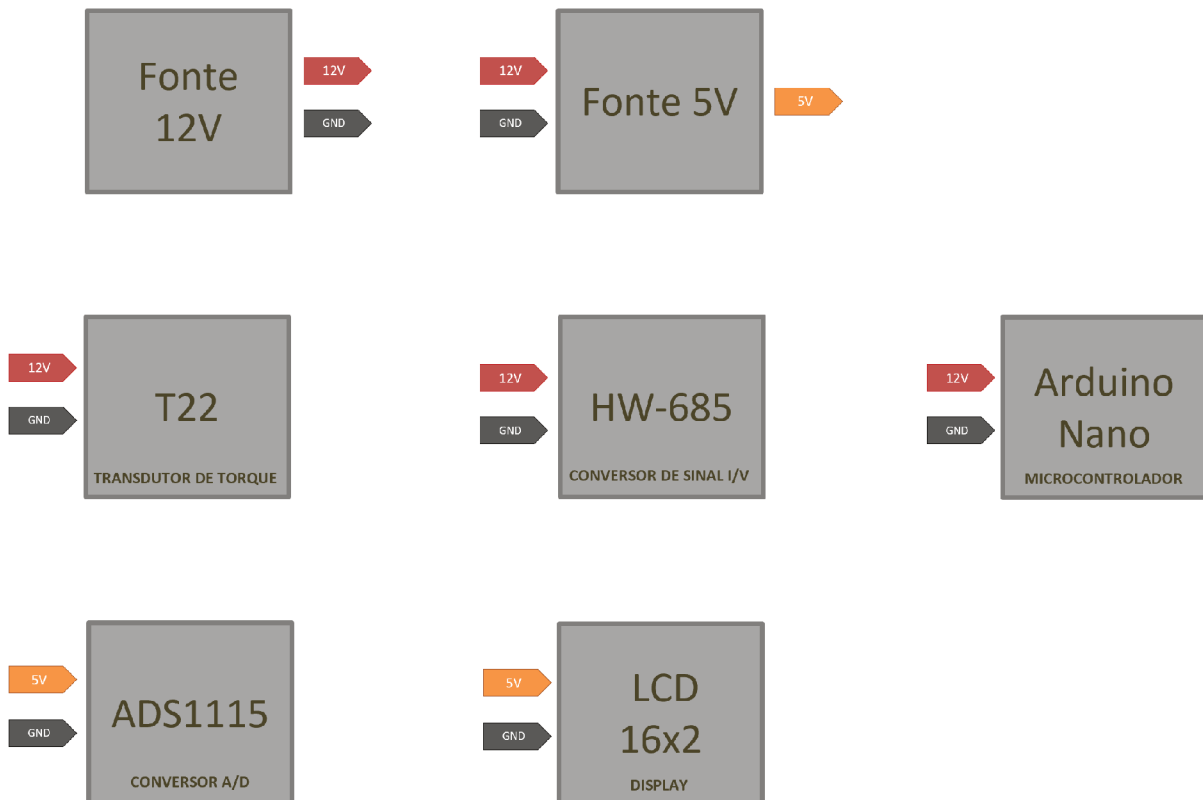
3.2.3 Componentes acessórios

Os componentes acessórios do projeto são: um botão de pulso e um resistor de $1\text{ k}\Omega$ para o circuito de calibração; um resistor de $240\ \Omega$ para a limitação do sinal de corrente do transdutor T22; e um diodo TVS de $5,6\text{ V}$ para proteção do sistema. Os detalhes são apresentados mais adiante ao longo do detalhamento do projeto.

3.3 DIAGRAMA ESQUEMÁTICO DO PROJETO

A Figura 3.8 mostra o diagrama de potência (alimentação) dos elementos que compõem o projeto. Conforme apresentado anteriormente, uma fonte chaveada de 12 V comercial fornece a alimentação geral para o sistema. Esta fonte alimenta diretamente os seguintes componentes: o transdutor de torque T22; o módulo conversor de sinal corrente/tensão HW-685; e o microcontrolador Arduino Nano. Os demais componentes, tais como o conversor analógico/digital ADS1115 e o módulo *display* LCD16x2, são alimentados em 5 V. Portanto, é utilizada a fonte secundária de 5 V que também é alimentado pela fonte de 12 V.

Figura 3.8 – Diagrama esquemático de alimentação dos componentes do dispositivo.



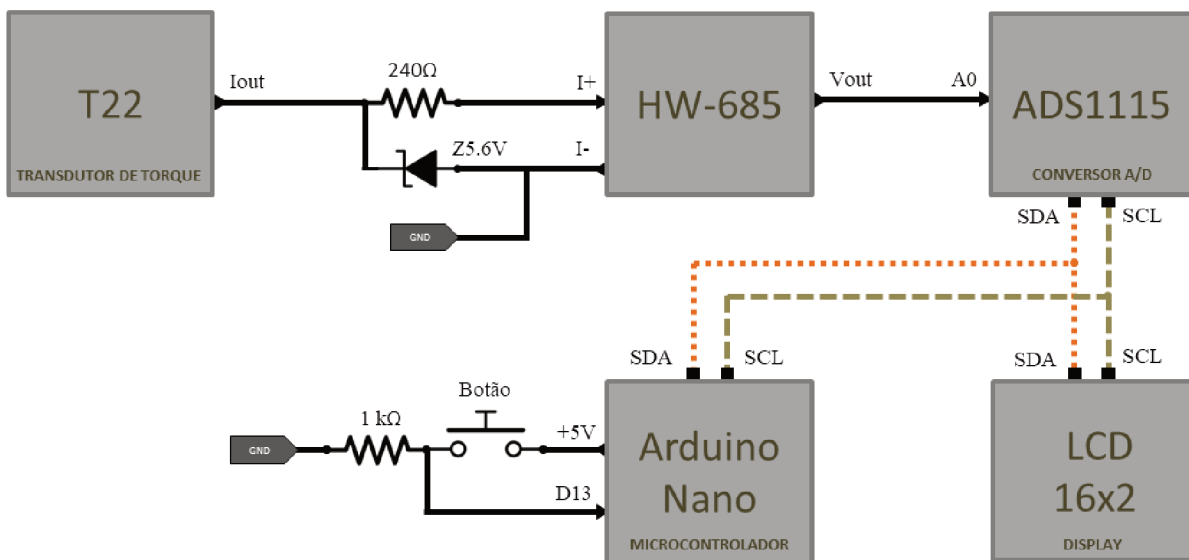
Fonte: Autor.

A Figura 3.9 ilustra o diagrama de conexões entre os diversos elementos do projeto. O sinal de corrente fornecido pelo transdutor de torque T22 é transformado em um sinal de tensão pelo conversor HW-685. A corrente sai do transdutor pelo terminal I_{out} e entra no terminal $I+$ do HW-685, sendo drenada pelo GND que é conectado ao terminal $I-$. Entre os terminais I_{out} e $I+$, conecta-se o resistor de referência de $240\ \Omega$, que é um requisito de limitação do sinal de corrente do transdutor T22. Ademais, é adicionado um diodo TVS entre I_{out} e GND para proteção e supressão de surtos transientes.

O sinal de tensão proveniente do HW-685 (V_{out}) é conectado ao conversor analógico digital ADS1115 pela pino de leitura A0. Portanto, o ADS1115 recebe a tensão transformada que equivale à leitura do sensor de corrente e faz a conversão de sinal analógico para digital que é enviada para o microcontrolador por um barramento serial com protocolo I2C. O Arduino Nano corresponde ao microcontrolador que recebe as leituras de forma digital pelo barramento I2C e faz os processamentos necessários para o projeto. Por fim, faz-se o envio de dados de volta para o barramento serial direcionado a um *display* LCD, exibindo as informações e os resultados ao usuário.

O sistema de calibração do dispositivo é realizado por meio da entrada digital D13 do Arduino que monitora o status de um botão conforme a alteração de nível lógico. Quando o botão não está pressionado, a entrada D13 permanece aterrada pelo GND. Uma vez que o botão é pressionado, a entrada digital recebe o sinal de 5 V gerado pelo Arduino através do resistor de 1 k Ω que separa o terminais do GND e de +5V.

Figura 3.9 – Diagrama esquemático de ligações do dispositivo.



Fonte: Autor.

3.4 REQUISITOS DE SOFTWARE

O *software* é a parte lógica de um sistema computacional que permite a execução de tarefas específicas pelo *hardware*. Trata-se de um conjunto de instruções ou programas que controlam e orientam o funcionamento de determinado dispositivo eletrônico [9]. Portanto, o software desempenha um papel fundamental neste projeto, sendo responsável pela configuração, controle e interpretação dos sinais capturados pelos sensores de torque. Ele atua como o elo de integração entre os diversos componentes de *hardware*, garantindo

que os dados obtidos pelos transdutores sejam processados e exibidos de forma clara e precisa em uma interface acessível ao usuário.

O *firmware* é um tipo especial de software que está intimamente ligado ao *hardware*. Ele é responsável por fornecer as instruções de baixo nível que permitem que o *hardware* funcione corretamente. Assim, o *firmware* é desenvolvido para dispositivos específicos, como microcontroladores, sensores e módulos eletrônicos. Tem funções mais básicas, como inicializar componentes de *hardware* ou controlar operações específicas [10]. No contexto do Arduino Nano, o *firmware* é o programa gravado no microcontrolador ATmega328, que instrui o dispositivo sobre como capturar, processar e exibir os sinais dos sensores.

Para o desenvolvimento do *firmware* que será executado no Arduino Nano, é necessário atender a uma série de requisitos que envolvem tanto o ambiente de desenvolvimento quanto às ferramentas utilizadas. O principal objetivo do *software* é fornecer uma solução eficiente e acessível que simplifique o uso do dispositivo e permita a sua calibragem e operação de maneira intuitiva.

3.4.1 Ambiente de desenvolvimento

O ambiente de desenvolvimento é um conjunto de ferramentas, configurações e recursos usados por desenvolvedores para criar, testar, depurar e implantar um software. Ele fornece tudo o que é necessário para escrever o código e garantir que ele funcione corretamente em um sistema ou dispositivo. Neste projeto, optou-se pela utilização do *Visual Studio Code* (VSCode) com o Ambiente Integrado de Desenvolvimento (IDE), em conjunto com a extensão PlatformIO, que oferece suporte a sistemas embarcados, incluindo o Arduino Nano.

3.4.2 Framework de Desenvolvimento

O *framework* é uma estrutura de ferramentas e bibliotecas pré desenvolvidas que facilitam a criação do software. Ele fornece componentes reutilizáveis e funcionalidades pré implementadas para ajudar os desenvolvedores a construir seus programas de maneira mais eficiente e padronizada. Serve como uma base para o desenvolvimento de software, reduzindo a necessidade de criar tudo do zero

No caso do Arduino, o *framework Arduino* é uma camada de abstração que permite programar o microcontrolador de forma mais simples, sem a necessidade de trabalhar diretamente com registradores ou escrever código em linguagem *assembly*. Ele inclui bibliotecas que facilitam a interação com componentes como *displays*, sensores e conversores.

3.4.3 Estrutura do Código

O programa implementado no Arduino Nano deve ser capaz de realizar as seguintes funções:

1. Capturar os sinais dos transdutores de torque e convertê-los para valores digitais;
2. Processar os dados medidos, aplicando eventuais calibrações;
3. Exibir os resultados em tempo real no *display* LCD;
4. Possibilitar uma rotina de calibração para ajustes de *offset* (torque nulo).

Para oferecer uma visão geral do funcionamento do programa, na próxima seção é apresentado um fluxograma com as etapas principais do algoritmo de operação do sistema. Esse fluxograma servirá como referência para compreender a lógica de operação do dispositivo, sendo que os detalhes da implementação são abordados com maior profundidade no capítulo 5.

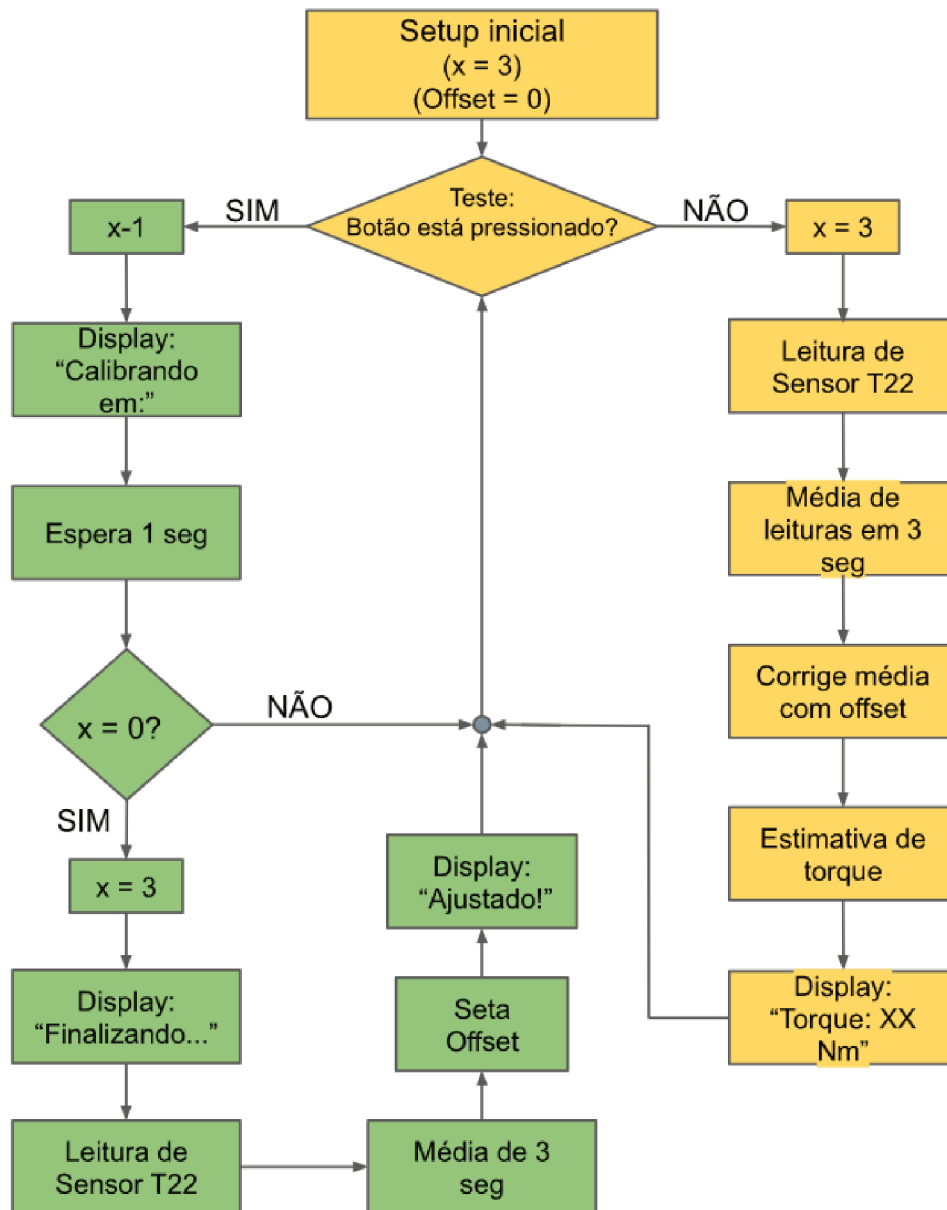
3.5 FLUXOGRAMA DO ALGORITMO

Sob o ponto de vista do usuário, o dispositivo deve funcionar da seguinte maneira: conecta-se o amostrador a um determinado transdutor de torque T22, inicializa-se o dispositivo, e a tela LCD deve exibir do valor de torque correspondente à leitura do sinal. Quando a bancada de ensaios estiver desligada, o valor de torque medido deve ser nulo. No entanto, os sinais de medição dos transdutores costumam apresentar um *offset* indesejável⁽¹⁾. Ou seja, um desvio sistemático nas medidas que deve ser corrigido digitalmente pelo programa. Este processo de correção acontece por meio de uma rotina de calibração. Com o sensor isento de qualquer esforço mecânico, o usuário deve pressionar o botão de calibração por três segundos. Então, o amostrador faz a leitura do *offset* e corrige o valor de torque exibido no *display* LCD. Após a calibração, o dispositivo segue a sua rotina de medição e os ensaios podem prosseguir.

O programa desenvolvido segue um algoritmo de tarefas que é ilustrado pelo fluxograma da Figura 3.10. Inicialmente são definidas as duas variáveis do *setup* inicial. A variável "x" é inicializada com valor igual a três e ela servirá como um contador de segundos para habilitar um *loop* de calibração do dispositivo. A variável "*offset*" é inicializada com valor igual a zero e representa literalmente o *offset* de calibração do sinal de torque. Em seguida, é realizado um teste que avalia a condição de uma porta lógica do Arduino. Por meio deste teste é possível identificar se o botão de calibração está pressionado. Caso positivo, inicia-se a rotina de calibração; e caso negativo, inicia-se a rotina de medição.

⁽¹⁾ O *offset* dos transdutores de torque fazem com que haja sempre um sinal de saída que não corresponde exatamente ao ponto de torque igual a zero, mesmo que o transdutor esteja totalmente relaxado (torque nulo). Esta imperfeição é particular de cada sensor, e por este motivo, é necessário realizar a calibração antes de qualquer ensaio

Figura 3.10 – Fluxograma do algoritmo.



Fonte: Autor.

3.5.1 Rotina de calibração

A rotina de calibração é responsável pela correção do valor de *offset* do transdutor de torque. O processo de calibração ocorre após o usuário pressionar o botão de calibração por três segundos ininterruptos. Portanto, dentro da rotina de calibração existe um processo de contagem regressiva de tempo antes que se inicie, de fato, a calibração. Após o primeiro teste positivo, o programa entra em um *loop* de contagem que decrementa a variável "x" em uma unidade, depois exibe no *display* a contagem de tempo, aguarda 1 segundo e então testa se o valor de "x" é igual a zero. Como "x" é iniciado com valor igual a três,

inicialmente o teste será negativo e o botão é testado novamente para garantir se ele continua pressionado. Este *loop* se repete três vezes no total (caso não haja alteração na condição do botão) até que o valor de "x" se torna zero e, então, inicia-se o processo de calibração, que ocorre da seguinte maneira:

1. A variável "x" é reiniciada com o valor igual a três;
2. O *display* exibe a informação de que a calibração está em processo de finalização (o botão não precisa mais estar pressionado);
3. O Arduino lê o sinal do transdutor de torque dentro de um intervalo de três segundos;
4. Calcula-se a média dos valores medidos;
5. Atribui-se o valor da média obtida na variável "*offset*";
6. O *display* exibe a palavra "Ajustado", informando que o processo de calibração finalizou;
7. O programa retoma o teste do botão de calibração.

Uma vez que o processo de calibração foi finalizado, o botão não é mais pressionado e, portanto, inicia-se a rotina de medição.

3.5.2 Rotina de medição

A rotina de medição se inicia sempre que o teste do botão de calibração der negativo, e ocorre da seguinte maneira:

1. A variável "x" é reiniciada com o valor igual a três;
2. O Arduino lê o sinal do transdutor de torque dentro de um intervalo de três segundos;
3. Calcula-se a média dos valores medidos;
4. Subtrai-se o valor do *offset* da média para corrigir a medida;
5. Estima-se o torque correspondente ao valor do sinal medido e corrigido;
6. O *display* exibe o valor de torque obtido;
7. O programa retoma o teste do botão de calibração.

Durante a realização de um ensaio, o dispositivo tende a ficar constantemente no *loop* da rotina de medição, uma vez que não há motivos para pressionar o botão de calibração. Se ocorrer o acionamento acidental do botão, a rotina de calibração irá se

iniciar. Porém, a menos que o botão seja pressionado por três segundos sem interrupção, não haverá comprometimento das medições seguintes em decorrência de uma calibração incorreta já que a alteração do valor original de *offset* acontece apenas após os três segundos. Ou seja, caso o botão seja liberado antes deste período, o processo de calibração é abortado. Por este motivo, no início da rotina de medição, a variável "x" é também reiniciada com valor igual a três.

3.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO 3

Neste capítulo foi apresentado o projeto do dispositivo amostrador de torque, compreendendo os aspectos principais do *hardware*, das conexões e diagramas de alimentação dos componentes. Ademais, foram apresentados os requisitos de *software* necessários e um algoritmo de funcionamento do programa para nortear os procedimentos de implementação do projeto que são abordados nos próximos capítulos.

CAPÍTULO 4

IMPLEMENTAÇÃO DO PROJETO - MONTAGEM DO HARDWARE E ADAPTAÇÕES DA BANCADA

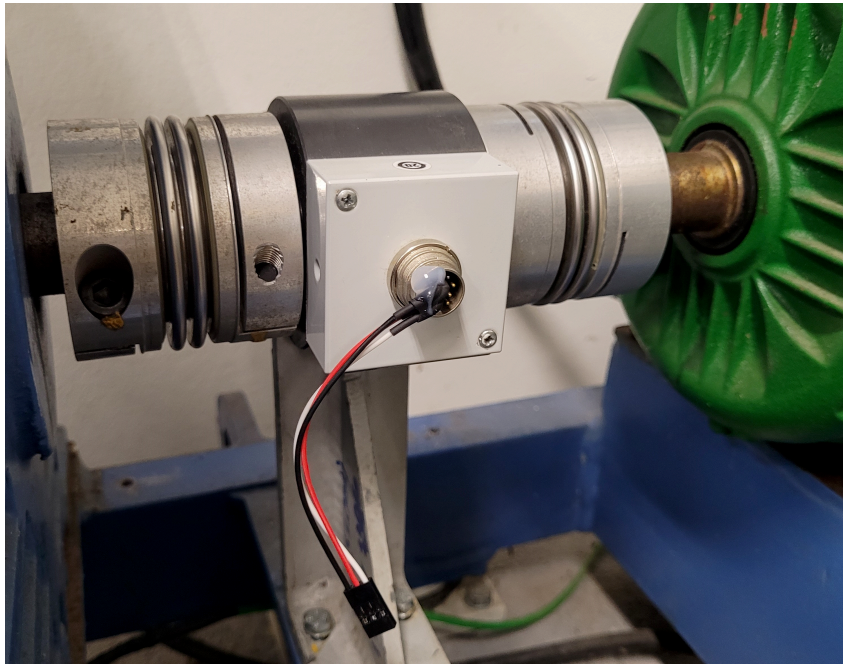
Neste capítulo são apresentados os procedimentos de implementação do *hardware* do projeto, bem como o condicionamento e a calibração do conversor de sinal HW-685. Além disso, são descritas as adaptações realizadas na bancada didática de ensaios de máquinas elétricas do LABMAQ para a viabilização da aplicação do dispositivo no laboratório.

4.1 ADAPTAÇÃO DOS TERMINAIS DE CONEXÃO DOS TRANSDUTORES

Conforme apresentado no capítulo 2, originalmente a bancada didática do LABMAQ foi projetada para receber um sistema de medição e amostragem de torque por meio do transdutor T22 da HBM com a instrumentação apropriada, do mesmo fabricante. No entanto, por questões econômicas, não foi possível adquirir os demais equipamentos, de modo que este trabalho propõe uma solução alternativa de baixo custo. Assim, os padrões de conexão utilizados no projeto do amostrador de torque são distintos do conector disponível no transdutor T22. O dispositivo desenvolvido neste TCC utiliza uma conexão Mike de três vias e foi projetado para adquirir exclusivamente os sinais proporcionais de corrente do T22 que, por sua vez, é fabricado com um conector de 12 vias padronizado pelo fabricante e que fornece adicionalmente os sinais proporcionais de tensão. Portanto, para viabilizar a aplicação do equipamento no laboratório, foi necessário realizar algumas adaptações.

Primeiramente, foram identificados os pinos de alimentação e sinal de corrente do conector do transdutor de torque por meio dos dados do fabricante apresentados da Figura 2.3. Para cada um desses três pinos é feita uma derivação para conexão com barramentos comerciais de pinos, comumente empregados em projetos de placas de circuito impresso. A Figura 4.1 mostra esta adaptação.

Figura 4.1 – Adaptação dos terminais para conexão com o transdutor de torque da bancada.



Fonte: Autor.

Em seguida, foi projetada uma tampa para se encaixar na estrutura de proteção do eixo da bancada possibilitando a fixação do terminal Mike fêmea. A Figura 4.2 mostra três conjuntos do sistema de adaptação, fabricados para cada uma das três bancadas didáticas do LABMAQ.

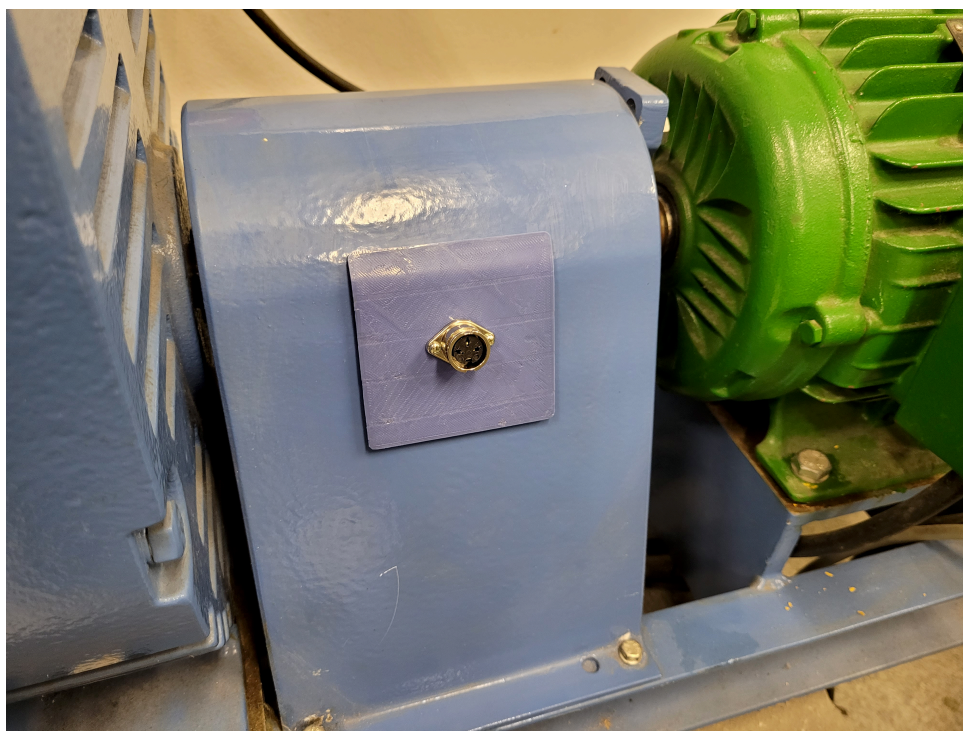
Figura 4.2 – Conectores Mike fêmea e peças de suporte.



Fonte: Autor.

A confecção das tampas foi feita por meio de uma impressora 3D e o material utilizado foi o ABS. Os conectores Mike fêmea são então fixados nas tampas por porcas e parafusos. Os barramentos de pinos são ligados ao T22 e a tampa é encaixada na estrutura da bancada. A Figura 4.3 mostra a adaptação completa.

Figura 4.3 – Estrutura do transdutor de torque da bancada após a adaptação.



Fonte: Autor.

4.2 CONDICIONAMENTO E CALIBRAÇÃO DO CONVERSOR DE SINAL HW-685

Para a aplicação do conversor de sinal HW-685 no projeto, é necessário ponderar alguns aspectos e definir as escalas de operação que serão utilizadas. Conforme apresentado anteriormente, o sinal proporcional de corrente do transdutor de torque varia de 2 mA até 18 mA. No entanto, o conversor HW-685 é projetado para sinais de corrente na faixa de 4 mA até 20 mA. Como a saída de tensão do conversor varia de 0 V até o fundo de escala (3,3 V; 5 V; ou 10 V), quando a entrada do HW-685 for de 2 mA, o valor da tensão será negativo, podendo ocasionar dano ao conversor analógico-digital ADS1115. Além disso, quando o conector do transdutor de torque está desconectado, a entrada de corrente é nula e assim, o risco é ainda maior.

Para resolver esta questão, é necessário fazer algumas ponderações. Primeiramente, deve-se observar que os valores de referência para o sinal de corrente do transdutor representam os valores nominais, portanto, refere-se a uma escala de operação até 50 N.m de torque. No entanto, a bancada didática tem capacidade de impor um torque máximo

de 35 N.m. Assim, a faixa de variação real do sinal de corrente do transdutor é de 4,4 mA até 15,6 mA, com o ponto de torque nulo em 10 mA. Ou seja, não há problemas em se aplicar o conversor HW-685 no que concerne a questão da divergência de faixas de operação. Porém, ainda existe o risco da entrada de corrente ser nula quando o dispositivo estiver desconectado do transdutor.

A solução proposta para esta situação é ajustar o conversor para operar em uma faixa deslocada e expandida de operação através dos ajustes dos valores máximo e mínimo com os *trimpots*. É importante observar que quando o fabricante informa a faixa de operação de 4 mA até 20 mA, significa que o sistema interno de amplificação precisa de uma corrente mínima para garantir a resolução e confiabilidade do sinal convertido. No entanto, é possível configurar o conversor para operar na faixa de 0 mA até 20 mA, assumindo o risco de perda de confiabilidade na faixa de 0 mA até 4 mA. Como a faixa de variação real do sinal de corrente do transdutor na bancada é de 4,4 mA até 15,6 mA, esta proposta se encaixa perfeitamente.

Para realizar este condicionamento do conversor, é necessário ajustar os *trimpots* para que a saída se encontre dentro da faixa desejada. Portanto, deve-se definir a escala do HW-685 e os níveis de tensão da faixa de operação. Considerando as escalas disponíveis para a entrada do ADS1115 apresentadas na seção 3.1.3 do capítulo anterior, optou-se por condicionar o sinal de saída do HW-685 na escala de 5 V e restringir os níveis de tensão da faixa de operação para 0 V até 4 V. Assim, o ADS1115 pode operar na escala de 4,096 V.

Em resumo, o condicionamento do HW-685 é feito com base nos ajustes das fronteiras de operação, considerando que o sinal de entrada (na faixa de 0 mA até 20 mA com ponto nulo em 10 mA), deve corresponder a uma saída de tensão na faixa de 0 V até 4 V, com ponto nulo em 2 V. Este procedimento é realizado conjuntamente com a calibração conforme os testes apresentados a seguir.

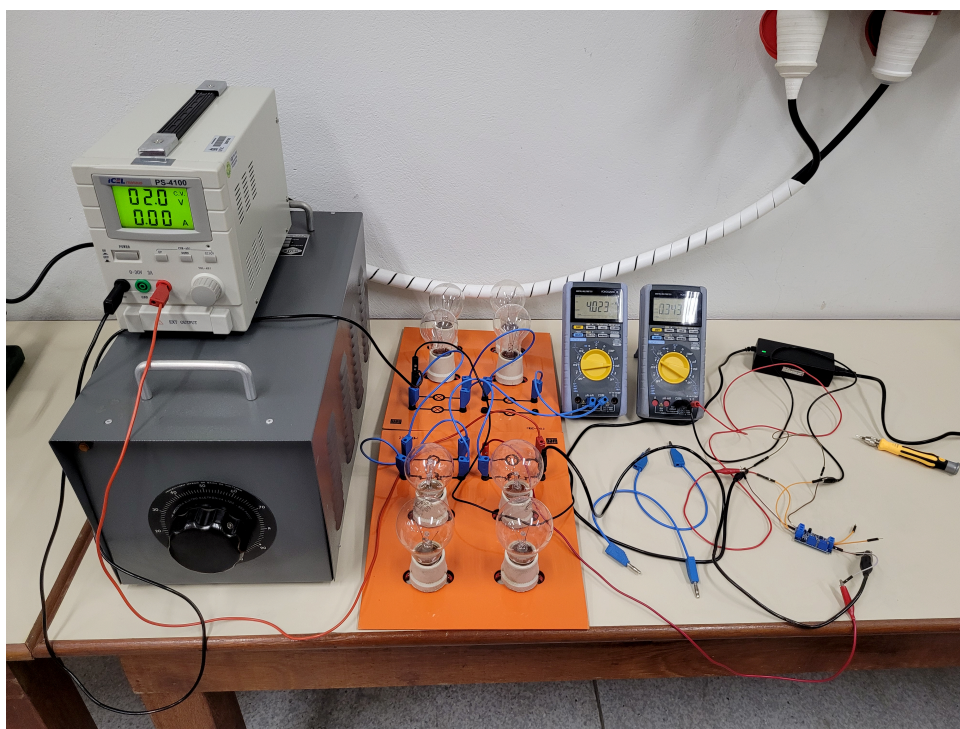
4.2.1 Testes de condicionamento e calibração do conversor de sinal

Para a realização dos testes de condicionamento e calibração do conversor HW-685 utilizou-se uma fonte de tensão em corrente contínua, associada em série com um conjunto de lâmpadas incandescentes para limitar a corrente dentro de uma faixa controlável. Foram utilizados dois multimedidores digitais da Yokogawa modelo TTY720 para as medições da corrente de entrada e da tensão de saída. O *setup* de ensaios pode ser observado na Figura 4.4. O procedimento consiste em estabelecer as correntes limitantes e ajustar os *trimpots* para que a saída de tensão fique calibrada.

A calibração do conversor é feita conforme os parâmetros originais de entrada de corrente, porém, com a correspondência de tensão de 0,2 V para cada 1 mA. Ou seja: o ponto de corrente igual a 4 mA é calibrado para uma saída de tensão igual a 1 V; e o

ponto de 20 mA é calibrado para 4 V.

Figura 4.4 – Testes de condicionamento e calibração do conversor de sinal HW-685.



Fonte: Autor.

Após a realização do procedimento de calibração é feito um ensaio de desempenho do conversor. Utilizando o mesmo *setup* de calibração, são medidos os valores de corrente de entrada e de tensão de saída para pontos distintos a fim de verificar se a correspondência de 0,2 V/mA se mantém. Os resultados obtidos são apresentados na Tabela 4.1, e certificam a linearidade do conversor dentro da faixa de valores reais de aplicação do amostrador de torque.

Tabela 4.1 – Resultados do ensaio de desempenho do conversor HW-685

Corrente (mA)	Tensão (V)
20,002	4,0002
15,001	2,9995
10,000	1,9994
4,999	0,9991
3,998	0,7998
3,001	0,5993
2,001	0,3993
1,002	0,1996
0,000	-0,0259

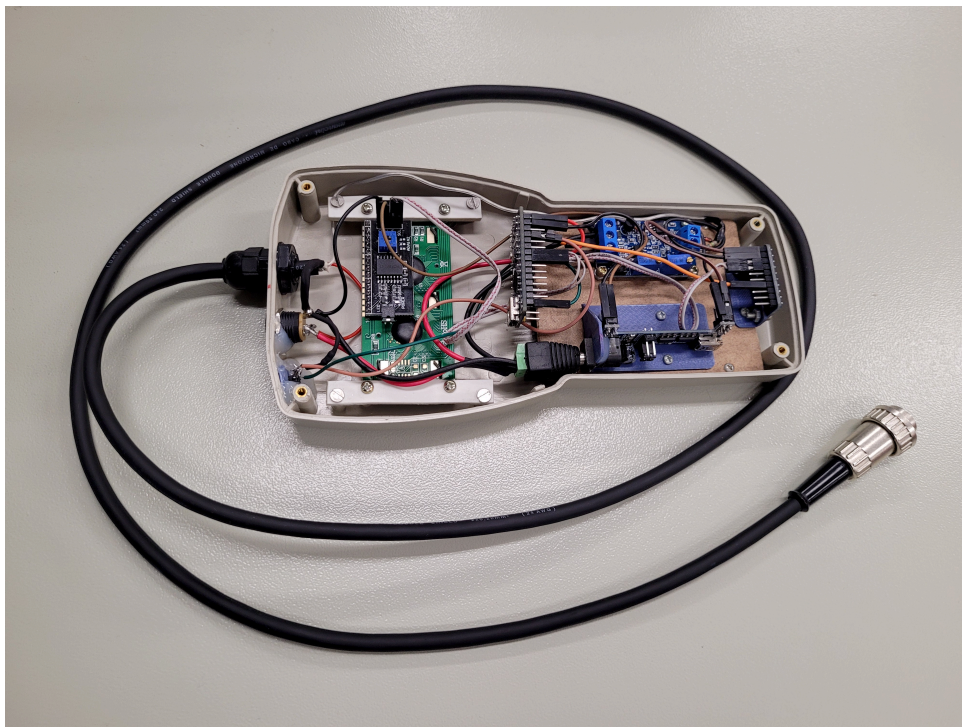
Fonte: Autor

As medidas obtidas para correntes inferiores a 4 mA demonstram uma leve atenuação na correspondência com a tensão, o que era esperado. No entanto, estes pontos não serão reproduzidos na prática, com exceção do valor nulo correspondente ao sensor desconectado. Neste caso, o valor de tensão de saída foi de $-0,0259$ V, que apesar de ser negativo, se encontra dentro da faixa de tolerância do HW-685 ($-0,3$ V).

4.3 MONTAGEM DO DISPOSITIVO DE MEDIÇÃO

Seguindo o projeto apresentado no capítulo 3, cada componente, módulo e placa foi instalado conforme os diagramas esquemáticos das Figuras 3.8 e 3.9. A parte interna do dispositivo montado pode ser observada pela Figura 4.5. Uma vez que o *hardware* está pronto, resta apenas a implementação do *software* para carregar o firmware no microcontrolador.

Figura 4.5 – Parte interna do dispositivo de medição após a conclusão da montagem.



Fonte: Autor.

4.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO 4

Neste capítulo foram apresentados os procedimentos de implementação do *hardware* do projeto. Após a conclusão desta importante etapa, é dado início à implementação do *software* para a programação do Arduino Nano, que comandará e gerenciará todas as funcionalidades propostas pelo projeto, conforme é apresentado no próximo capítulo.

CAPÍTULO 5

IMPLEMENTAÇÃO DO PROJETO - DESENVOLVIMENTO DO SOFTWARE

Neste capítulo é apresentado o desenvolvimento do software do projeto, bem como é descrito de maneira estrutural o código-fonte do dispositivo amostrador de torque.

5.1 AMBIENTE DE PROGRAMAÇÃO

Para o desenvolvimento do *software* do dispositivo amostrador de torque, foi utilizado como ambiente de programação a ferramenta de desenvolvimento *Visual Studio Code* (VScode). Neste *software* é possível editar, compilar e carregar o código-fonte no microcontrolador Arduino Nano. Conjuntamente, foi empregada uma extensão interna de *software*, a PlatformIO. Essa extensão promove a integração para diversos sistemas embarcados, fazendo com que o processo de desenvolvimento seja mais intuitivo. Informações adicionais sobre o ambiente de programação estão disponíveis no apêndice A, onde é apresentado um guia prático para a utilização das ferramentas de desenvolvimento do projeto.

5.2 DESENVOLVIMENTO DO CÓDIGO-FONTE

O código-fonte do projeto se trata de um *firmware* desenvolvido para o módulo Arduino Nano e está disponibilizado integralmente no apêndice B. A linguagem de programação utilizada foi o C++.

No início do código são incluídas as bibliotecas principais para o funcionamento do sistema, conforme o código-fonte 5.1. São elas:

Arduino.h: biblioteca que implementa o suporte ao *hardware* ATmega328.

Adafruit_ADS1X15.h: biblioteca desenvolvida pelos engenheiros do grupo Adafruit que oferece controle para conversores ADC da *Texas Instruments*, *ADS1015* e *ADS1115*, sendo de 12 e 16 bits de resolução, respectivamente.

LiquidCrystal_I2C.h: biblioteca usada para controlar o *display* LCD Hitachi HD44780U integrado com o conversor serial *I2C NXP PCF8574*.

Código-fonte 5.1 – Inserção das bibliotecas

```
3 #include <Arduino.h>
4 #include <Adafruit_ADS1X15.h>
5 #include <LiquidCrystal_I2C.h>
```

Em seguida, é feita a configuração do conversor analógico digital, definindo a versão empregada no projeto (*ADS1115* de 16 bits), ao qual se atribui o nome de variável "*ads*". Conjuntamente, configura-se também o *display* LCD para que ele responda no endereço 0x27 do *I2C*, cujo nome atribuído é "*lcd*" (vide código-fonte 5.2).

Código-fonte 5.2 – configuração das bibliotecas

```
7 Adafruit_ADS1115 ads;
8 LiquidCrystal_I2C lcd(0x27,16,2);
```

Na sequência, define-se as principais variáveis e seus valores iniciais, conforme mostra o código-fonte 5.3. Respectivamente, tem-se: *buttonPin* como sendo o pino atrelado a posição do botão de calibração (valor é fixo); *pressTime* é um valor inteiro utilizado como contador para a monitoração do botão de calibração; *adcValue*, *voltsAvg_offset* e *voltsAvg* são utilizados para a leitura do conversor ADC; *torqueValue* é a variável referente ao cálculo do torque baseado nas tensões obtidas; *button_status* é uma variável booleana que identifica o clique do botão de calibração como verdadeiro ou não; *voltAvgCorrect* se trata valor corrigido de tensão, após a calibração.

Código-fonte 5.3 – Declaração das variáveis

```
10 const int buttonPin = 13;
11 int pressTime = 3;
12 int16_t adcValue = 0;
13 float voltsAvg = 0;
14 float voltsAvg_offset = 0;
15 float torqueValue;
16 volatile bool button_status = false;
17 float voltAvgCorrect;
```

Um *enum* chamado *DisplayState* é definido e serve para criar um conjunto de valores nomeados, representando estados possíveis de exibição. A função *writeLCD* atualiza o *display* LCD com mensagens baseadas no estado atual (*DisplayState*). Ele utiliza um *switch* para tratar os diferentes estados, configurando o conteúdo do *display* de acordo com as condições associadas a cada estado. O valor do torque exibido no *display*, representa a

média do torque medido e é o estado que fica maior parte do tempo em operação. O estado de *testing* indica no *display* uma contagem regressiva para entrar no modo de calibração. O *adjusting* se trata da etapa da escrita da mensagem de "*Finalizando...*" enquanto é feita a execução da calibração em segundo plano. O *adjusted* representa a escrita que a calibração foi concluída. O código-fonte 5.4 mostra um trecho do programa que trata das configurações do *display*. Conforme mencionado anteriormente, o código completo está disponível no apêndice B.

Código-fonte 5.4 – Definição das funções do *display* LCD

```
19 enum DisplayState {
20 SHOW_TORQUE,
21 SHOW_ADJUSTED,
22 SHOW_ADJUSTING,
23 SHOW_TESTING
24 };
```

A função *readT22* lê valores do *ADS1115* para calcular valores médios de tensão ao longo de um período definido e, dependendo do estado (*T22Mode*), ajusta valores conforme o *offset*. Esses parâmetros são definidos no trecho de código apresentado no código-fonte 5.5.

Código-fonte 5.5 – Definição da função de leitura do sinal do *ADS1115*

```
26 enum T22Mode {
27 SET_TORQUE,
28 SET_OFFSET
29 };
```

Uma vez finalizada a programação do sistema de medição e de exibição no *display* LCD, inicia-se a programação da parte operacional do dispositivo. A função *setup* é responsável pela inicialização do sistema no ambiente do Arduino. Ela é declarada no código conforme mostra o trecho do código-fonte 5.6. configura-se o pino do botão de calibração como uma entrada, viabilizando a mudança de estado lógico para o modo de calibração. Em seguida, o *display* é ligado, ascendendo a luz de fundo e exibindo uma mensagem de "boas vindas". Em seguida, o ADC é inicializado e configurado para ter ganho unitário e, portanto, operar na faixa de 0 V a 4 V, conforme detalhado no Capítulo 3.

Código-fonte 5.6 – Configuração de *setup*

```
112 void setup(void)
113 {
114 pinMode(buttonPin, INPUT);
```

```
115
116 lcd.init();
117 lcd.backlight();
118 lcd.setCursor(3,0); lcd.print("Hello T22!");
119 delay(500);
120
121 ads.setGain(GAIN_ONE);
122 if (!ads.begin()) {Serial.println("Failed to initialize ADS."); while (1);}
123 }
```

Por fim, é inserida uma rotina de *loop*, que implementa o comportamento contínuo do sistema. Ela utiliza a entrada do botão para alternar entre modos de operação, executando ações relacionadas às funções de processamento de leituras feitas com o transdutor e exibição de torque no LCD ou das funções relacionadas à calibragem de *offset* e escrita desse processo no *display*, conforme mostra o código-fonte 5.7.

Código-fonte 5.7 – Configuração da rotina de *loop*

```
131 void loop(void)
132 {
133   button_status=digitalRead(buttonPin);
134   if (button_status)
135   {
136     pressTime--;
137     writeLCD(SHOW_TESTING);
138     delay(1000);
139     if (pressTime == 0) {
140       writeLCD(SHOW_ADJUSTING);
141       readT22(SET_OFFSET);
142       writeLCD(SHOW_ADJUSTED);
143       pressTime=3;
144     }
145   } else {
146     pressTime=3;
147     readT22(SET_TORQUE);
148     writeLCD(SHOW_TORQUE);
149   }
150 }
```


5.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO 5

Neste capítulo foi apresentado o estágio de desenvolvimento do *software* do dispositivo amostrador de torque desenvolvido neste trabalho de conclusão de curso. O código-fonte foi apresentado de forma estrutural e resumida. O código completo está disponível no Apêndice B, que pode ser copiado para o ambiente de programação para ser replicado conforme for necessário. Findadas as etapas de desenvolvimento, a implementação do dispositivo está concluída e o amostrador de torque pode ser testado em campo. No próximo capítulo, são apresentados os ensaios de validação do projeto implementado.

CAPÍTULO 6

ENSAIOS DE VALIDAÇÃO DO DISPOSITIVO

Neste capítulo são apresentados os ensaios de validação do dispositivo desenvolvido neste trabalho de conclusão de curso.

6.1 ESTRATÉGIA ADOTADA PARA A VALIDAÇÃO DO TRABALHO

Para que se possa avaliar a precisão e confiabilidade do dispositivo amostrador de torque desenvolvido, idealmente, seria necessário obter parâmetros de comparação por meio de outro dispositivo de referência. Não havendo esta possibilidade, faz-se necessário a elaboração de uma estratégia alternativa.

O transdutor T22 se encontra instalado na bancada de ensaios de máquinas elétricas e está posicionado no eixo que interliga a máquina primária com as outras duas máquinas (motor de indução e gerador síncrono). Portanto, com o torque medido, para uma dada velocidade, é possível calcular o valor da potência mecânica transferida neste ponto de acoplamento. Se este fluxo de potência for mensurável, é possível comparar os resultados para a validação do dispositivo.

Deste modo, assumindo um cenário de testes em que o motor de indução está desenergizado e o gerador síncrono opera alimentando uma carga elétrica qualquer, a potência transferida pelo eixo no ponto de medição corresponde à carga do gerador, somada com as perdas energéticas associadas. Tais perdas podem ser resumidas em: perdas no núcleo do gerador; perdas por efeito Joule nos enrolamentos do gerador; e perdas por atrito e ventilação, associadas aos mancais e ventiladores, tanto do gerador quanto do motor de indução, pois ambos compartilham o mesmo eixo.

As perdas no núcleo estão associadas à tensão e à frequência de operação da máquina síncrona. Para um ensaio no qual a velocidade de rotação é fixa e a tensão terminal é sempre ajustada para o seu valor nominal, pode-se dizer que as perdas no núcleo não variam significativamente para diferentes pontos de carga do gerador. Do mesmo modo, as perdas por atrito e ventilação estão relacionadas com a velocidade de rotação e, portanto, também não devem variar.

Por outro lado, as perdas por efeito Joule ocorrem em decorrência da passagem de corrente elétrica nos enrolamentos de armadura do gerador. Assim, elas variam conforme a carga e podem ser quantificadas pela equação 6.1, na qual P_{joule} corresponde às perdas por efeito joule, R_a é a resistência de armadura do gerador, e I_a é a corrente de armadura que flui pela carga.

$$P_{joule} = 3.R_a.I_a^2 \quad (6.1)$$

Dadas as condições apresentadas, é possível realizar uma série de ensaios de validação por meio da medição da potência elétrica de saída do gerador e da estimativa das perdas por efeito Joule conforme as medidas corrente de armadura para cada ponto de carga. O valor de R_a , é medido ao longo do mesmo ensaio e as demais perdas são consideradas constantes para todos os pontos de operação.

6.2 ENSAIOS DE VALIDAÇÃO

Os ensaios realizados para a validação do dispositivo amostrador de torque seguem o seguinte roteiro de ações:

1. Acionar a máquina primária e atingir a velocidade nominal (1800 rpm);
2. Colocar carga nominal no gerador e aguardar até que se atinja o regime térmico permanente ;
3. Desenergizar o sistema;
4. Medir rapidamente a resistência de armadura;
5. Calibrar o dispositivo amostrador de torque para a condição de torque nulo;
6. Acionar a máquina primária até a velocidade nominal sem estabelecer o campo do gerador e realizar a medição de torque para a obtenção das perdas rotacionais;
7. Estabelecer o campo até atingir a tensão nominal de armadura (220 V) e medir o torque para obter as perdas no núcleo;
8. Iniciar as medições com progressão gradual da carga no gerador, mantendo sempre a tensão terminal em 220 V;
9. Testar condições iniciais e finalizar o ensaio.

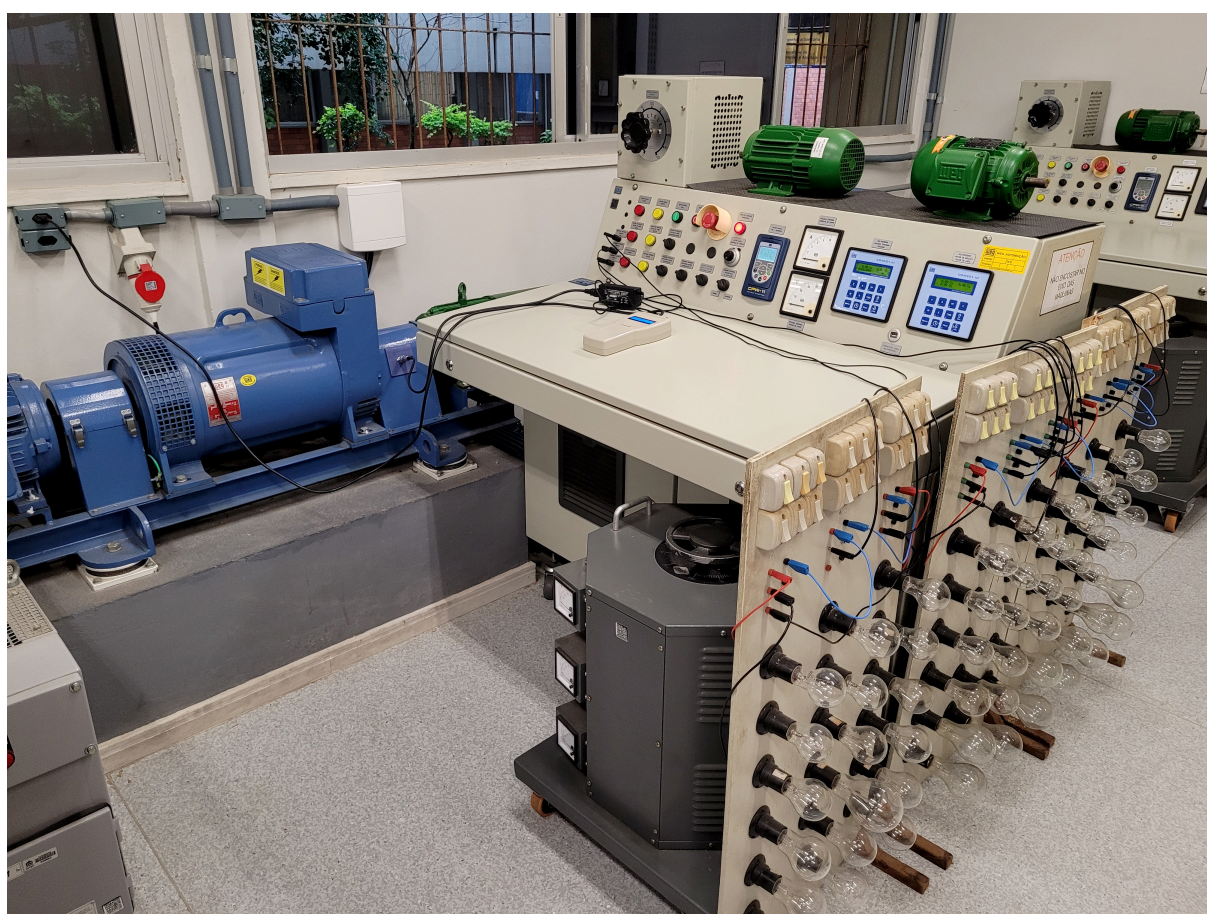
A relação entre o torque (T) e a potência mecânica (P_m) é dada pela equação 6.2, onde ($v_{rad/s}$) é a velocidade de rotação em radianos por segundo.

$$P_m = T.v_{rad/s} \quad (6.2)$$

6.2.1 Setup da bancada

Para a realização dos ensaios de validação, o dispositivo amostrador de torque desenvolvido neste trabalho é conectado ao transdutor T22 e os terminais de armadura do gerador são ligados à bancas de cargas resistivas. As cargas são compostas por conjuntos de lâmpadas incandescentes ligadas em delta (equilibradamente), com interruptores que permitem o chaveamento para se obter uma progressão gradual do carregamento do gerador. A Figura 6.1 mostra o *setup* da bancada de ensaios.

Figura 6.1 – *Setup* da bancada para a realização dos ensaios de validação.



Fonte: Autor.

Os multimedidores da bancada são responsáveis por fornecer os valores de tensão e corrente de linha, bem como a potência trifásica fornecida pelo gerador às cargas de lâmpadas. A resistência de armadura é medida com a máquina desenergizada e em circuito aberto, por meio do multímetro Yokogawa TY720.

6.2.2 Resultados obtidos

Seguindo o roteiro de ensaios, a bancada foi energizada e os conjuntos de lâmpadas foram chaveados para que o gerador operasse em condição nominal. A máquina permanece neste estado por um intervalo de tempo suficientemente longo até atingir o seu regime térmico permanente, ou seja, até que não haja mais variação considerável de sua temperatura interna. Idealmente, para que se possa garantir esta condição, seria necessário dispor de medidores de temperatura internos no gerador. Na impossibilidade de realizar este procedimento, definiu-se um intervalo de tempo de uma hora, com o gerador operando sob carga nominal, para então iniciar os procedimentos de medição.

Transcorrido este intervalo de tempo de aquecimento, a bancada foi desenergizada e, então, a resistência de armadura foi medida, obtendo-se: $R_a = 0,23 \Omega$. Em seguida, o amostrador de torque é calibrado para a posição de torque nulo e o sistema é reenergizado novamente. Com a máquina primária girando a 1800 rpm e o campo do gerador ainda sem excitação, é feita a medição de torque pelo amostrador, obtendo-se o valor de 1,98 N.m. Estas etapas do ensaio devem ser realizadas rapidamente para que não ocorra variação significativa da temperatura interna do gerador.

A partir deste momento, o campo da máquina síncrona é excitado até que o gerador atinja a tensão nominal de 220 V. Assim, iniciam-se os ensaios para cada ponto de carga até a condição nominal. A tensão terminal é ajustada para se manter constante independentemente da carga por meio do reostato de campo da bancada. Os resultados obtidos são apresentados na Tabela 6.1. Por fim, são testadas as condições iniciais: a máquina é desligada e o amostrador de torque deve medir "0 N.m" novamente, garantindo que não houve alteração no funcionamento do dispositivo.

Tabela 6.1 – Resultados dos ensaios.

Carga (W)	Tensão (V)	Corrente (A)	Torque (N.m)
0	220,1	0	2,94
595	220,4	1,56	6,12
1187	220,1	3,13	9,28
1800	220,2	4,71	12,5
2398	220,7	6,28	15,7
2980	219,9	7,82	18,8
3614	221,6	9,4	22,34
4200	221,3	10,47	25,5
4749	219,7	12,47	28,5
5100	221,1	13,3	30,47

Fonte: Autor.

Conhecendo a velocidade de rotação, o valor do torque medido antes da excitação de

campo permite obter o valor das perdas rotacionais. Em seguida, com o estabelecimento do campo e ainda sem carga (primeiro ponto da Tabela 6.1) a medição de torque corresponde ao somatório das perdas rotacionais com as perdas no núcleo. Realizando os cálculos, obtém-se:

$$\begin{aligned} \text{Perdas rotacionais } P_{rot} &= 373,2 \text{ W} \\ \text{Perdas no núcleo } P_{nu} &= 180,9 \text{ W} \end{aligned}$$

As medidas estimadas de perdas rotacionais e de núcleo são valores considerados constantes para todos os pontos de carga dos ensaios. Assim, é possível realizar uma análise comparativa, conforme a progressão do carregamento, entre a potência mecânica medida por meio do amostrador de torque e o somatório das perdas com a potência elétrica de saída do gerador. Os resultados são apresentados na Tabela 6.2.

A primeira coluna da tabela mostra os valores tomados como referência para cada ponto de carga, obtidos diretamente da medição de potência dos multimedidores da bancada. Na segunda coluna são exibidos os valores da potência mecânica de entrada do sistema (P_{in}), que correspondem à potência mecânica total calculada por meio das medidas do amostrador de torque pela equação 6.2. Como as perdas por efeito Joule (P_{joule}) variam para cada ponto de carga, elas são calculadas pela equação 6.1 para cada valor de corrente de armadura e os resultados são apresentados na terceira coluna da tabela. Por fim, na quarta coluna são os obtidos os valores para comparação com a referência, que representam a potência de saída do gerador (P_{out}), correspondente à potência de entrada subtraída de todas as perdas associadas, conforme a equação 6.3.

$$P_{out} = P_{in} - P_{rot} - P_{nu} - P_{joule} \quad (6.3)$$

Tabela 6.2 – Análise de potência dos ensaios realizados.

Carga (W)	P_{in} (W)	P_{joule} (W)	P_{out} (W)	var (%)
595	1153,6	1,7	598	0,50
1187	1749,2	6,8	1188	0,08
1800	2356,2	15,3	1787	-0,72
2398	2959,4	27,2	2378	-0,83
2980	3543,7	42,2	2947	-1,11
3614	4211,0	61,0	3596	-0,50
4200	4806,6	75,6	4177	-0,54
4749	5372,1	107,3	4711	-0,80
5100	5743,5	122,1	5067	-0,65

Fonte: Autor.

A última coluna da Tabela 6.2 mostra a relação percentual da diferença entre P_{out} e os valores de referência. Observa-se que os resultados se mostraram promissores,

apresentando variações relativamente baixas para todos os pontos de cargas (por volta de 1,0%).

6.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO 6

Os resultados dos ensaios de validação foram conclusivos e indicam que o dispositivo amostrador de torque desenvolvido neste trabalho de conclusão de curso é capaz de realizar medidas precisas e confiáveis. A análise comparativa entre a potência elétrica medida pelo multimedidor da bancada e a potência de saída do gerador, calculada por meio das medições do amostrador de torque e da segregação das perdas do sistema, demonstrou convergência de resultados. A variação percentual entre os valores, para cada ponto de carga, foi da ordem de 1,0%, que é um valor consideravelmente baixo, se levado em consideração as margens de incertezas do transdutor T22 e do multimedidor da bancada. No próximo capítulo é apresentada a conclusão geral do trabalho e são feitas sugestões para trabalhos futuros.

CAPÍTULO 7

CONCLUSÃO E SUGESTÕES PARA TRABALHOS FUTUROS

O presente trabalho de conclusão de curso teve como objetivo o desenvolvimento e a implementação de um dispositivo de baixo custo para medição dos sinais dos transdutores de torque das bancadas didáticas do Laboratório de Máquinas e Acionamentos Elétricos (LABMAQ) da Universidade Federal de Santa Catarina (UFSC). Dada a ausência de um sistema comercial financeiramente acessível para essa finalidade, foi desenvolvida uma solução alternativa utilizando componentes eletrônicos amplamente disponíveis, garantindo precisão e confiabilidade nas medições.

A concepção do amostrador de torque envolveu a escolha criteriosa de seus componentes, levando em consideração as características do transdutor HBM T22. Para garantir a compatibilidade dos sinais e preservar a integridade das medições, foi necessário o uso de um módulo conversor de corrente para tensão, o que possibilitou o correto condicionamento do sinal antes da digitalização. Além disso, a adoção do ADC ADS1115, com resolução de 16 bits, proporcionou uma conversão de sinal mais precisa em comparação ao conversor interno do microcontrolador ATmega328, presente no Arduino Nano.

Os testes de validação foram conduzidos a partir da comparação entre a potência elétrica medida pelo multimedidor da bancada e a potência mecânica calculada com base nos dados do amostrador de torque. Os resultados demonstraram uma boa correlação entre os valores obtidos, com variações da ordem de 1,0%, reforçando a precisão e confiabilidade do dispositivo desenvolvido.

Além do desempenho técnico satisfatório, destaca-se o baixo custo da solução proposta. O dispositivo foi construído com um investimento de aproximadamente \$45,00 dólares americanos, em insumos. Tal custo é significativamente baixo se for comparado aos sistemas comerciais de medição de torque. Assim, este projeto se mostra uma alternativa economicamente viável para laboratórios acadêmicos e aplicações didáticas.

Diante dos resultados obtidos, algumas sugestões podem ser exploradas em trabalhos futuros para aprimorar e expandir a aplicabilidade do amostrador de torque:

1. Estudo de propagação de incertezas - Uma análise mais aprofundada das incertezas associadas ao processo de medição permitirá quantificar cientificamente as margens de erro do dispositivo, proporcionando uma avaliação mais rigorosa da confiabilidade dos resultados.
2. Aplicação em ensaios de motores de indução - A utilização do amostrador em testes específicos com motores de indução permitirá a avaliação de parâmetros de rendimento e a segregação detalhada das perdas, contribuindo para um entendimento mais abrangente do comportamento dessas máquinas elétricas.
3. Desenvolvimento de uma nova versão do amostrador - Uma possível evolução do sistema inclui a implementação de uma interface de comunicação com um computador, permitindo a amostragem contínua dos sinais do transdutor ao longo do tempo. Isso viabilizaria a análise de torques transitórios e dinâmicos, possibilitando estudos mais detalhados sobre o comportamento das máquinas elétricas em diferentes regimes de operação.

Com isso, este trabalho representa um avanço significativo no monitoramento e análise de torque em bancadas didáticas de ensino de máquinas elétricas do LABMAQ, fornecendo uma base sólida para estudos futuros e contribuindo para a evolução das metodologias experimentais na área de conversão eletromecânica de energia.

REFERÊNCIAS

- [1] ELY, F. N. Análise do sistema de medição trifásica dos ensaios de máquinas rotativas elétricas do LABMAQ tendo como objetivo a quantificação das incertezas de medição. Universidade Federal de Santa Catarina, p. 73, 2018. Citado na página 25.
- [2] INC., H. B. . K. *HBM T22 Torque transducer - Datasheet*. [S.l.], 2021. Citado na página 27.
- [3] INC., H. B. . K. *HBM T22 Torque transducer - Mounting Instructions*. [S.l.], 2021. Citado na página 28.
- [4] MAKERHERO Placa Arduino Nano - V3. 2024. Disponível em: <<https://www.makerhero.com/produto/placa-nano-v3-0-cabo-usb-para-arduino/>>. Acessado em: 30 de agosto de 2024. Citado na página 31.
- [5] MAKERHERO Display LCD 16x2 - Backlight Azul. 2024. Disponível em: <<https://www.makerhero.com/produto/display-lcd-16x2-backlight-azul/>>. Acessado em: 30 de agosto de 2024. Citado na página 32.
- [6] CONVERSOR Analógico Digital 4 Canais - ADS1115. 2024. Disponível em: <<https://www.makerhero.com/produto/conversor-analogico-digital-4-canais-ads1115/>>. Acessado em: 30 de agosto de 2024. Citado na página 33.
- [7] MAKERHERO Módulo conversor de corrente em tensão - WH685. 2024. Disponível em: <<https://www.makerhero.com/produto/modulo-conversor-de-corrente-em-tensao/>>. Acessado em: 30 de agosto de 2024. Citado na página 34.
- [8] MAKERHERO Fonte ajustável - MB102. 2024. Disponível em: <<https://www.makerhero.com/produto/fonte-ajustavel-para-protoboard-mb102/>>. Acessado em: 30 de agosto de 2024. Citado na página 35.
- [9] KAUR, D. *An Introduction to System Software*. [S.l.]: Alpha Science, 2021. ISBN 1783325380; 9781783325382. Citado na página 38.
- [10] BANIK, V. Z. S. *System Firmware: An Essential Guide to Open Source and Embedded Solutions*. [S.l.]: Apress, 2022. ISBN 1484279387; 9781484279380. Citado na página 39.
- [11] GETTING Started with VS Code and PlatformIO IDE for ESP32 and ESP8266 (Windows, Mac OS X, Linux Ubuntu). 2025. Disponível em: <<https://randomnerdtutorials.com/vs-code-platformio-ide-esp32-esp8266-arduino/>>. Acessado em: 22 de janeiro de 2025. Citado na página 85.

Apêndices

APÊNDICE A

GUIA PRÁTICO PARA A UTILIZAÇÃO DAS FERRAMENTAS DE DESENVOLVIMENTO DO PROJETO

Neste apêndice é apresentado um guia prático de utilização das ferramentas de desenvolvimento empregadas neste trabalho de conclusão de curso. O objetivo deste documento é auxiliar outros alunos em projetos similares ou de continuidade do dispositivo amostrador de torque.

A.1 GUIA PRÁTICO PARA A UTILIZAÇÃO DAS FERRAMENTAS DE DESENVOLVIMENTO DO PROJETO

O Amostrador de Torque tem seu software desenvolvido por completo em um ambiente baseado no software gratuito da *Microsoft*, *Visual Studio Code (VS Code)*. O *VS Code* é uma das IDEs mais utilizadas no mundo e, além de ferramentas de edição com suporte a diversas linguagens de programação, ele oferece uma biblioteca com uma gama de poderosas extensões. As extensões do *VS Code* abrangem suporte a todas as áreas da engenharia de software, desde sistemas embarcados, desenvolvimento web, aplicativos, inteligência artificial, robótica ou processamento de dados.

Uma dessas extensões é a *PlatformIO*, utilizada neste projeto como a alternativa para desenvolvimento de software embarcado. Tradicionalmente é usado o ambiente *Arduino IDE*, pois ele é capaz de compilar e carregar *firmwares* em placas Arduino sem nenhum tipo de configuração adicional. Mas o *PlatformIO* foi escolhido pois além das funções citadas ele também proporciona funcionalidades avançadas como controle de versão integrado, ferramentas de depuração (*debug*), e suporte a múltiplas placas e bibliotecas.

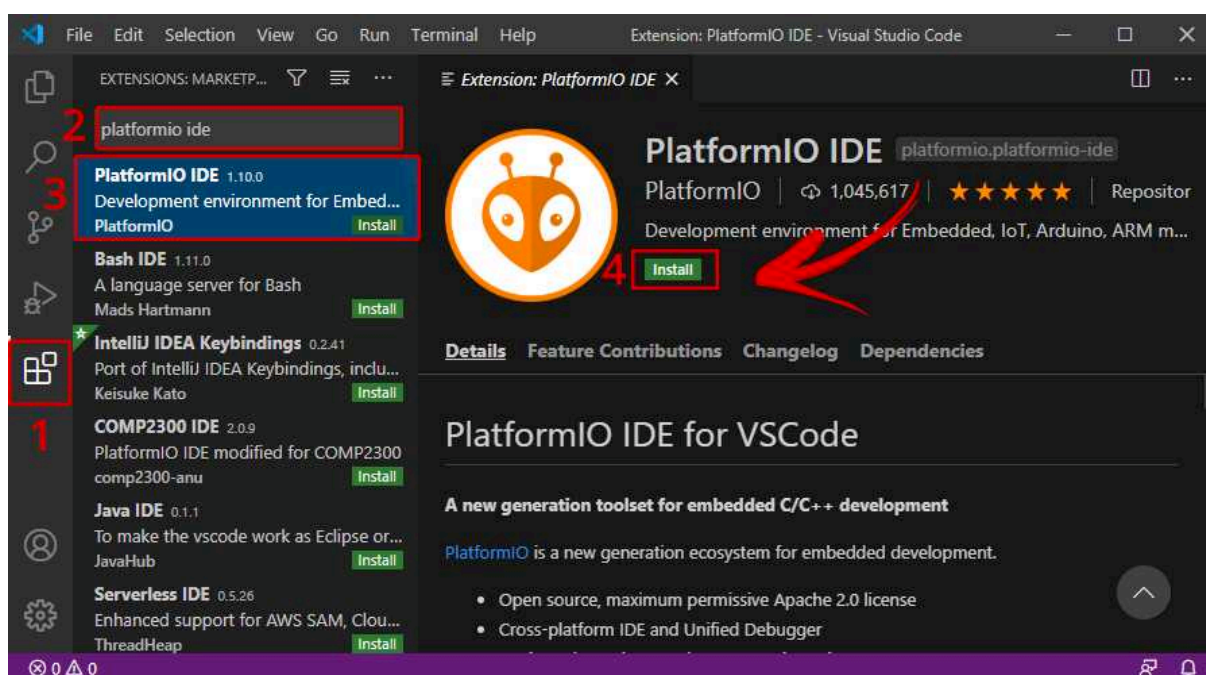
O conteúdo deste apêndice busca guiar outros estudantes e desenvolveres no processo de configuração do ambiente de desenvolvimento no sistema operacional *Windows*. Desde a instalação dos softwares necessários até a criação do projeto, usando como exemplo o projeto do amostrador de torque desenvolvido neste TCC. Vale ressaltar que outros sistemas operacionais também são suportados. Se for de interesse, pode-se consultar o tutorial apensado no anexo A.

A.2 INSTALAÇÃO DO VS CODE E PERIFÉRICOS

Inicialmente, é necessário fazer a instalação do VS Code. No link <<https://code.visualstudio.com/>> selecione a opção “*Download for Windows*” para baixar o instalador com a versão mais recente. Siga o *Wizard* de instalação da forma que preferir. Ao finalizar a instalação, é necessário instalar a linguagem de *scriptização* de alto nível, *Python*, de preferência uma versão 3.5 ou maior. Caso já tenha o *Python* instalado, verifique a versão, e o atualize se necessário. O instalador do *Python* pode ser encontrado no link <<https://www.python.org/downloads/windows/>>. Adicione o *Python* ao *PATH* durante o *Wizard* de instalação. A *PlatformIO* é baseado em *scripts Python* e, por isso, sua instalação é necessária.

Por fim, é necessário instalar a extensão do *PlatformIO* dentro do *VS Code*. O passo a passo é ilustrado pela Figura A.1. Abra a IDE e, na aba lateral, abra a interface de extensões, procure por *PlatformIO IDE*. Clique na opção de *Install* e espere. Após instalado, certifique-se que a extensão está ativa. É recomendado reiniciar o *VS Code* para concluir.

Figura A.1 – Tela de instalação da plataforma - Em ordem numérica: (1) Janela de extensões; (2) Barra de pesquisa de extensão com o nome da plataforma; (3) Lista de resultados da pesquisa; (4) Opção de instalação na página de extensão *PlatformIO IDE*.



Fonte: *PlatformIO IDE* - Adaptado pelo autor.

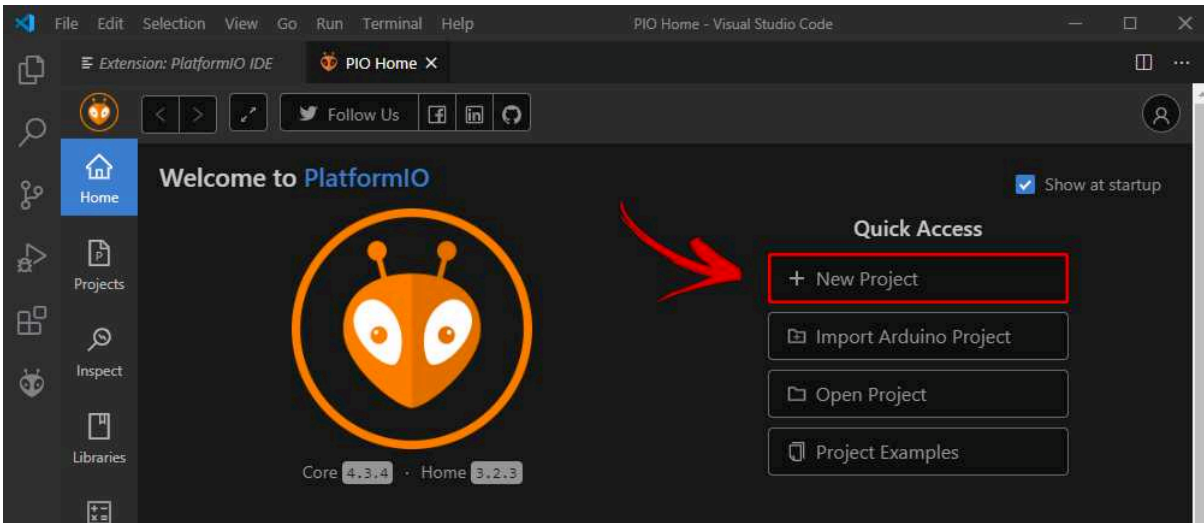
A.3 CRIAÇÃO DO PROJETO

Tomando como exemplo o projeto do amostrador de torque, é necessário criar um projeto para a plataforma *Arduino Nano*. O *Arduino Nano* é a placa do microcontrolador responsável pelo processamento geral e demanda de um *firmware* específico. O restante dos componentes internos do amostrador de torque agem como escravos enquanto este microcontrolador é o mestre.

Abra o *VS Code* e, em seguida, clique no ícone do *PlatformIO* para abrir sua *Home*. Clique na opção *New Project*, conforme mostra a Figura A.2. Abrindo um *wizard* do projeto, é necessário escolher *Name*, *Board* e *Framework*. Dê o nome que desejar e em "*Board*" procure pela placa *Arduino Nano ATmega328*. Em *Framework* selecione a única opção para essa placa, *Arduino*. Dessa forma é usado o *Arduino Core* para as implementações.

Após alguns segundos, uma série de arquivos são adicionados ao *Explorer* do projeto, criando um esqueleto geral para o projeto. A *PlatformIO* precisa deles para compilar e gerar os binários do *firmware* corretamente, portanto, não é recomendado alterar esses arquivos. Vide Figura A.3.

Figura A.2 – Home da PlatformIO com opção de “Novo Projeto”, em vermelho.



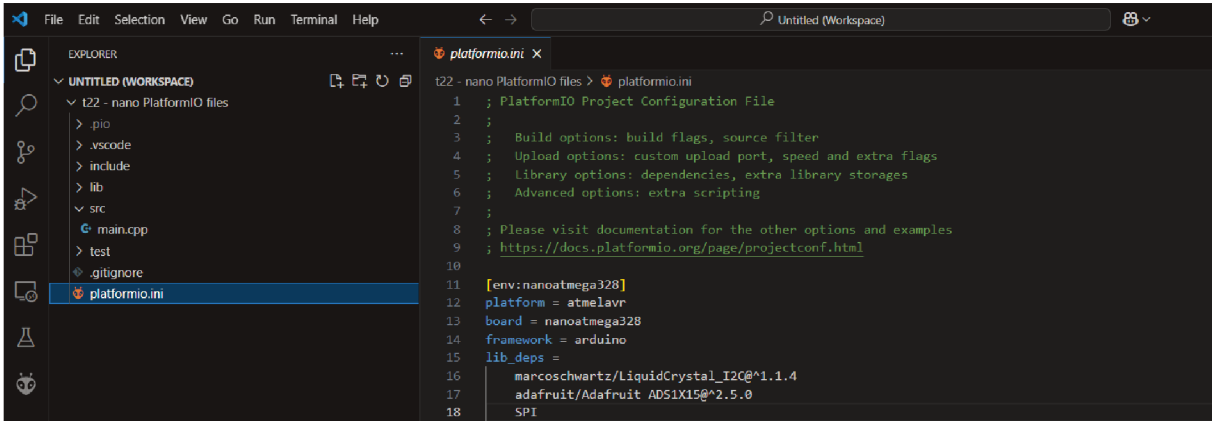
Fonte: PlatformIO IDE - Adaptado pelo autor.

Figura A.3 – Árvore de arquivos criados pela PlatformIO no Explorer do VSCode.



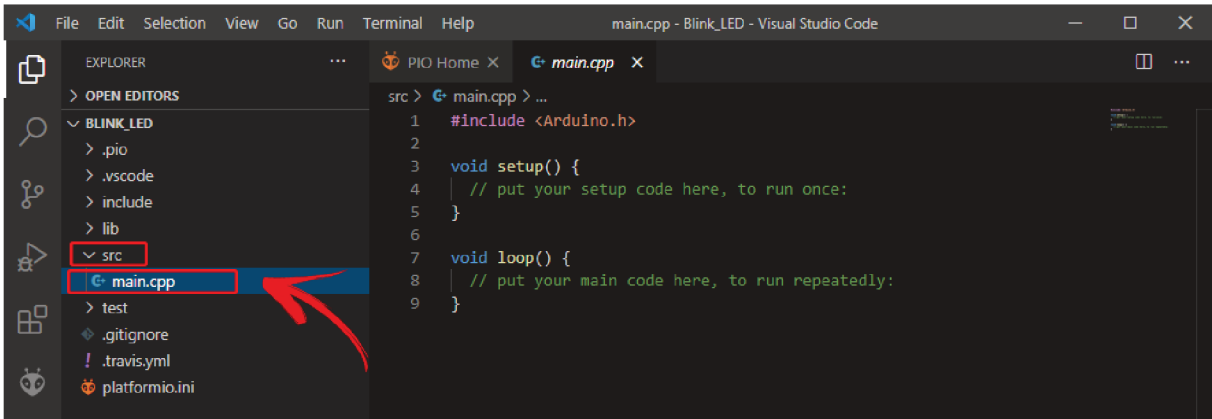
Fonte: PlatformIO IDE - Adaptado pelo autor.

Os arquivos importantes para o projeto são principalmente o *platformio.ini* e o *main.cpp*, encontrados dentro de uma pasta de trabalho *src*. O primeiro deve mostrar as configurações base, como as que foram selecionadas na criação do projeto (vide Figura A.4). O segundo, contém o código fonte principal, que por padrão, é estruturado no Arduino com as funções *setup()* e *loop()* (vide Figura A.5). Vale destacar também que projetos que fazem uso desse *framework* iniciam com a inclusão da biblioteca *Arduino.h*.

Figura A.4 – Arquivo *platformio.ini* com as configurações iniciais do projeto do Amostrador de Torque.

```
File Edit Selection View Go Run Terminal Help
platformio.ini
t22 - nano PlatformIO files > platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:nanoatmega328]
12 platform = atmelavr
13 board = nanoatmega328
14 framework = arduino
15 lib_deps =
16   marcoschwartz/LiquidCrystal_I2C@^1.1.4
17   adafruit/Adafruit_ADS1X15@^2.5.0
18   SPI
```

Fonte: PlatformIO IDE - Adaptado pelo autor.

Figura A.5 – Arquivo *main* genérico criado por *default* pela PlatformIO.

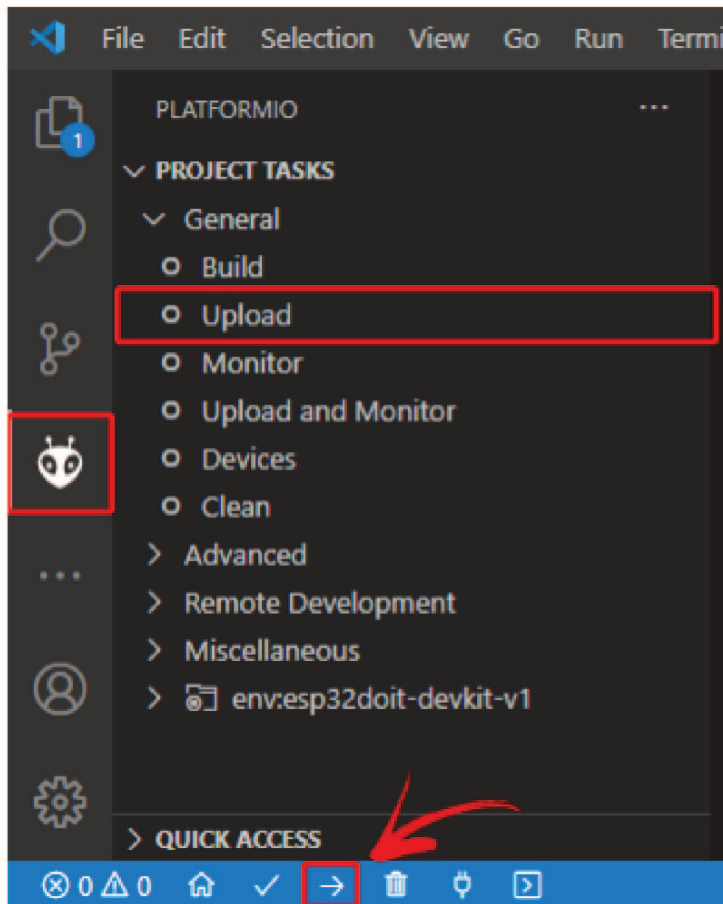
```
File Edit Selection View Go Run Terminal Help
main.cpp - Blink_LED - Visual Studio Code
EXPLOLER
OPEN EDITORS
BLINK_LED
.pio
.vscode
include
lib
src
main.cpp
test
.gitignore
.travis.yml
platformio.ini
src > main.cpp > ...
1 #include <Arduino.h>
2
3 void setup() {
4   // put your setup code here, to run once:
5 }
6
7 void loop() {
8   // put your main code here, to run repeatedly:
9 }
```

Fonte: PlatformIO IDE - Adaptado pelo autor.

A.4 CONEXÃO COM O ARDUINO PARA INSERÇÃO DO FIRMWARE DESENVOLVIDO

Para fazer *upload* do *firmware* do amostrador de torque no microcontrolador Arduino Nano, adicione os arquivos *main.cpp* e *platformio.ini*, salve-os e conecte um cabo micro USB entre o Arduino e a máquina do ambiente de programação, clique na *Task* ou na flecha na barra de tarefas para iniciar o *Upload*, conforme mostra a Figura A.6.

Figura A.6 – Serviços referentes ao projeto disponibilizados pela extensão *PlatformIO* com as opções de *upload* sinalizadas.



Fonte: *PlatformIO* IDE - Adaptado pelo autor.

O *VS Code* deve então abrir um terminal de *log* com mensagens referentes ao processo do *PlatformIO*, iniciando com a adição de dependências, configuração e compilação. Ao final do *build*, os arquivos *firmware.hex* e *firmware.elf* são criados. O primeiro corresponde ao código de máquina em hexadecimal pronto para ser gravado na memória flash do microcontrolador e o segundo é um arquivo completo com instruções de máquina também e informações de depuração. Com ele é estimado o espaço de memória RAM e Flash que o *firmware* deve ocupar (vide Figura A.7).

Figura A.7 – *Logs* gerados pela *PlatformIO* ao executar o *Build* com sucesso.

```
Building in release mode
Checking size .pio\build\nanoatmega328\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [=== ] 34.7% (used 710 bytes from 2048 bytes)
Flash: [=== ] 31.1% (used 9564 bytes from 30720 bytes)
===== [SUCCESS] Took 1.38 seconds =====
```

Fonte: *PlatformIO* IDE - Adaptado pelo autor.

Em seguida, no processo de *upload*, a plataforma inicia a procura por uma porta USB com o Arduino Nano para fazer o *upload* do arquivo *.hex* contendo os binários. Ao encontrar, a escrita em memória flash é feita. E uma mensagem de sucesso é imprimida no terminal, conforme mostra a Figura A.8.

Figura A.8 – Logs gerados pela *PlatformIO* ao executar o *Upload* com sucesso.

```
Wrote 216752 bytes (109861 compressed) at 0x00010000 in 2.7 seconds (effective 631.3 kbit/
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 30.85 seconds =====
```

Fonte: *PlatformIO* IDE - Adaptado pelo autor.

Finalizado o processo de *upload* do *firmware*, desconecte o cabo USB, conecte o dispositivo na alimentação normal e o microcontrolador está pronto para ser usado. Para alterar o projeto ou adaptá-lo, é interessante saber que a *PlatformIO* oferece meios de adicionar outras bibliotecas pela opção *Libraries* dentro de sua *Home*, esse gerenciador de pacotes oferece códigos reutilizáveis compatíveis com a plataforma e que adicionam funcionalidades tanto de *hardware* como de *software*. Alguns exemplos de *hardware* são: sensores; módulos de comunicação rádio frequência; outros microcontroladores; motores e sistemas de controle PID. Em se tratando de *software*, tem-se: protocolos de comunicação serial e paralelo; algoritmos padrões; gerenciamento de tarefas com base em tempo; implementação de timers para ações assíncronas; requisições HTTP e outras implementações de redes, entre outros exemplos.

Em caso de dúvidas, é recomendado ler o artigo apresentado no Anexo A, que foi utilizado como base neste trabalho. A *PlatformIO* também dispõe de uma grande comunidade de usuários online e muita documentação para suporte.

APÊNDICE B



CÓDIGO-FONTE DO DISPOSITIVO

Neste apêndice é disponibilizado o código fonte do amostrador de torque.

```
1 //CÓDIGO-FONTE DO DISPOSITIVO
2
3 #include <Arduino.h>
4 #include <Adafruit_ADS1X15.h>
5 #include <LiquidCrystal_I2C.h>
6
7 Adafruit_ADS1115 ads; /*ADC 16-bit version */
8 LiquidCrystal_I2C lcd(0x27,20,4);
9
10 const int buttonPin = 13;
11 int pressTime = 3;
12 int16_t adcValue = 0;
13 float voltsAvg = 0;
14 float voltsAvg_offset = 0;
15 float torqueValue;
16 volatile bool button_status = false;
17 float voltAvgCorrect;
18
19 enum DisplayState {
20 SHOW_TORQUE,
21 SHOW_ADJUSTED,
22 SHOW_ADJUSTING,
23 SHOW_TESTING
24 };
25
26 enum T22Mode {
27 SET_TORQUE,
28 SET_OFFSET
29 };
30
31 void writeLCD(DisplayState state)
32 {
33 lcd.clear();
34 switch (state) {
35 case SHOW_TORQUE:
36 if (voltAvgCorrect < 0.3){
37 lcd.setCursor(0,0); lcd.print("T22 desconectado");
38 }
39 else{
40
```



```
41 if (voltAvgCorrect >= 1.99 && voltAvgCorrect <= 2.01)
42 {
43   lcd.setCursor(0,0); lcd.print("Torque:");
44   lcd.setCursor(8,0); lcd.print(abs(torqueValue));
45   lcd.setCursor(14,0); lcd.print("Nm");
46   lcd.setCursor(0,1); lcd.print("----");
47 }
48 else
49 {
50   if(voltAvgCorrect >= 2) //2.15V = valor torque zero 10mA
51   {
52     lcd.setCursor(0,0); lcd.print("Torque:");
53     lcd.setCursor(8,0); lcd.print(abs(torqueValue));
54     lcd.setCursor(14,0); lcd.print("Nm");
55     lcd.setCursor(0,1); lcd.print("Rot: (+)");
56   }
57   else
58   {
59     lcd.setCursor(0,0); lcd.print("Torque:");
60     lcd.setCursor(8,0); lcd.print(abs(torqueValue));
61     lcd.setCursor(14,0); lcd.print("Nm");
62     lcd.setCursor(0,1); lcd.print("Rot: (-)");
63   }
64 }
65 }
66 break;
67
68 case SHOW_TESTING:
69   lcd.setCursor(0,0); lcd.print("Calibrando em");
70   lcd.setCursor(5,1); lcd.print(pressTime);
71   lcd.setCursor(7,1); lcd.print("seg");
72   break;
73
74 case SHOW_ADJUSTING:
75   lcd.setCursor(1,0); lcd.print("Finalizando...");
76   break;
77
78 case SHOW_ADJUSTED:
79   lcd.setCursor(3,1); lcd.print("Ajustado!");
80   break;
81 }
```

```
82 }
83
84 void readT22(T22Mode state) //float readT22
85 {
86     const unsigned long samplingPeriod = 100; // Amostragem 100 milisegundos
87     const unsigned long averagingPeriod = 3000; // 3 segundos
88     const int numSamples = averagingPeriod / samplingPeriod;
89
90     float totalVolts = 0;
91     for (int i = 0; i < numSamples; i++) {
92         adcValue = ads.readADC_SingleEnded(0);
93         float volts = ads.computeVolts(adcValue);
94         totalVolts += volts;
95         delay(samplingPeriod);
96     }
97
98     voltsAvg = totalVolts / numSamples;
99
100    switch (state){
101    case SET_TORQUE:
102        voltAvgCorrect = voltsAvg - voltsAvg_offset;
103        torqueValue = (voltAvgCorrect - 2)*31.25;
104        break;
105
106    case SET_OFFSET:
107        voltsAvg_offset = voltsAvg - 2;
108        break;
109    }
110 }
111
112 void setup(void)
113 {
114     //button init
115     pinMode(buttonPin, INPUT);
116
117     //lcd
118     lcd.init(); // inicializa lcd
119     lcd.backlight(); //backlight off
120     lcd.setCursor(3,0); lcd.print("Hello T22!"); // Exibe no display.
121     delay(500);
122 }
```

```
123 //serial debugging
124 Serial.begin(9600);
125 Serial.println("Hello T22!");
126
127 ads.setGain(GAIN_ONE);
128 if (!ads.begin()) {Serial.println("Failed to initialize ADS."); while (1);}
129 }
130
131 void loop(void)
132 {
133   button_status=digitalRead(buttonPin);
134   if (button_status) //teste
135   {
136     pressTime--;
137     writeLCD(SHOW_TESTING);
138     delay(1000);
139     if (pressTime == 0) {
140       writeLCD(SHOW_ADJUSTING);
141       Serial.println("Button held for 3 seconds.");
142       readT22(SET_OFFSET);
143       writeLCD(SHOW_ADJUSTED);
144       pressTime=3;
145     }
146   } else {
147     pressTime=3;
148     readT22(SET_TORQUE);
149     Serial.println("-----");
150     Serial.print("AINO: "); Serial.print(adcValue); Serial.print(" ");
151     Serial.print(voltAvgCorrect); Serial.print("V"); Serial.print(" ");
152     Serial.print(torqueValue);Serial.println("Nm");
153     writeLCD(SHOW_TORQUE);
154   }
155 }
```


Anexos

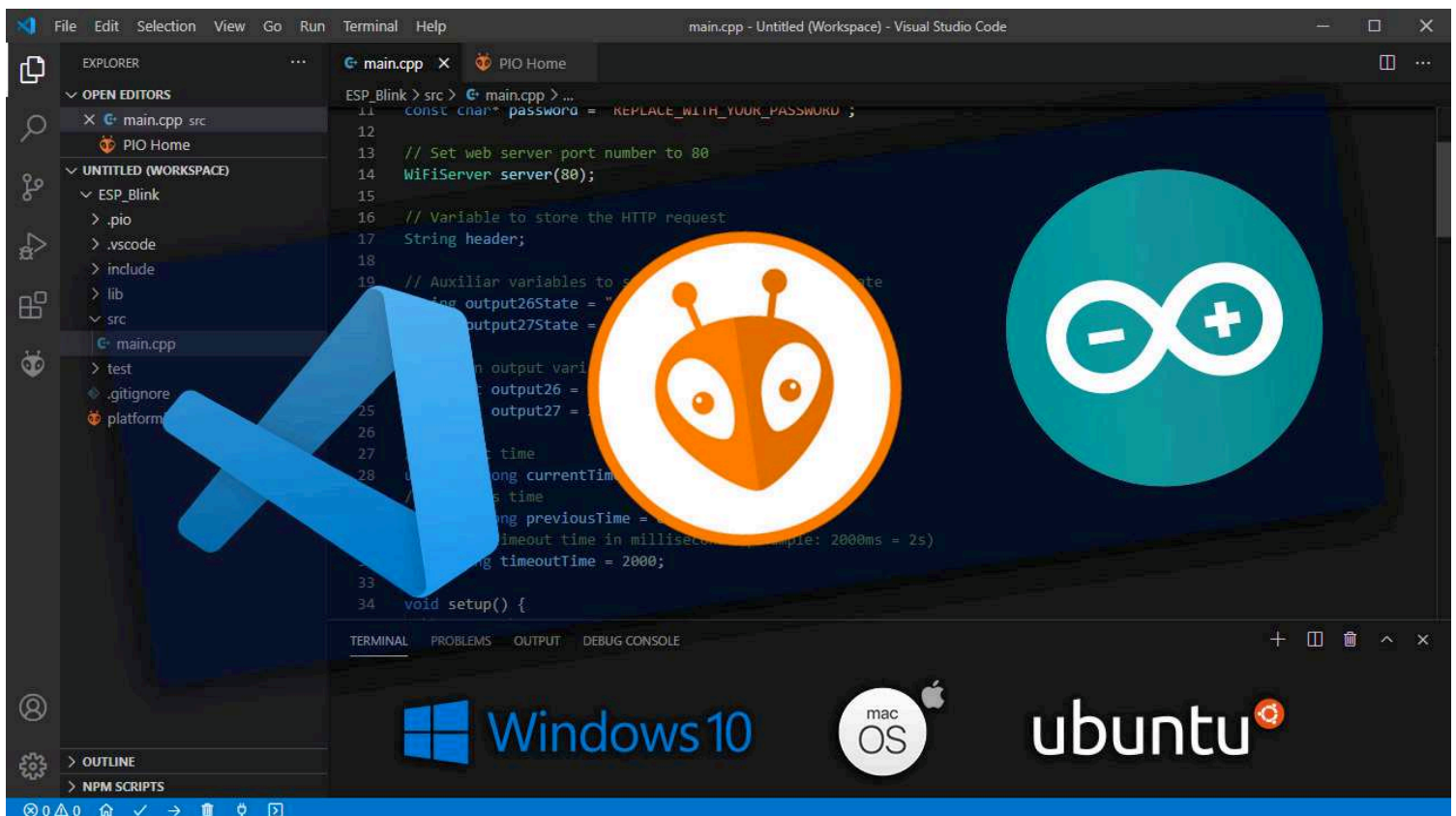
ANEXO A

TUTORIAL DE UTILIZAÇÃO (RANDOM TUTORIALS) - VS CODE E PLATFORMIO IDE

Este anexo contém um tutorial de utilização do *VS Code* e da *PlatformIO* IDE fornecido pelo *website* da Random Tutorials [11].

Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266 (Windows, Mac OS X, Linux Ubuntu)

Learn how to program the ESP32 and ESP8266 NodeMCU boards using VS Code (Microsoft Visual Studio Code) with PlatformIO IDE extension. We cover how to install the software on Windows, Mac OS X or Ubuntu operating systems.



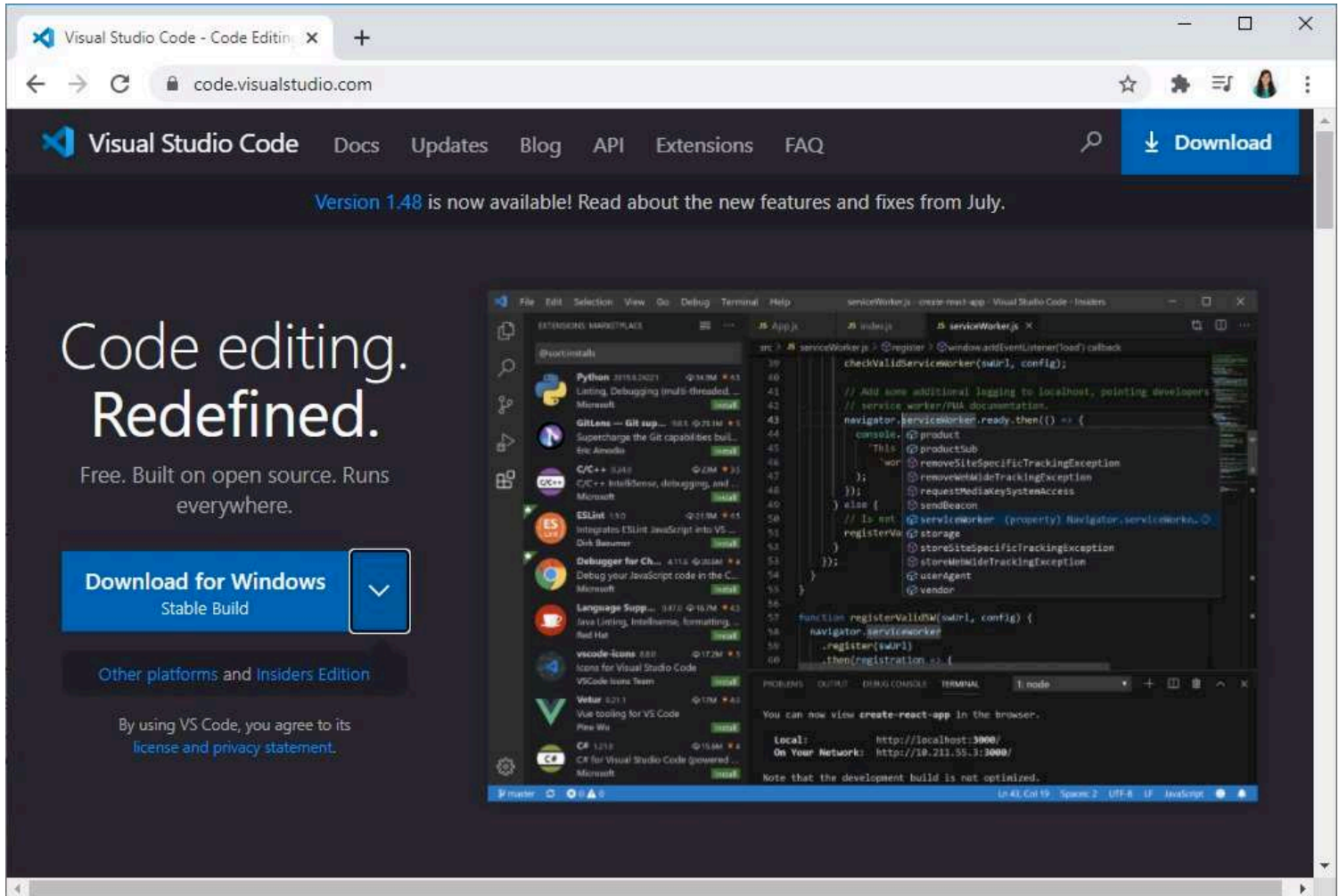
The Arduino IDE works great for small applications. However, for advanced projects with more than 200 lines of code, multiple files, and other advanced features like auto completion and error checking, VS Code with the PlatformIO IDE extension is the best alternative.

In this tutorial, we'll cover the following topics:

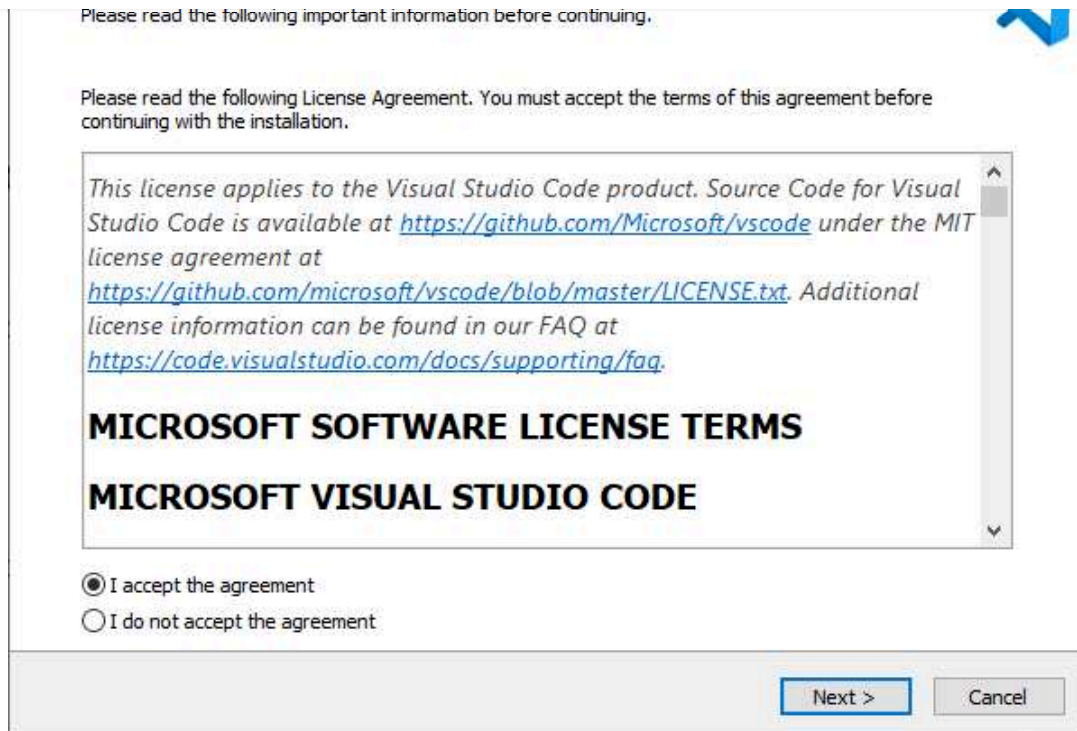
- Installing VS Code (Visual Studio Code):
 - [A\) Windows](#)
 - [B\) Mac OS X](#)
 - [C\) Linux Ubuntu](#)
- [Installing PlatformIO IDE Extension on VS Code](#)
- [Visual Studio Quick Interface Overview](#)
- [PlatformIO IDE Overview](#)
- [Uploading Code using PlatformIO IDE: ESP32/ESP8266](#)
- [Changing the Serial Monitor Baud Rate – PlatformIO IDE](#)

A) Installing VS Code on Windows (Visual Studio Code)

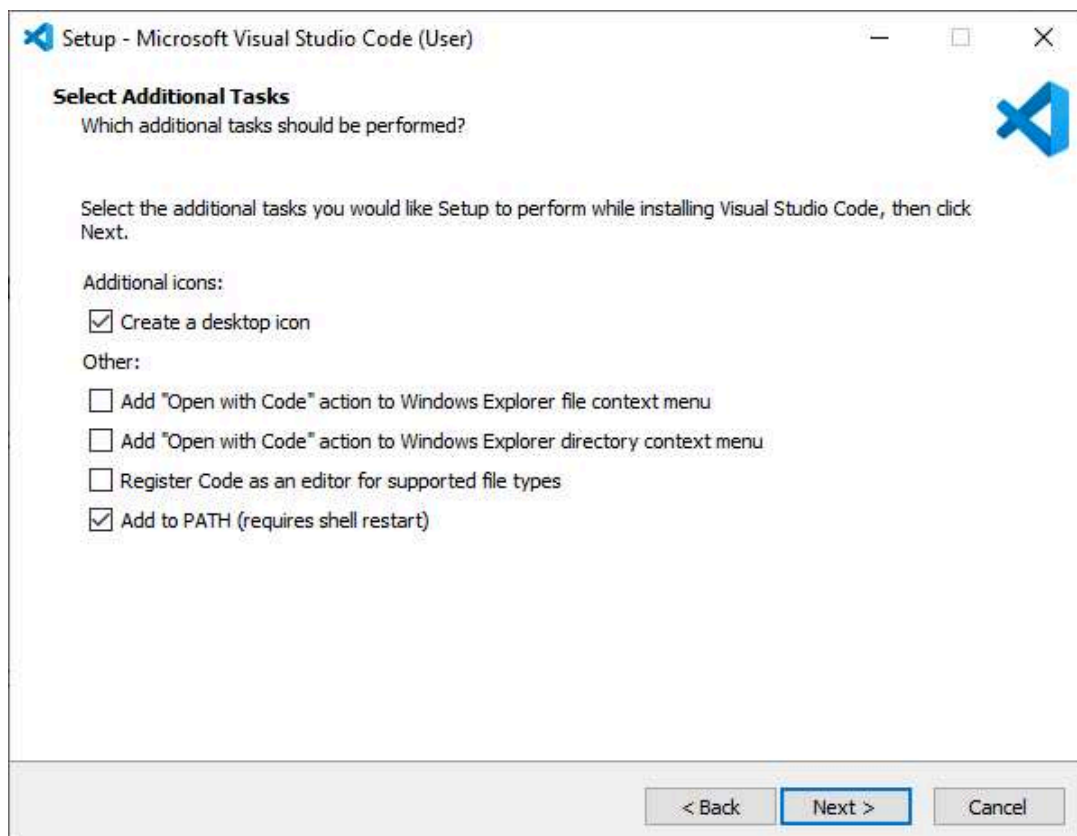
Go to <https://code.visualstudio.com/> and download the stable build for your operating system (Windows).



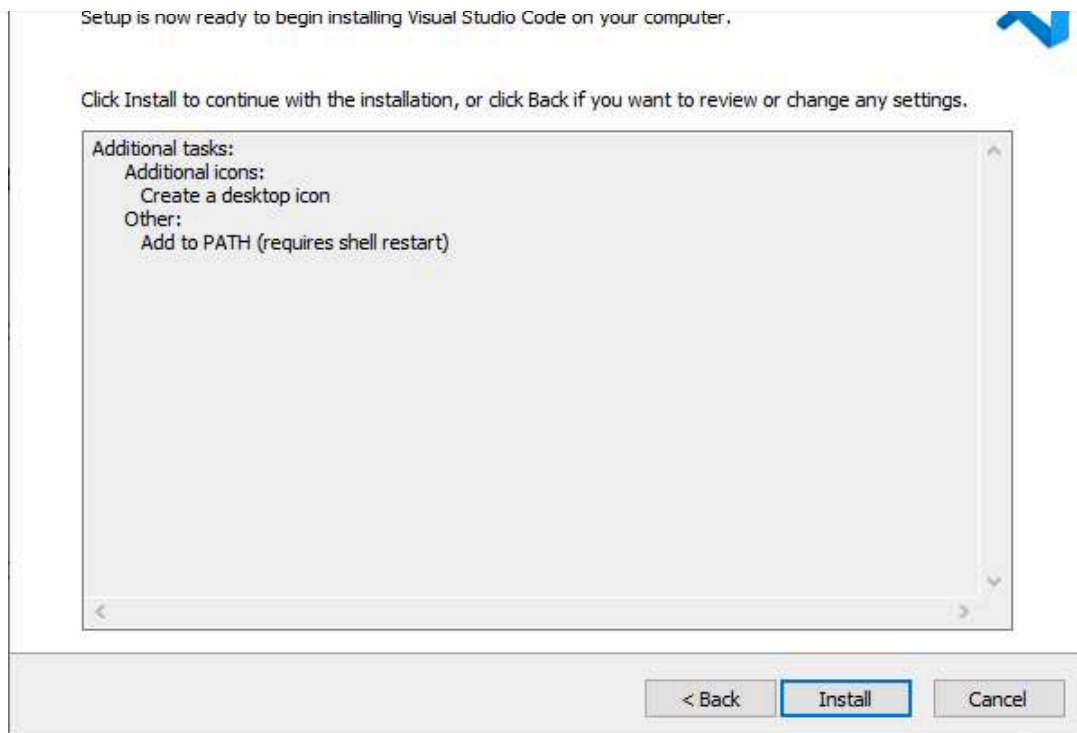
Click on the installation wizard to start the installation and follow all the steps to complete the installation. Accept the agreement and press the **Next** button.



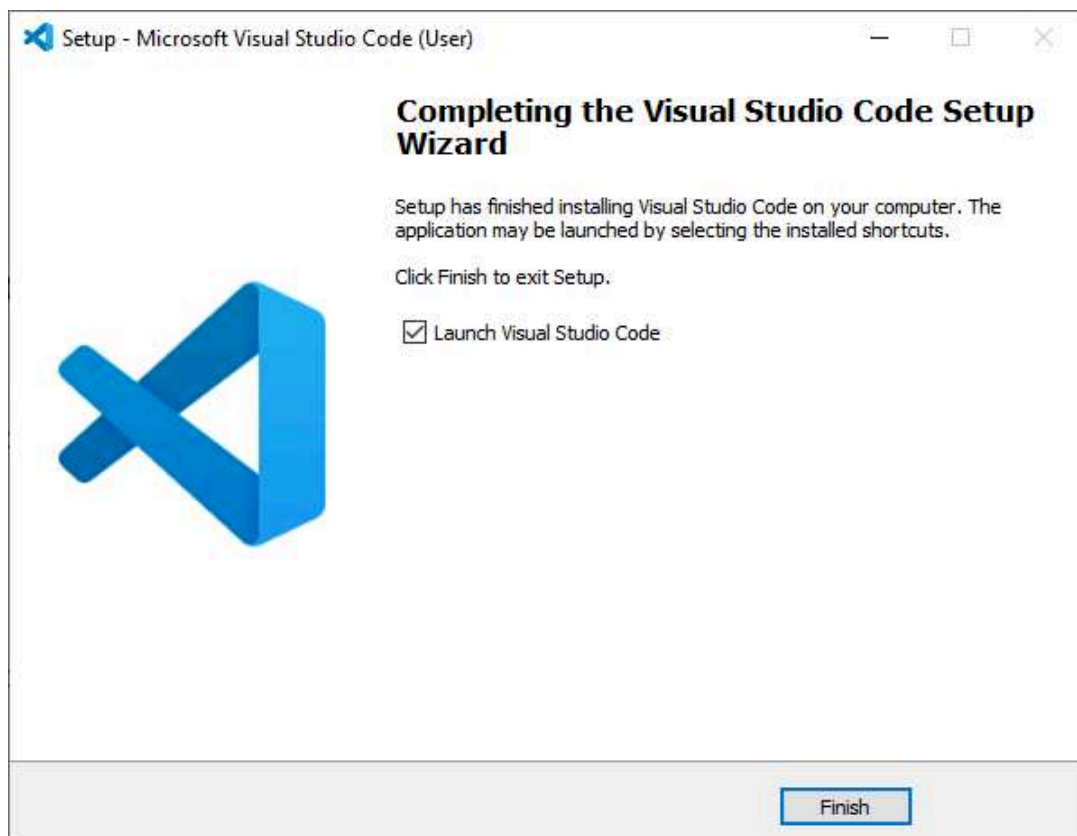
Select the following options and click **Next**.



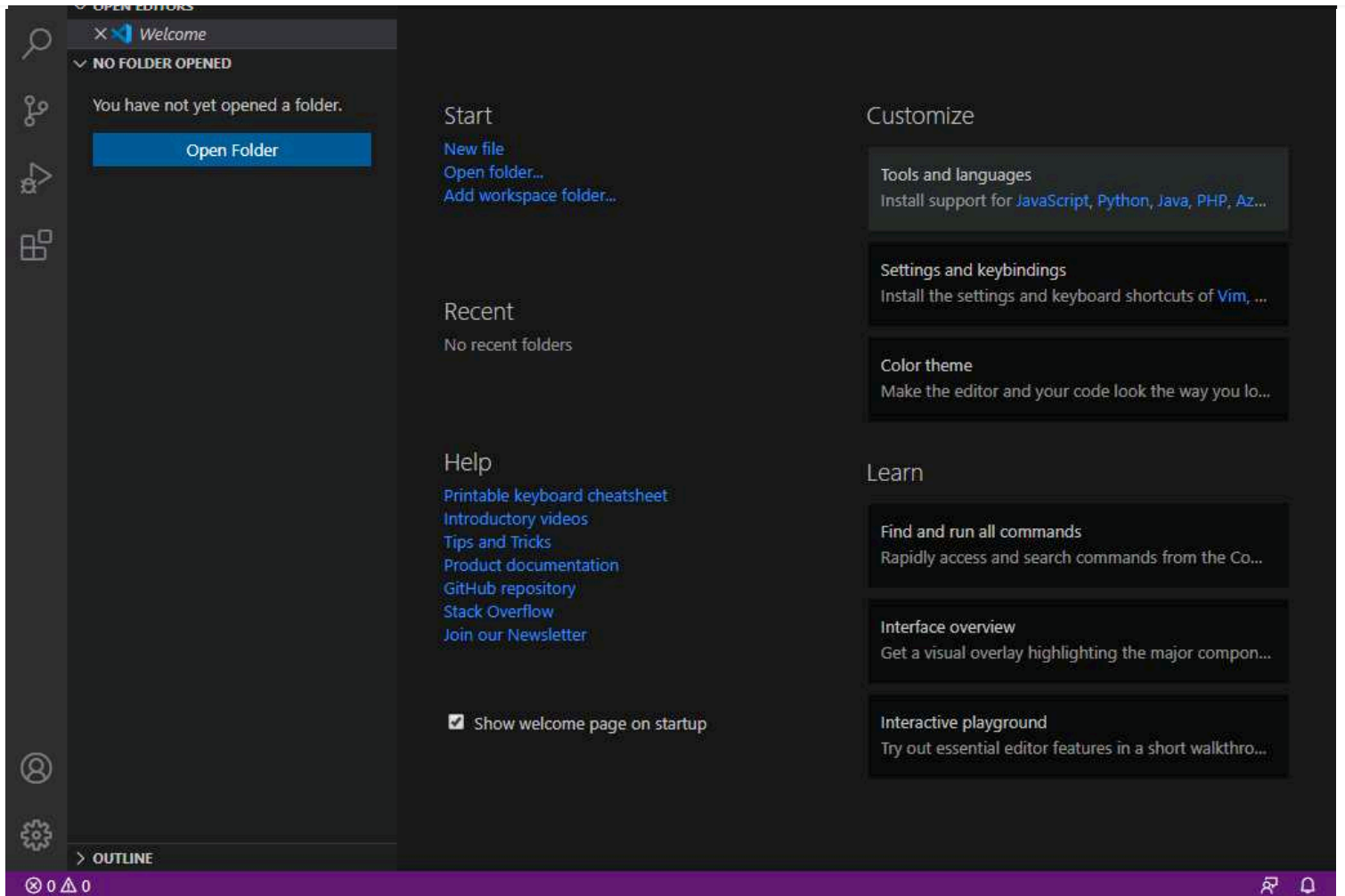
Press the **Install** button.



Finally, click **Finish** to finish the installation.



Open VS Code and you'll be greeted by a Welcome tab with the released notes of the newest version.



That's it. Visual Studio Code was successfully installed.

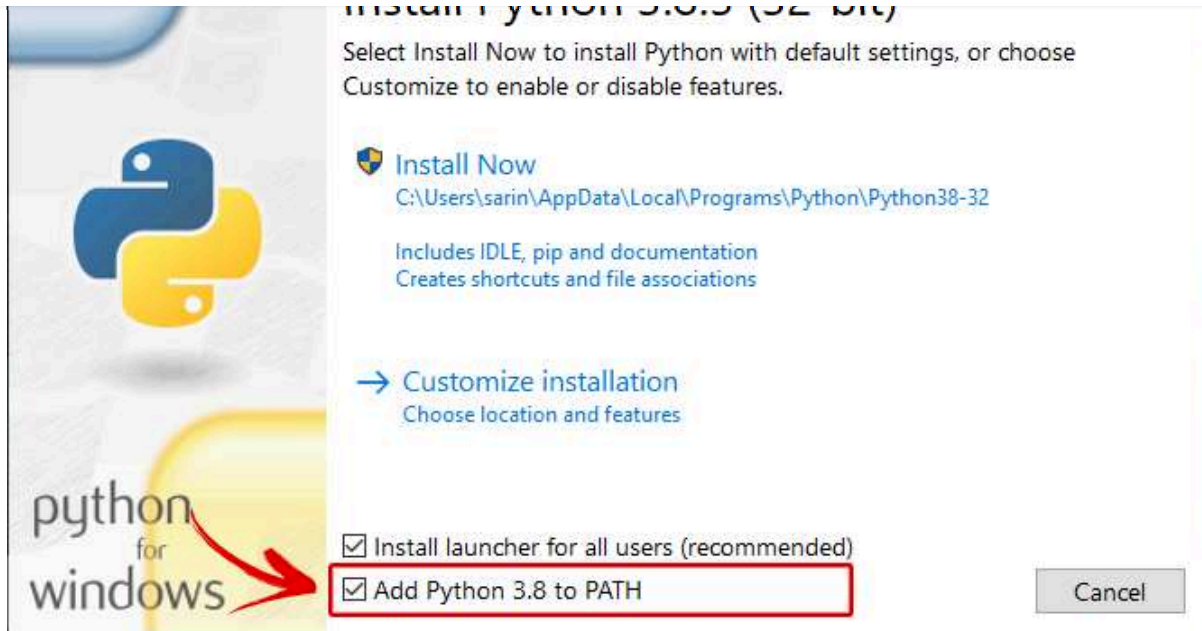
Installing Python on Windows

To program the ESP32 and ESP8266 boards with PlatformIO IDE you need Python 3.5 or higher installed in your computer. We're using Python 3.8.5.

Go to python.org/download and download Python 3.8.5 or a newest version.

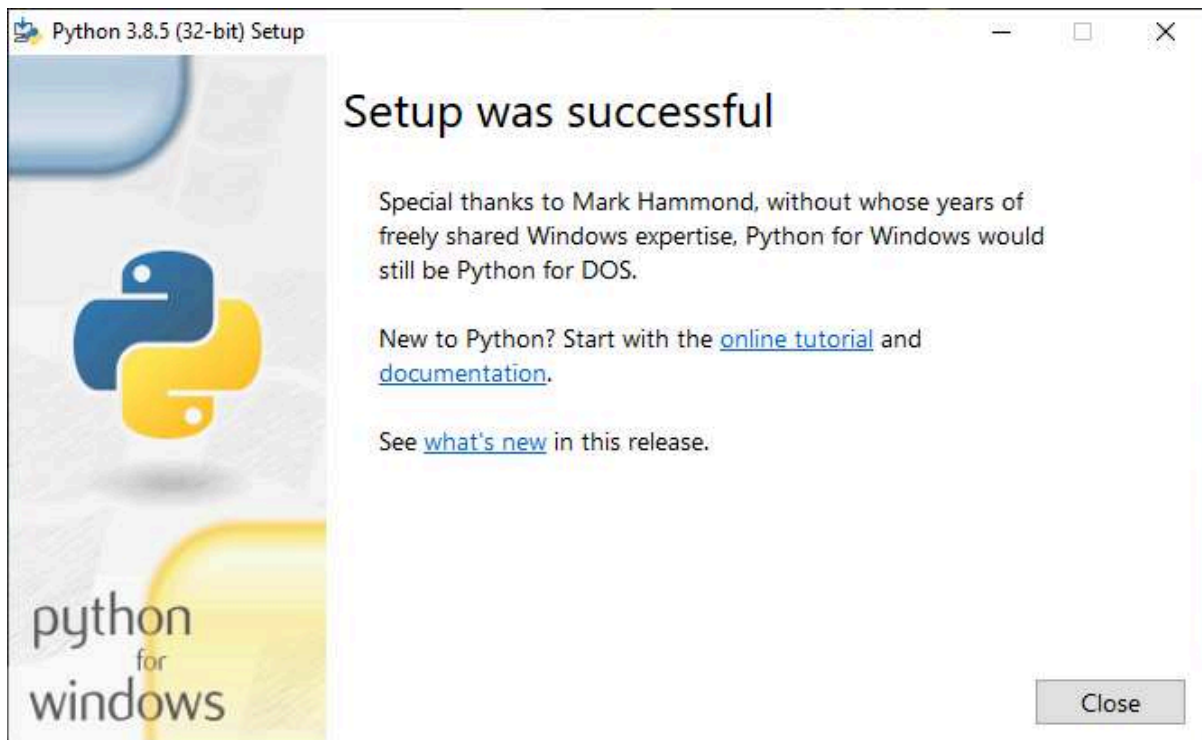
Open the downloaded file to start the Python installation wizard.

The following window shows up.



IMPORTANT: Make sure you check the option **Add Python 3.8 to PATH**. Then, you can click on the **Install Now** button.

When the installation is successful you'll get the following message.

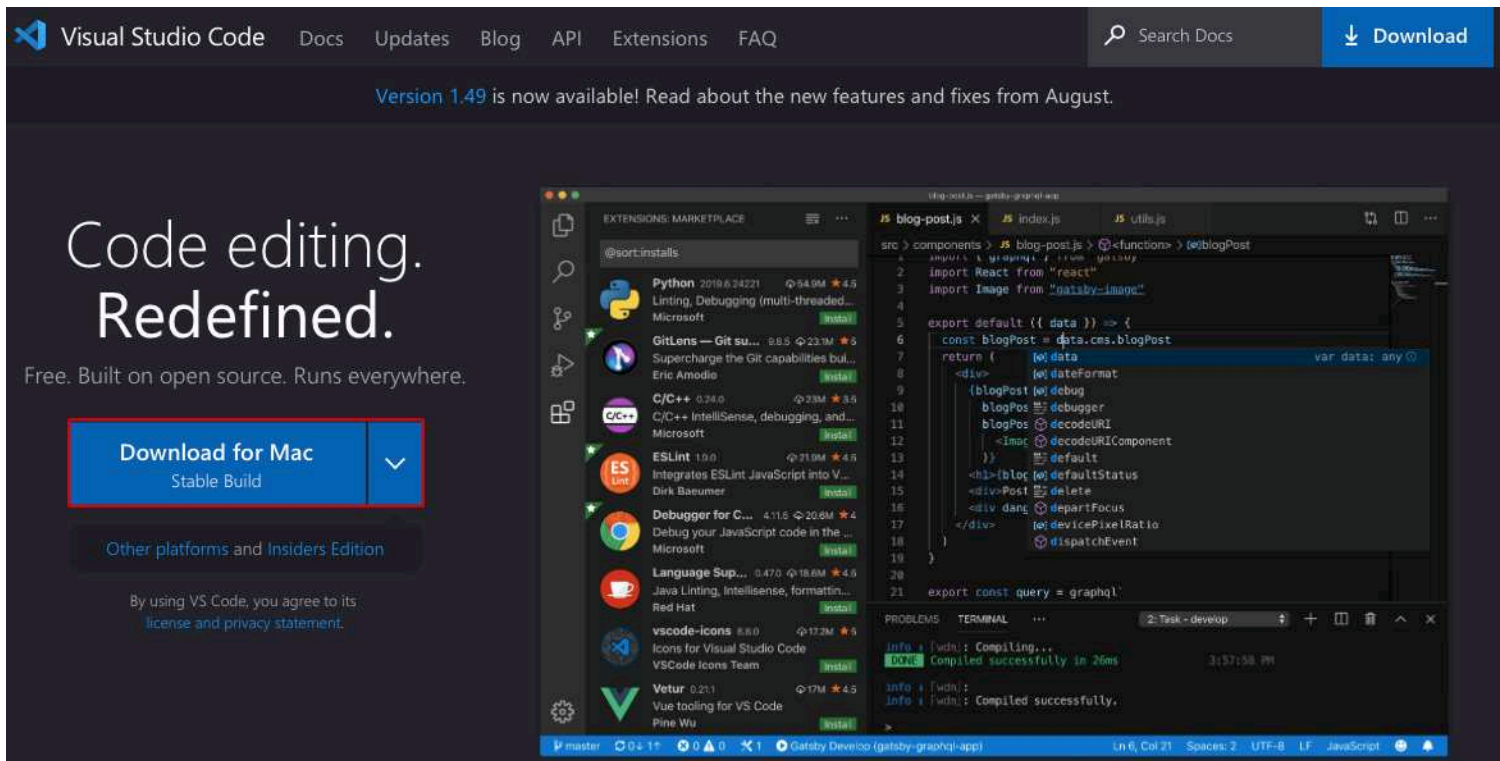


You can click the **Close** button.

Now, [go to this section](#) to install PlatformIO IDE extension.

Code)

Go to <https://code.visualstudio.com/> and download the stable build for your operating system (Mac OS X).



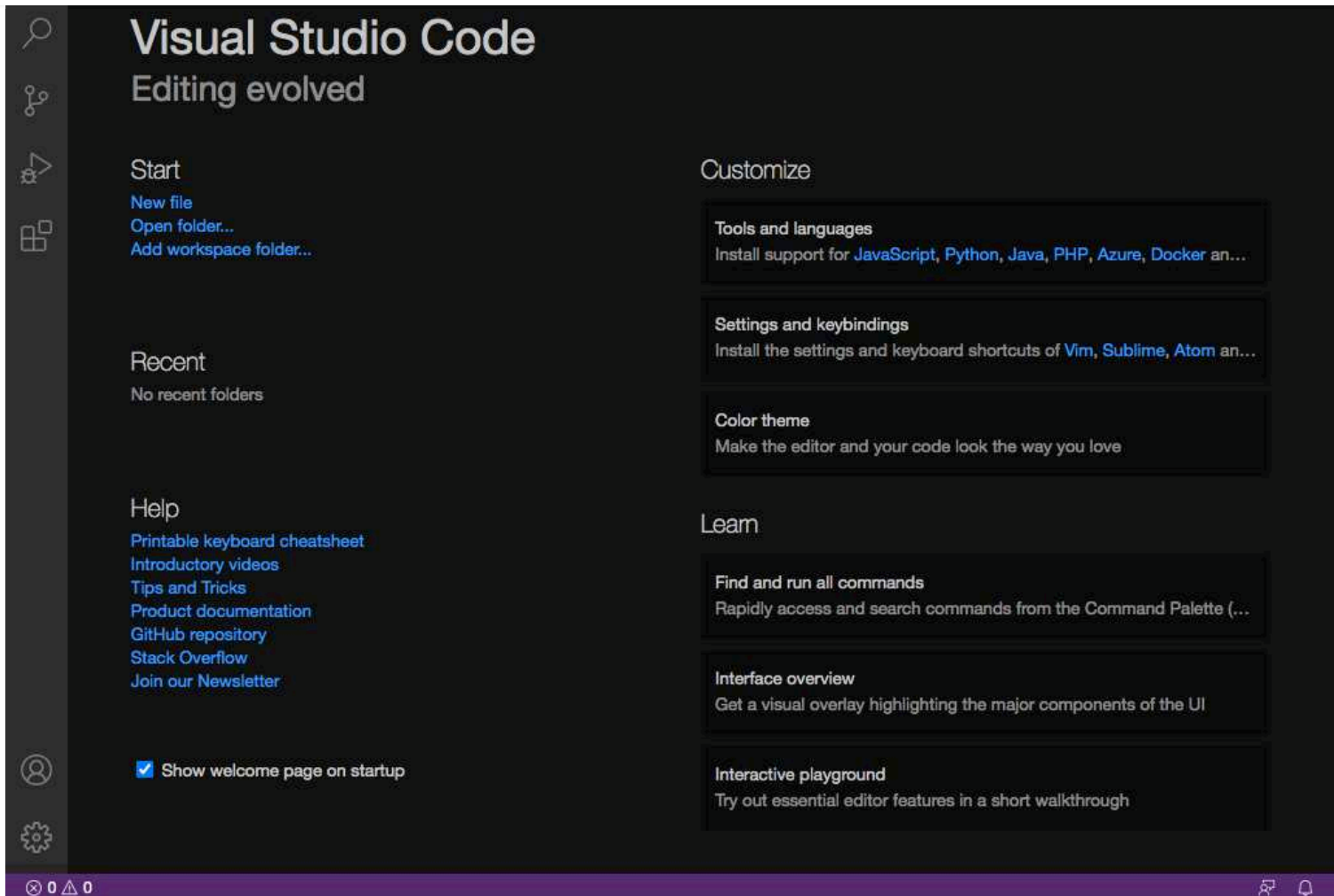
After downloading the Visual Studio Code application file, you'll be prompted with the following message. Press the **“Open”** button.



Or open your Downloads folder and open Visual Studio Code.



After that, you'll be greeted by a Welcome tab with the released notes of the newest version.



That's it. Visual Studio Code was successfully installed.

Installing Python on Mac OS X

To program the ESP32 and ESP8266 boards with PlatformIO IDE you need Python 3.5 or higher installed in your computer. We're using Python 3.8.5.

To install Python I'll be using Homebrew. If you don't have the brew command available, type the next command:

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```



Then, run the brew command to install Python 3.X:

```
Santos — bash — 135x13
==> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
  https://docs.brew.sh/Analytics
No analytics data has been sent yet (or will be during this `install` run).

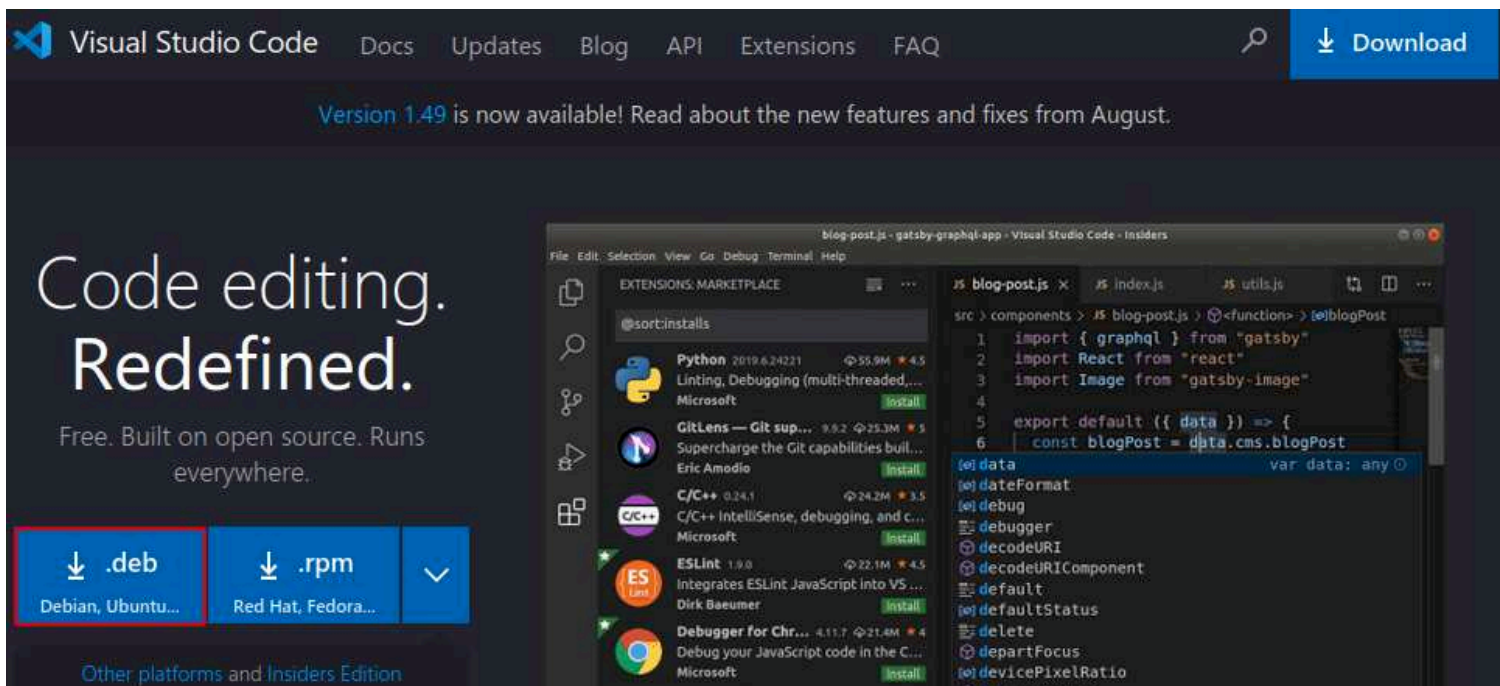
==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brew#donations

==> Next steps:
- Run `brew help` to get started
- Further documentation:
  https://docs.brew.sh
Santos-Air-2:~ Santos$ brew install python3
```

Now, [go to this section](#) to install PlatformIO IDE extension.

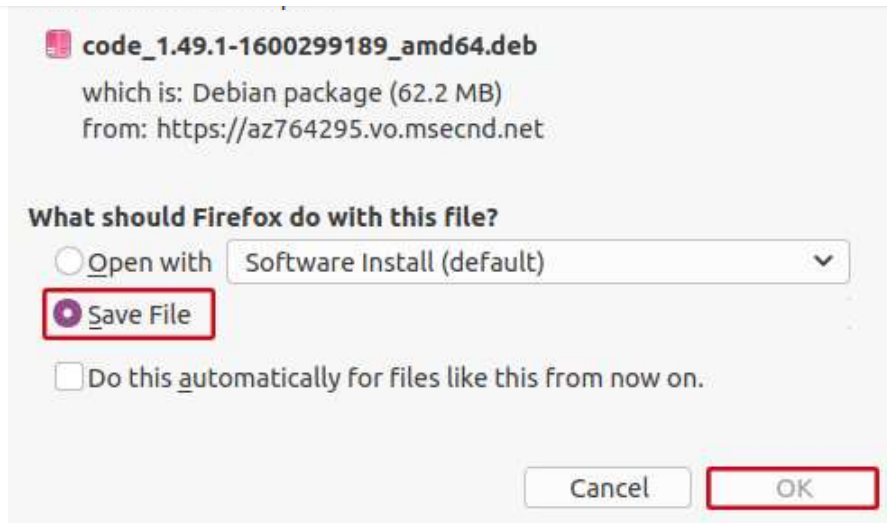
C) Installing VS Code on Linux Ubuntu (Visual Studio Code)

Go to <https://code.visualstudio.com/> and download the stable build for your operating system (Linux Ubuntu).



The screenshot shows the Visual Studio Code website. At the top, there are navigation links: Visual Studio Code, Docs, Updates, Blog, API, Extensions, and FAQ. A blue 'Download' button is in the top right. Below the navigation, a banner reads 'Version 1.49 is now available! Read about the new features and fixes from August.' The main content area features the text 'Code editing. Redefined.' and 'Free. Built on open source. Runs everywhere.' Below this, there are three download buttons: '.deb' (highlighted with a red box) for Debian, Ubuntu, etc.; '.rpm' for Red Hat, Fedora, etc.; and a dropdown arrow. At the bottom, it says 'Other platforms and Insiders Edition'. In the background, a blurred image of the VS Code interface shows the Extensions Marketplace with a list of installed and available extensions like Python, GitLens, C/C++, ESLint, and Debugger for Chrome.

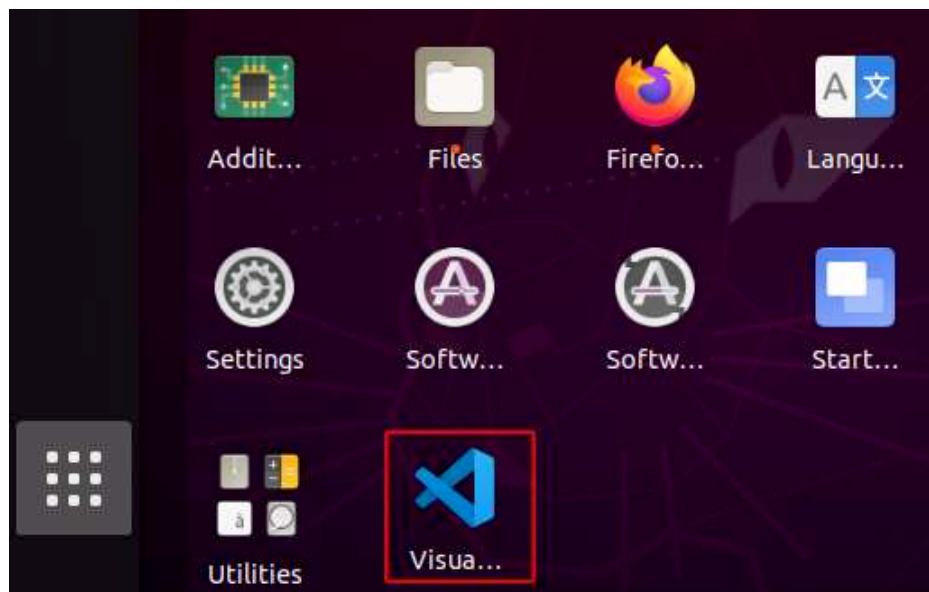
Save the installation file:



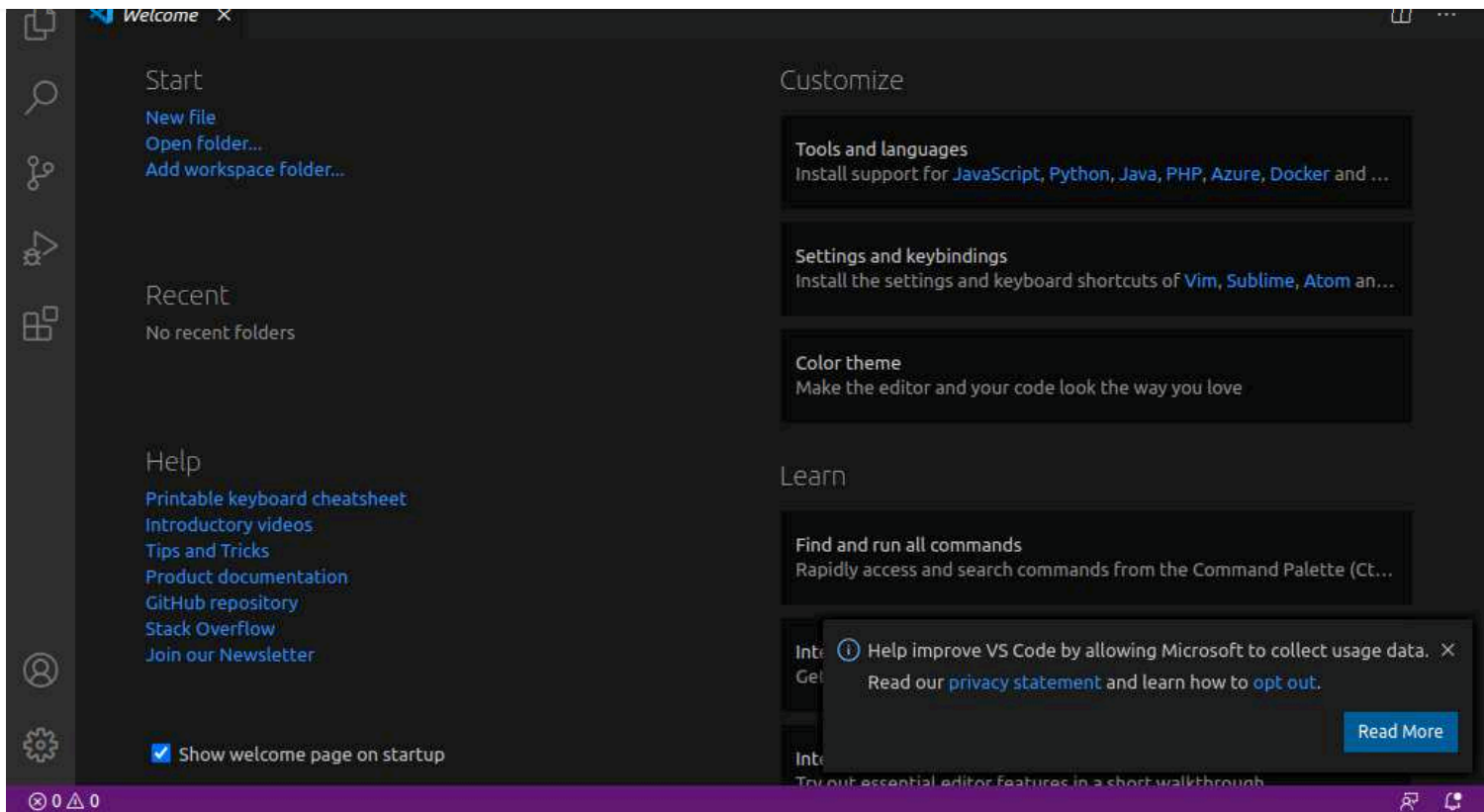
To install it, open a Terminal windows, navigate to your Downloads folder and run the following command to install VS Code.

```
$ cd Downloads  
~/Downloads $ sudo apt install ./code_1.49.1-1600299189_amd64.deb
```

When the installation is finished, VS Code should be available in your applications menu.



Open VS Code and you'll be greeted by a Welcome tab with the released notes of the newest version.



That's it. Visual Studio Code was successfully installed.

Installing Python on Linux Ubuntu

To program the ESP32 and ESP8266 boards with PlatformIO IDE you need Python 3.5 or higher installed in your computer. We're using Python 3.8.

Open the Terminal window and check that you already have Python 3 installed.

```
$ python3 --version
python 3.8.2
```

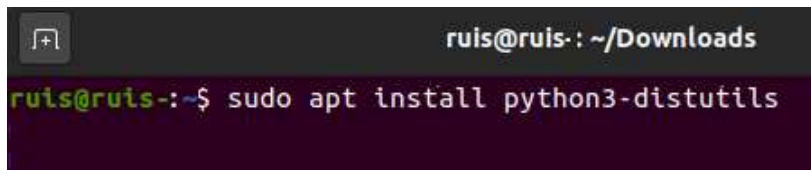
A terminal window screenshot showing the command 'python3 --version' being executed. The output is 'Python 3.8.2'. The prompt is 'ruis@ruis: ~\$'. The terminal has a dark background with light-colored text.

As you can see in the preceding figure, Python 3.8.2 is already installed.

If you don't have Python 3.8.X installed, run the next command to install it:

```
$ sudo apt install python3
```

```
$ sudo apt install python3-distutils
```



```
ruis@ruis: ~/Downloads
ruis@ruis:~$ sudo apt install python3-distutils
```

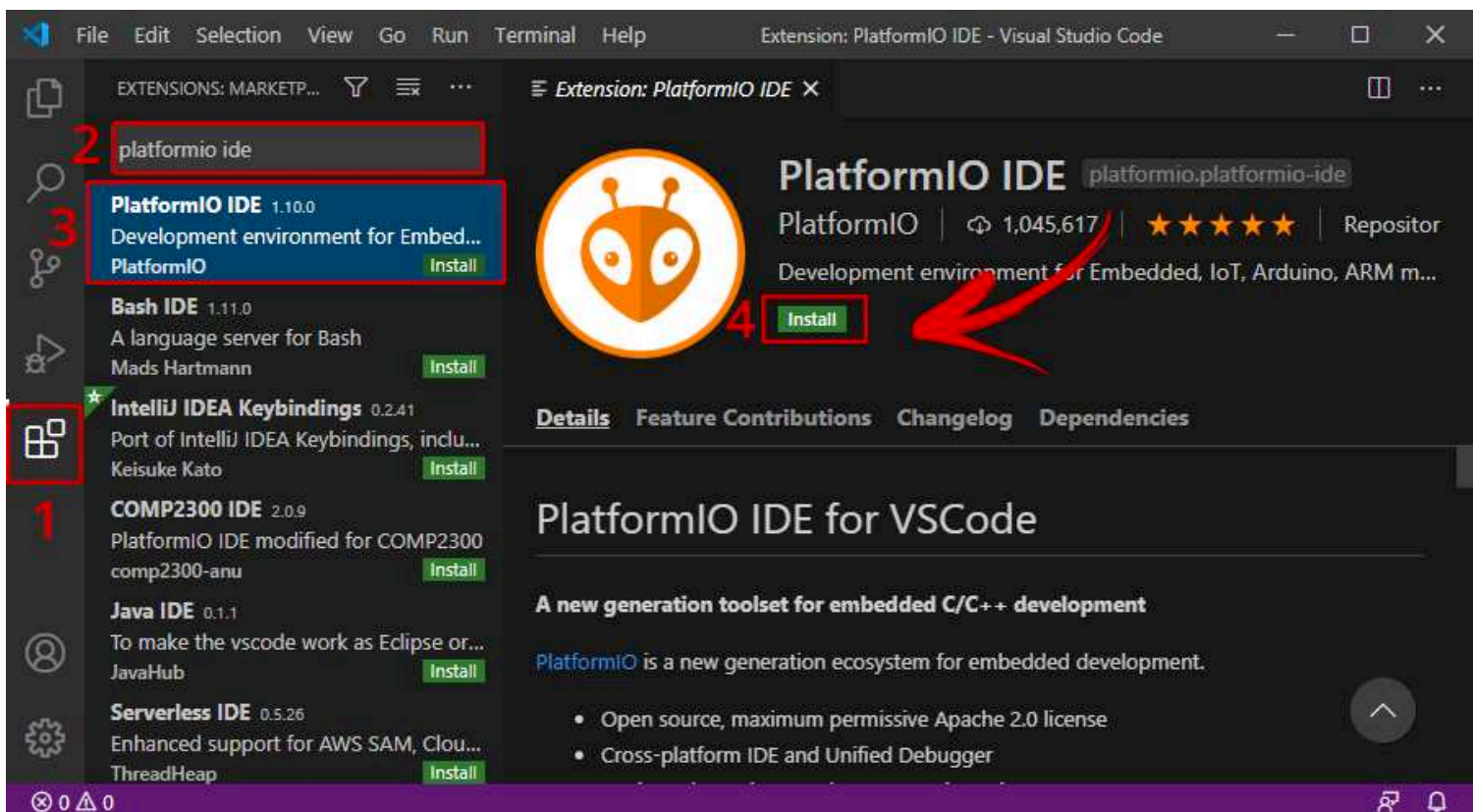
Now, [go to this section](#) to install PlatformIO IDE extension.

Installing PlatformIO IDE Extension on VS Code

It is possible to program the [ESP32](#) and [ESP8266](#) boards using VS Code with the PlatformIO IDE extension. Follow the next steps to install the PlatformIO IDE extension.

Open VS Code:

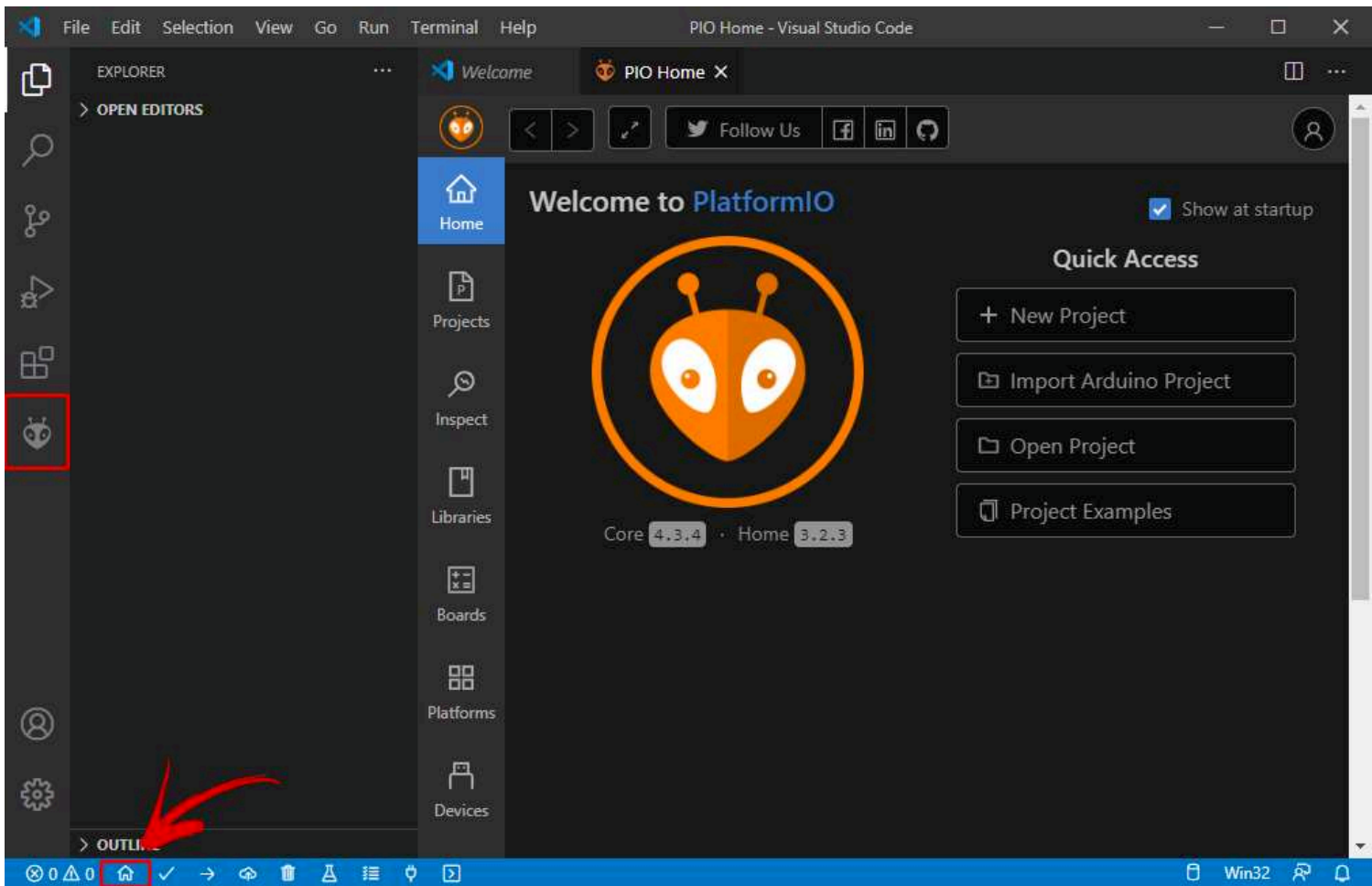
1. Click on the **Extensions** icon or press **Ctrl+Shift+X** to open the **Extensions** tab
2. Search for “**PlatformIO IDE**”
3. Select the first option
4. Finally, click the **Install** button (Note: the installation may take a few minutes)



After installing, make sure that PlatformIO IDE extension is enabled as shown below.



After that, the PlatformIO icon should show up on the left sidebar as well as an **Home** icon that redirects you to PlatformIO home.



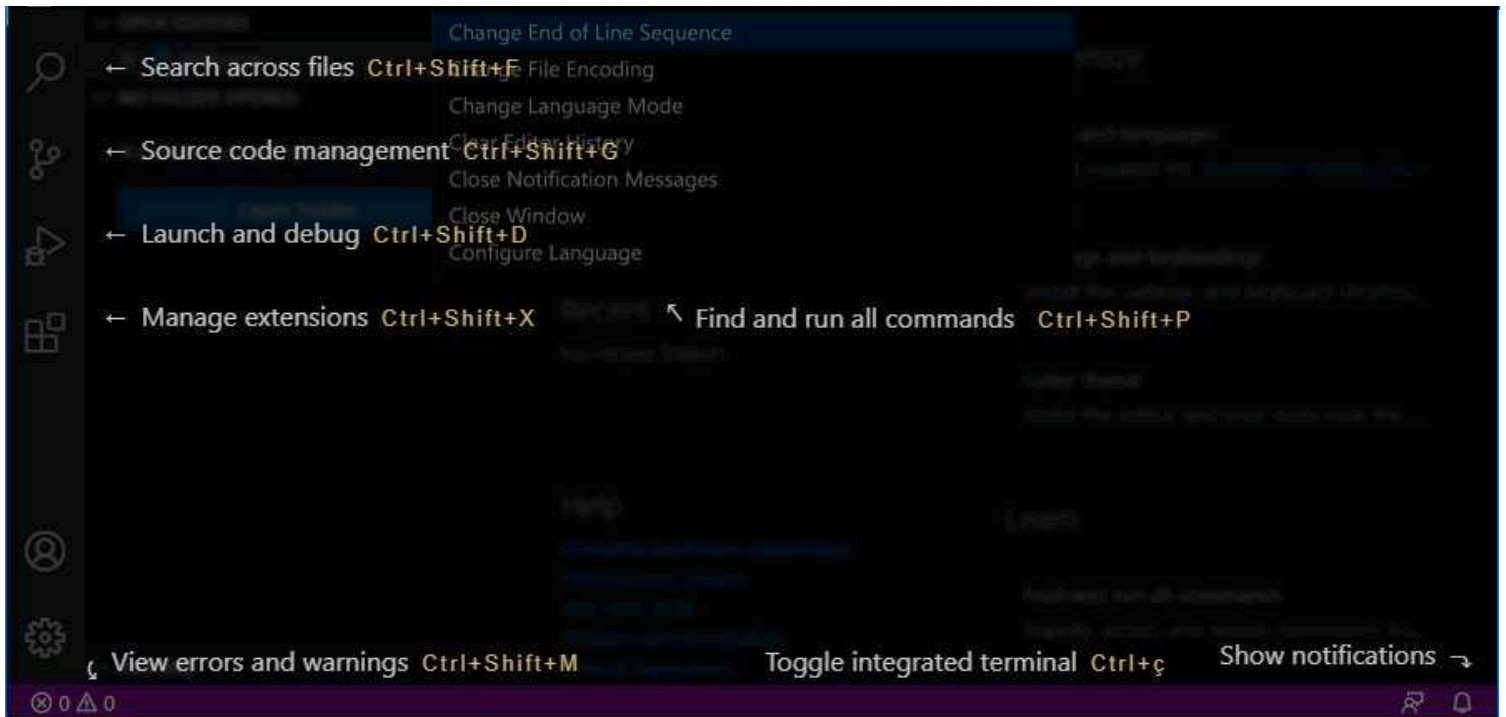
That's it, PlatformIO IDE extension was successfully added to VS Code.

If you don't see the **PIO** icon and the quick tools at the bottom, you may need to restart VS code for the changes to take effect.

Either way, we recommend restarting VS Code before proceeding.

VS Code Quick Interface Overview

Open VS Code. The following print screen shows the meaning of each icon on the left sidebar and its shortcuts:



- File explorer
- Search across files
- Source code management (using gist)
- Launch and debug your code
- Manage extensions

Additionally, you can press **Ctrl+Shift+P** or go to **View > Command Palette...** to show all the available commands. If you're searching for a command and you don't know where it is or its shortcut, you just need to go to the Command Palette and search for it.

At the bottom, there's a blue bar with PlatformIO commands.

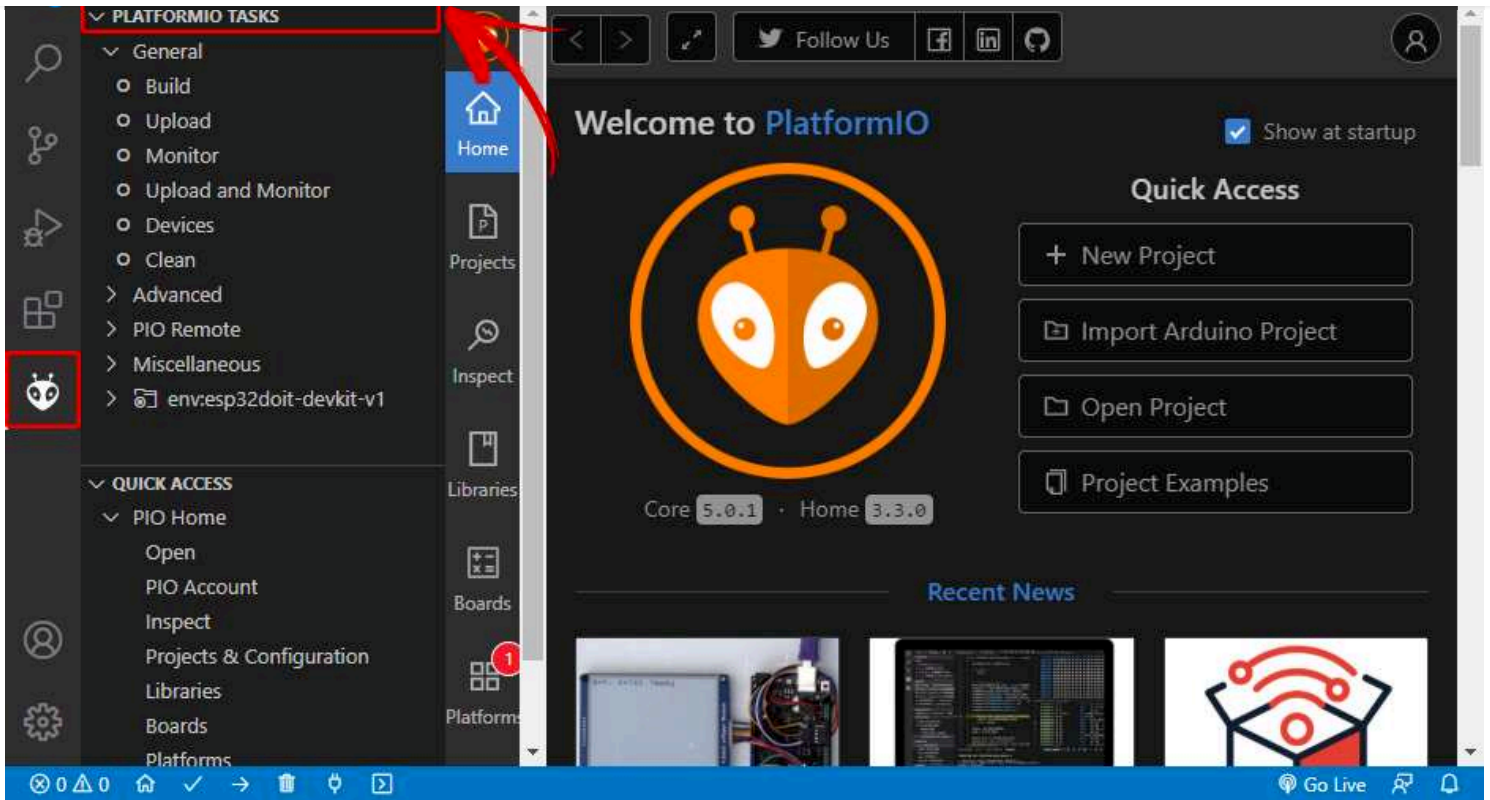


Here's the what icon does from left to right:

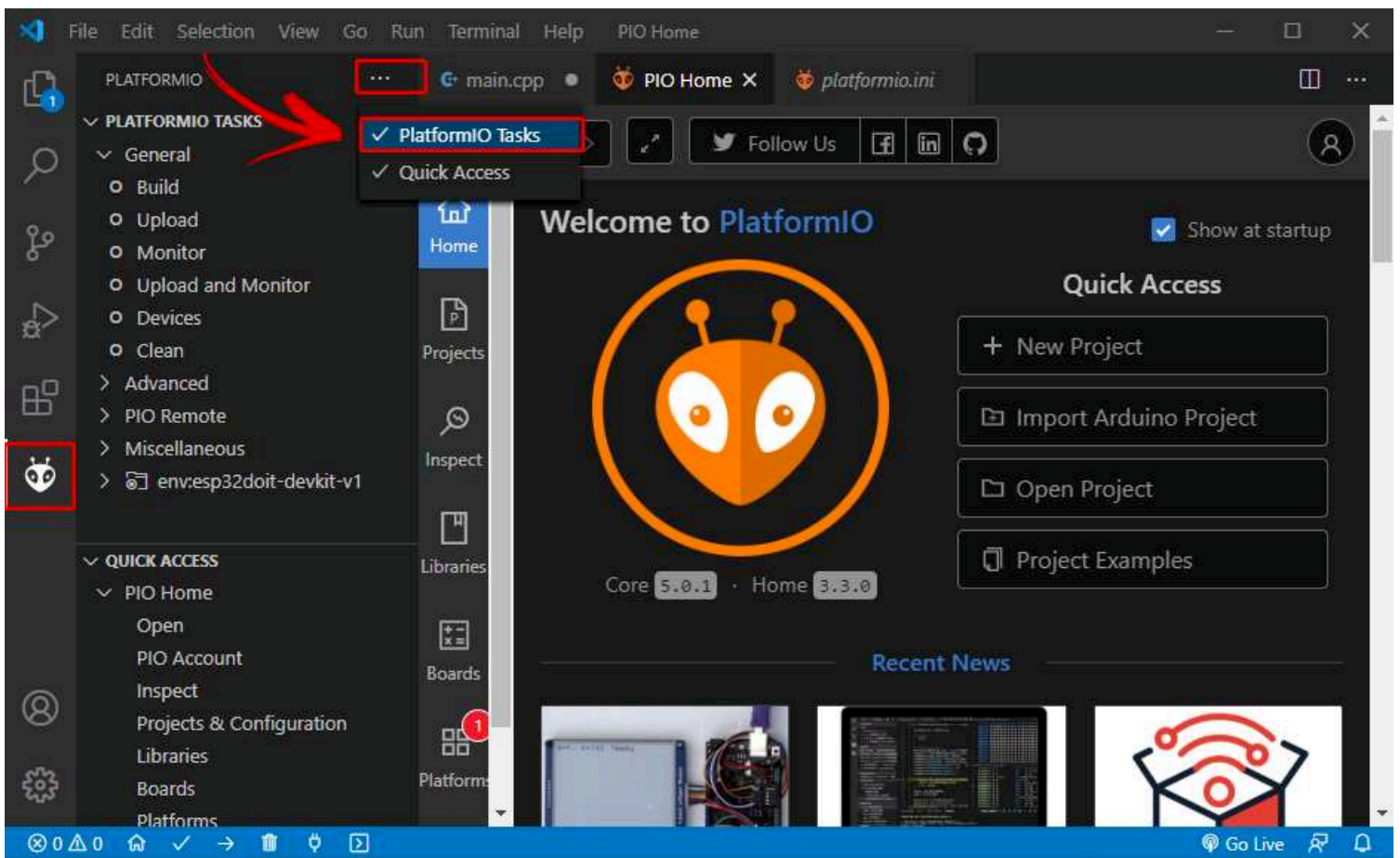
- PlatformIO Home
- Build/Compile
- Upload
- Clean
- Serial Monitor
- New Terminal

If you hover your mouse over the icons, it will show what each icon does.

Alternatively, you can also click on the PIO icon to see all the PlatformIO tasks.



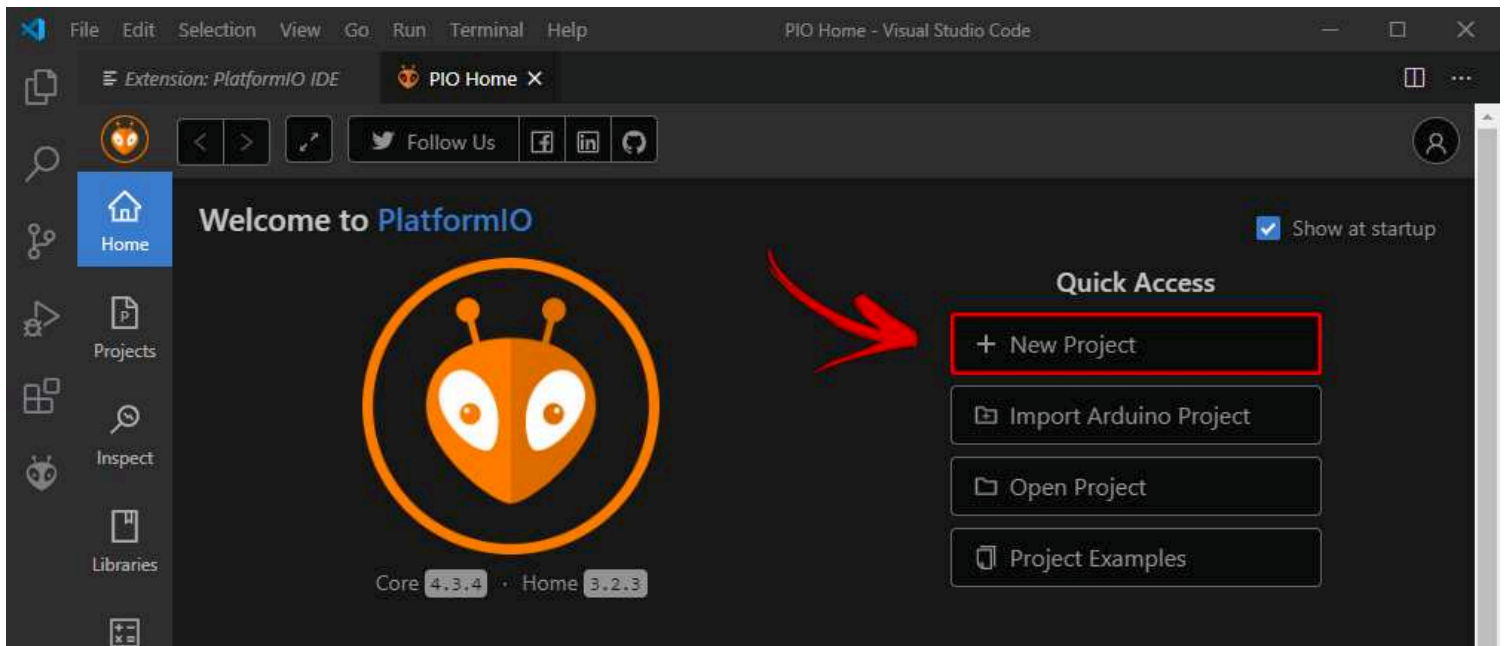
If the tasks don't show up on your IDE when you click the icon, you may need to click on the three dot icon at the top and enable PlatformIO tasks as shown below.



PlatformIO IDE Overview

Create a New Project

On VS Code, click on the PlatformIO **Home** icon. Click on **+ New Project** to start a new project.



Give your project a name (for example *Blink_LED*) and select the board you're using. In our case, we're using the **DOIT ESP32 DEVKIT V1**. The Framework should be "Arduino" to use the Arduino core.

You can choose the default location to save your project or a custom location.

The default location is in this path *Documents > PlatformIO > Projects*. For this test, you can use the default location. Finally, click "Finish".

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name:

Board:

Framework:

Location: Use default location ?

For this example, we'll be using the [DOIT ESP32 DEVKIT board](#). If you are using an [ESP8266 NodeMCU board](#) the process is very similar, you just need to select your ESP8266 board:

Project Wizard ✕

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name:

Board:

Framework:

Location: Use default location ?

The Blink_LED project should be accessible from the Explorer tab.

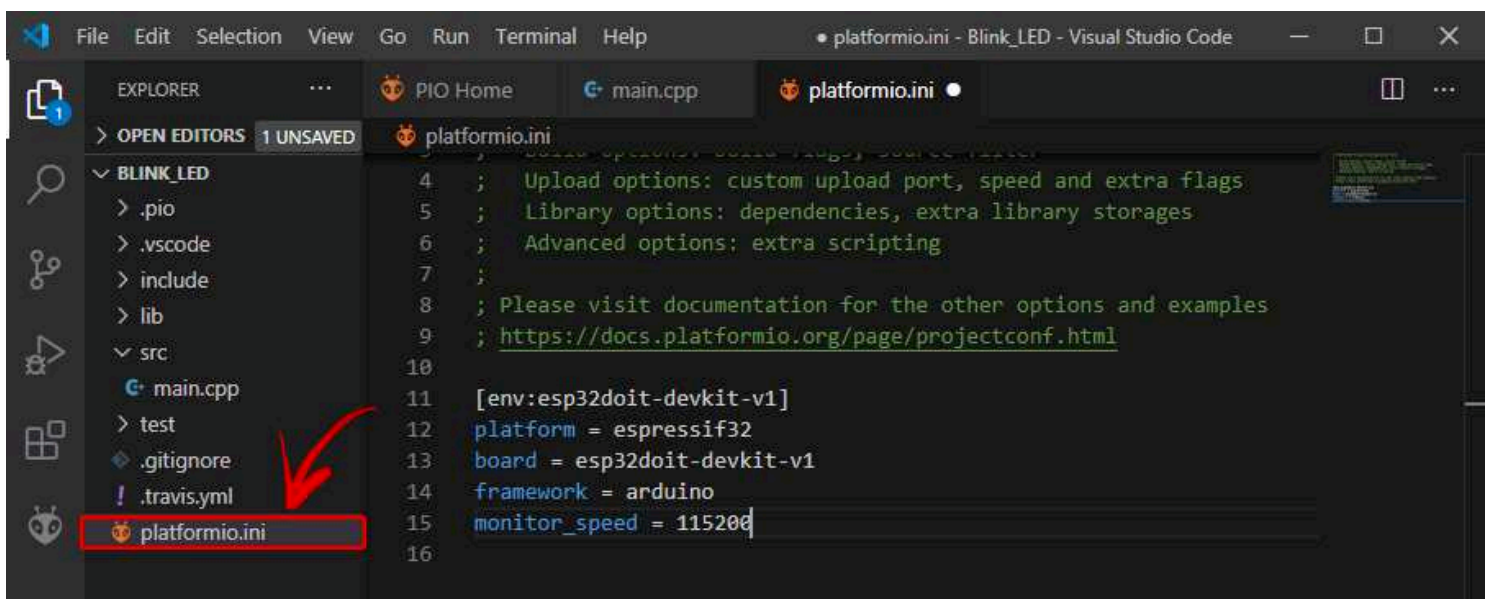


VS Code and PlatformIO have a folder structure that is different from the standard *.ino* project. If you click on the Explorer tab, you'll see all the files it created under your project folder. It may seem a lot of files to work with. But, don't worry, usually you'll just need to deal with one or two of those files.

Warning: you shouldn't delete, modify or move the folders and the *platformio.ini* file. Otherwise, you will no longer be able to compile your project using PlatformIO.

platformio.ini file

The *platformio.ini* file is the PlatformIO Configuration File for your project. It shows the platform, board, and framework for your project. You can also add other configurations like libraries to be included, upload options, changing the Serial Monitor baud rate and other configurations.



- **platform:** which corresponds to the SoC used by the board.
- **board:** the development board you're using.
- **framework:** the software environment that will run the project code.

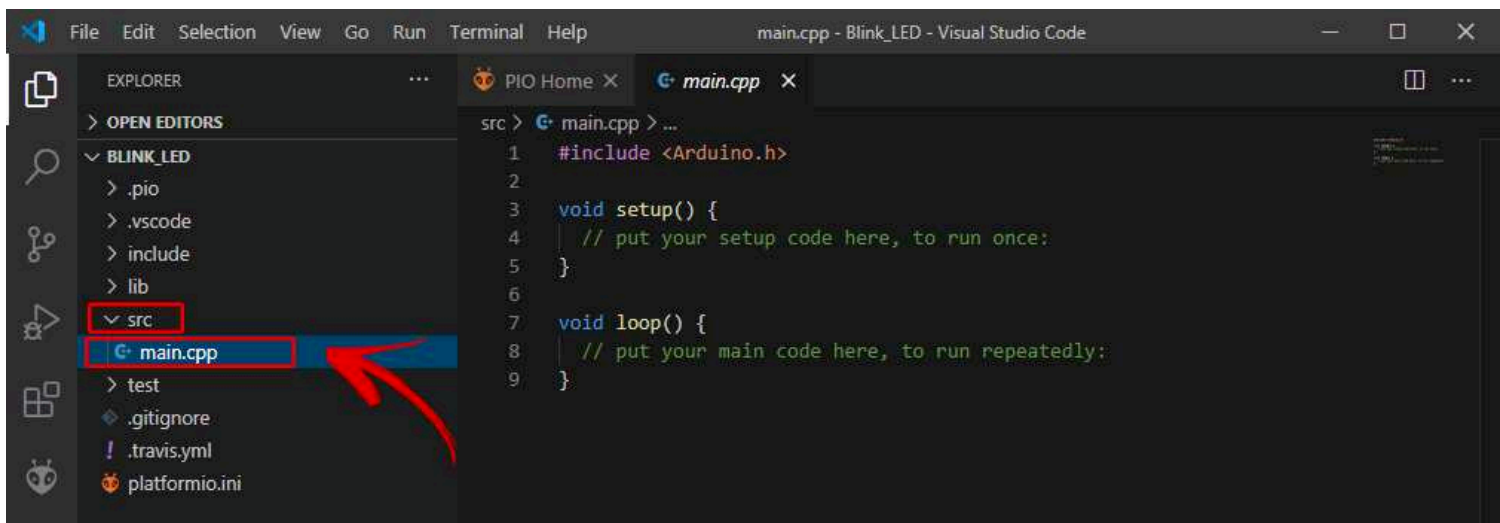
With the ESP32 and ESP8266, if you want to use a baud rate of 115200 in your Serial Monitor, you just need to add the following line to your *platformio.ini* file.

After that, make sure you save the changes made to the file by pressing **Ctrl+S**.

In this file, you can also include the identifier of libraries you'll use in your project using the `lib_deps` directive, as we'll see later.

src folder

The `src` folder is your working folder. Under the `src` folder, there's a `main.cpp` file. That's where you write your code. Click on that file. The structure of an Arduino program should open with the `setup()` and `loop()` functions.



In PlatformIO, all your Arduino sketches should start with the `#include <Arduino.h>`.

Uploading Code using PlatformIO IDE: ESP32/ESP8266

Copy the following code to your `main.cpp` file.

```
/*
 * *****
 * Rui Santos
 * Complete project details at https://RandomNerdTutorials.com/vs-code-platformio-ide
 * *****
 */

#include <Arduino.h>

#define LED 2

void setup() {
    // put your setup code here, to run once:
```

```
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
    digitalWrite(LED, HIGH);  
    Serial.println("LED is on");  
    delay(1000);  
    digitalWrite(LED, LOW);  
    Serial.println("LED is off");  
    delay(1000);  
}
```

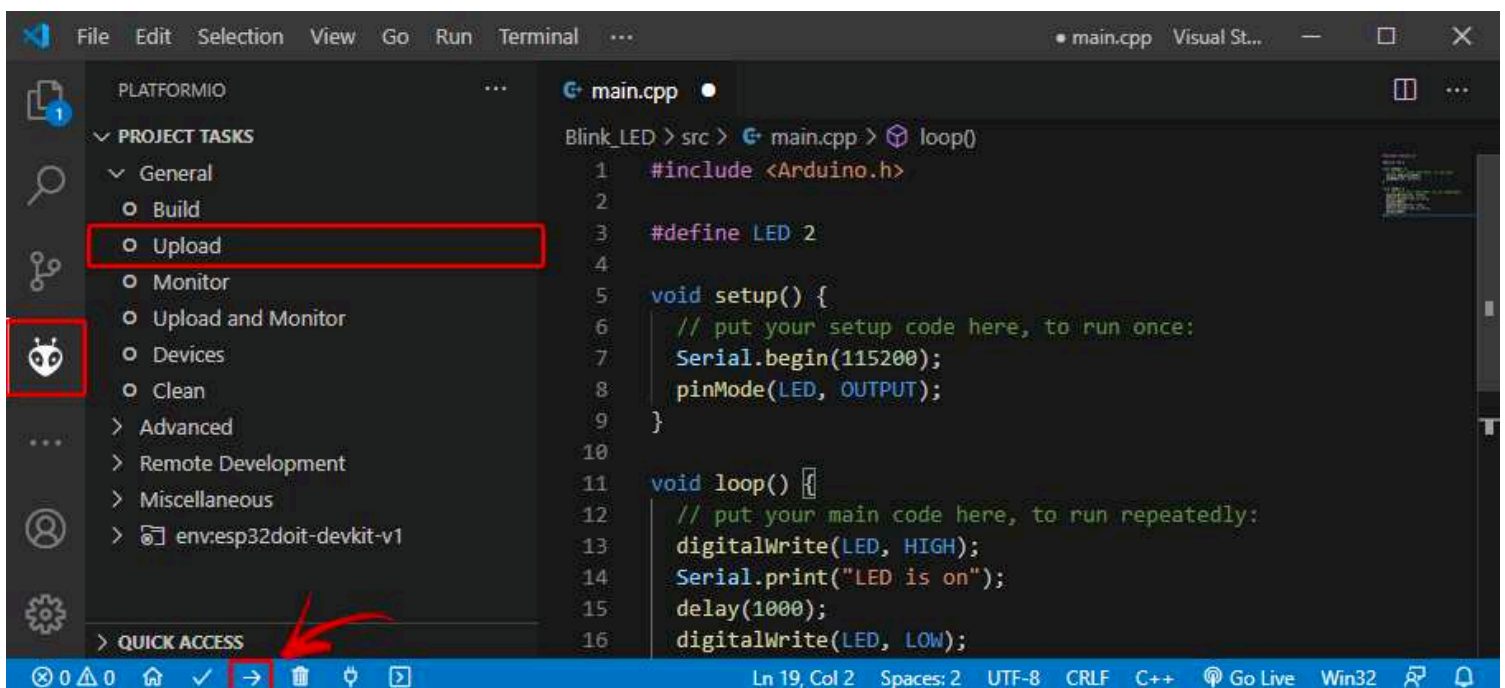
[View raw code](#)

This code blinks the on-board LED every second. It works with the ESP32 and ESP8266 boards (both have the on-board LED connected to GPIO 2).

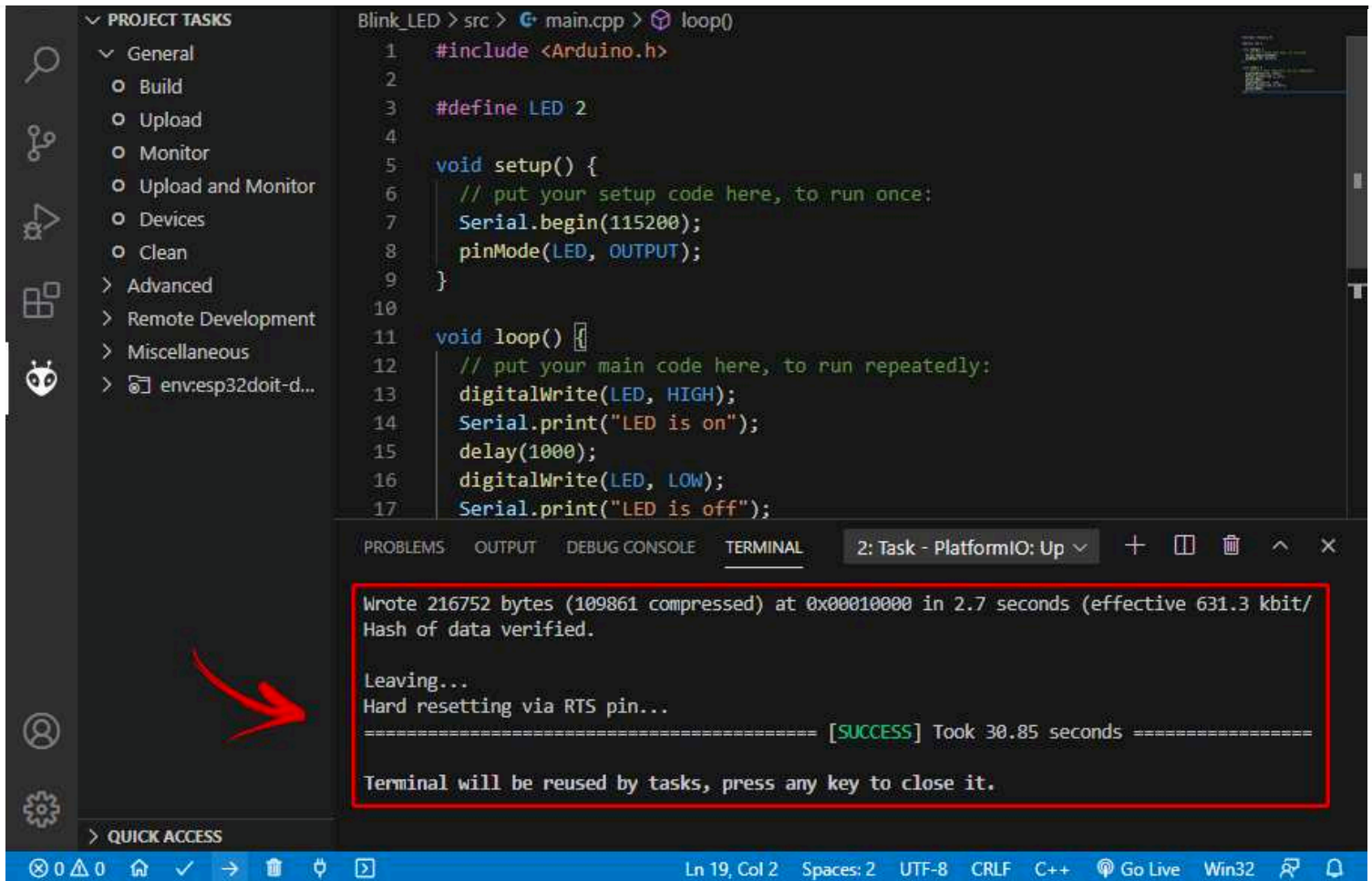
We recommend that you copy this code manually, so that you see the autocompletion and other interesting features of the IDE in action. Additionally, if you have a syntax error somewhere in your program, it will underline it in red even before compiling.

After that, press **Ctrl+S** or go to **File > Save** to save the file.

Now, you can click on the Upload icon to compile and upload the code. Alternatively, you can go to the PIO Project Tasks menu and select **Upload**.



If the code is successfully uploaded, you should get the following message.



The screenshot shows an IDE interface with a project named 'Blink_LED'. The code in the editor is as follows:

```
Blink_LED > src > main.cpp > loop()
1  #include <Arduino.h>
2
3  #define LED 2
4
5  void setup() {
6    // put your setup code here, to run once:
7    Serial.begin(115200);
8    pinMode(LED, OUTPUT);
9  }
10
11 void loop() {
12   // put your main code here, to run repeatedly:
13   digitalWrite(LED, HIGH);
14   Serial.print("LED is on");
15   delay(1000);
16   digitalWrite(LED, LOW);
17   Serial.print("LED is off");
18 }
```

The terminal window shows the following output:

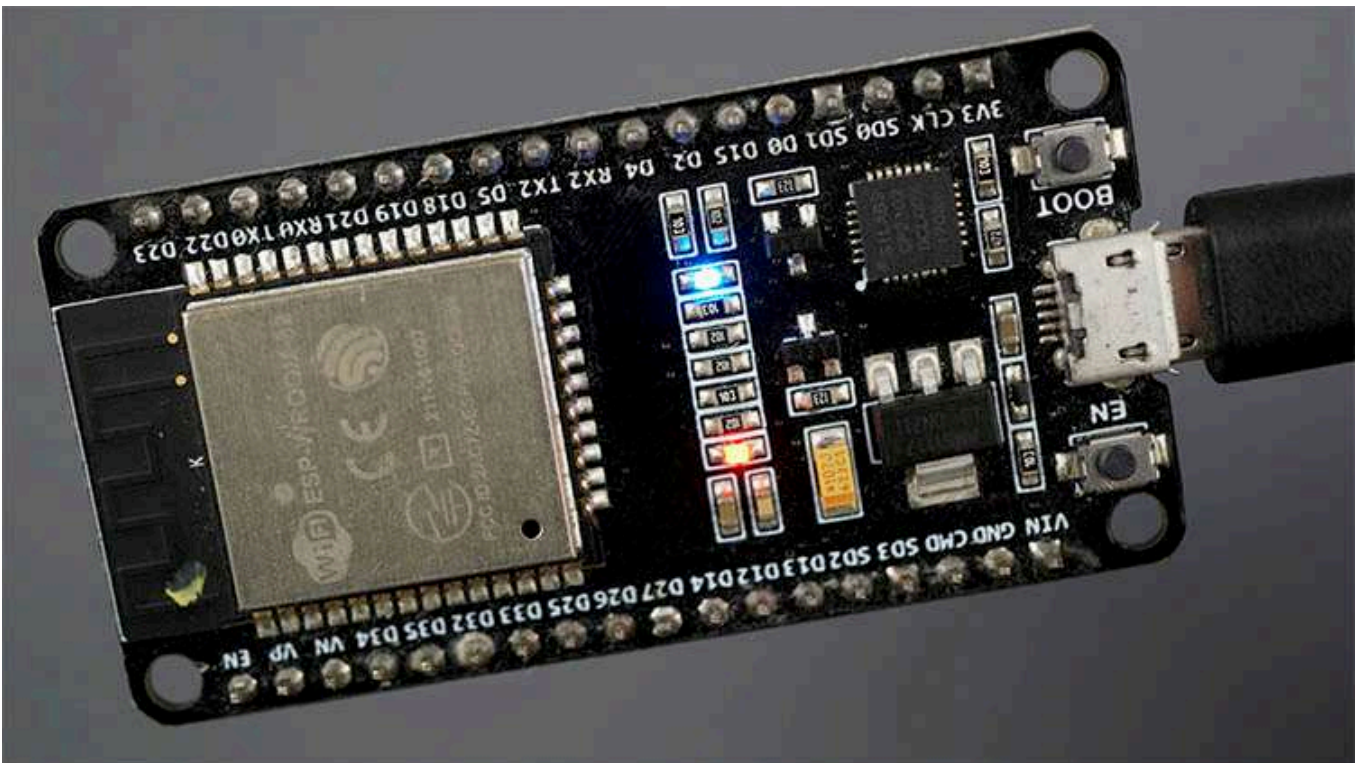
```
Wrote 216752 bytes (109861 compressed) at 0x00010000 in 2.7 seconds (effective 631.3 kbit/
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 30.85 seconds =====

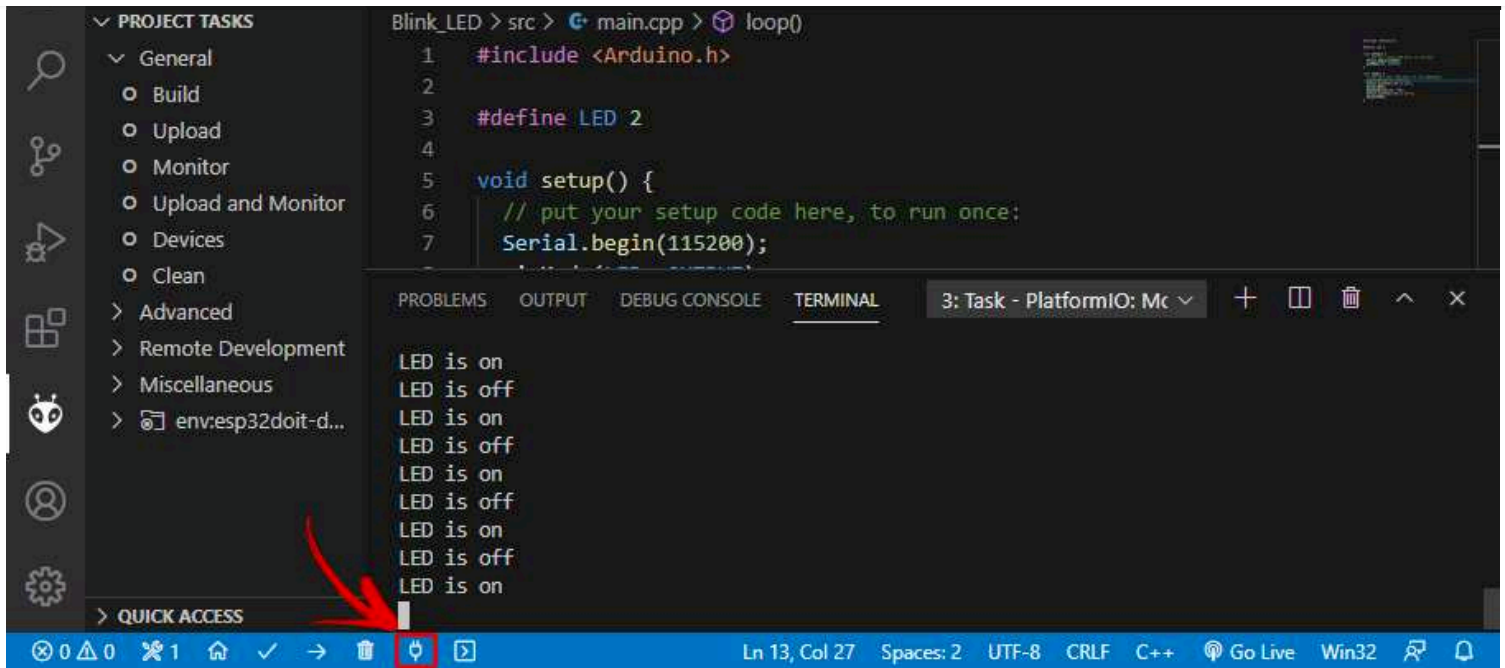
Terminal will be reused by tasks, press any key to close it.
```

A red arrow points to the terminal output. The status bar at the bottom indicates 'Ln 19, Col 2 Spaces: 2 UTF-8 CRLF C++ Go Live Win32'.

After uploading the code, the ESP32 or ESP8266 should be blinking its on-board LED every second.



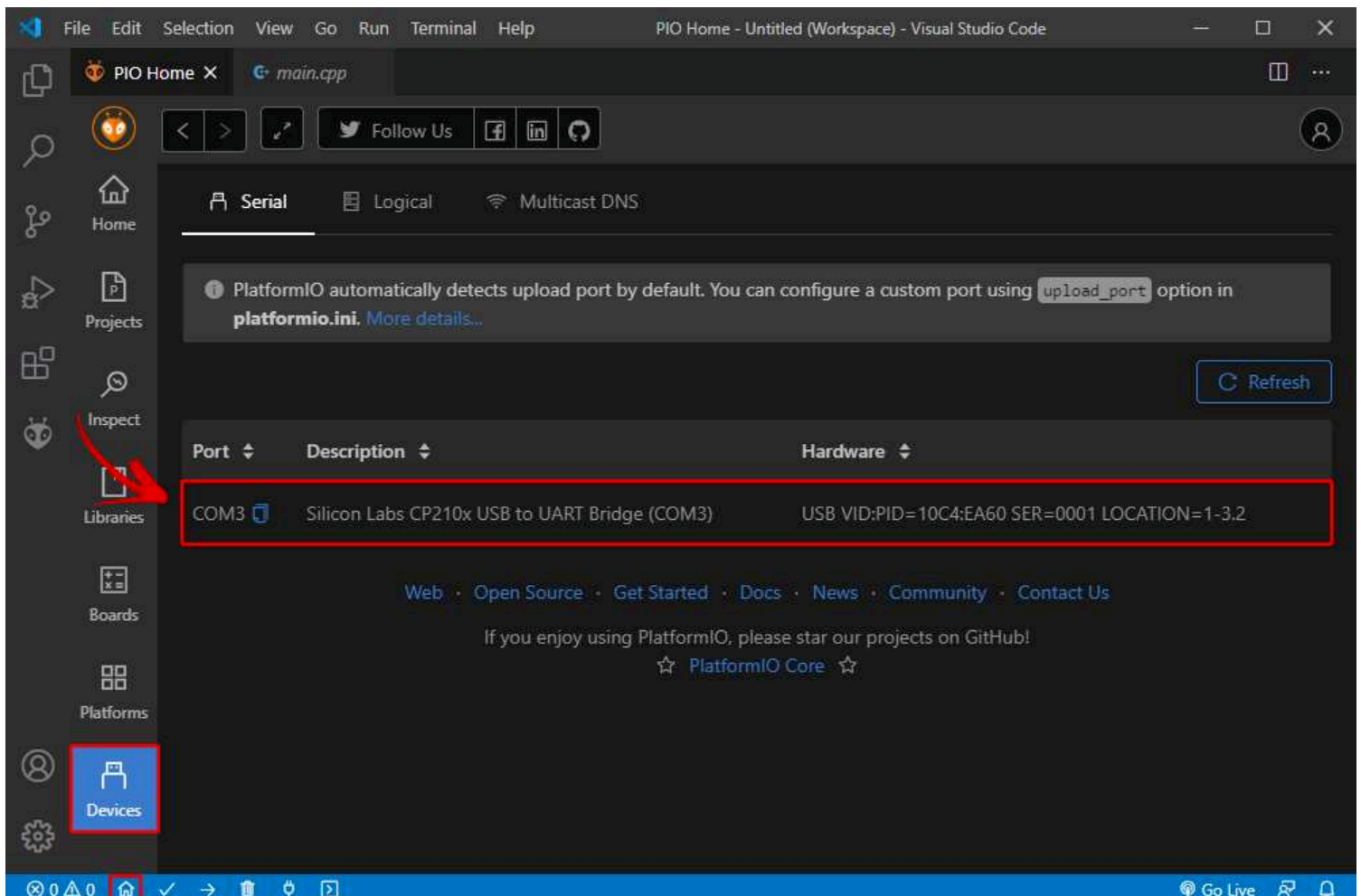
Now, click on the Serial Monitor icon and you should see it printing the current LED state.



Note: if you don't see the Terminal window, go to the menu Terminal > New Terminal.

Detect COM Port

PlatformIO will automatically detect the port your board is connected to. To check the connected devices you can go to the **PIO Home** and click the **Devices** icon.



1) If you try to upload a new sketch to your ESP32 and you get this error message *A fatal error occurred. Failed to connect to ESP32: Timed out... Connecting...*. It means that your ESP32 is not in flashing/uploading mode.

Having the right board name and COM port selected, follow these steps:

- Hold-down the **BOOT** button in your ESP32 board
- Press the **Upload** button in the Arduino IDE to upload your sketch
- After you see the *“Connecting....”* message in your Arduino IDE, release the finger from the **BOOT** button
- After that, you should see the *“Done uploading”* message

You'll also have to repeat that button sequence every time you want to upload a new sketch. But if you want to solve this issue once for all without the need to press the **BOOT** button, follow the suggestions in the next guide:

- [\[SOLVED\] Failed to connect to ESP32: Timed out waiting for packet header](#)

2) If you get the error “COM Port not found/not available”, you might need to install the CP210x Drivers:

- [Install USB Drivers – CP210x USB to UART Bridge \(Windows PC\)](#)
- [Install USB Drivers – CP210x USB to UART Bridge \(Mac OS X\)](#)

If you experience any problems or issues with your ESP32, take a look at our in-depth [ESP32 Troubleshooting Guide](#).

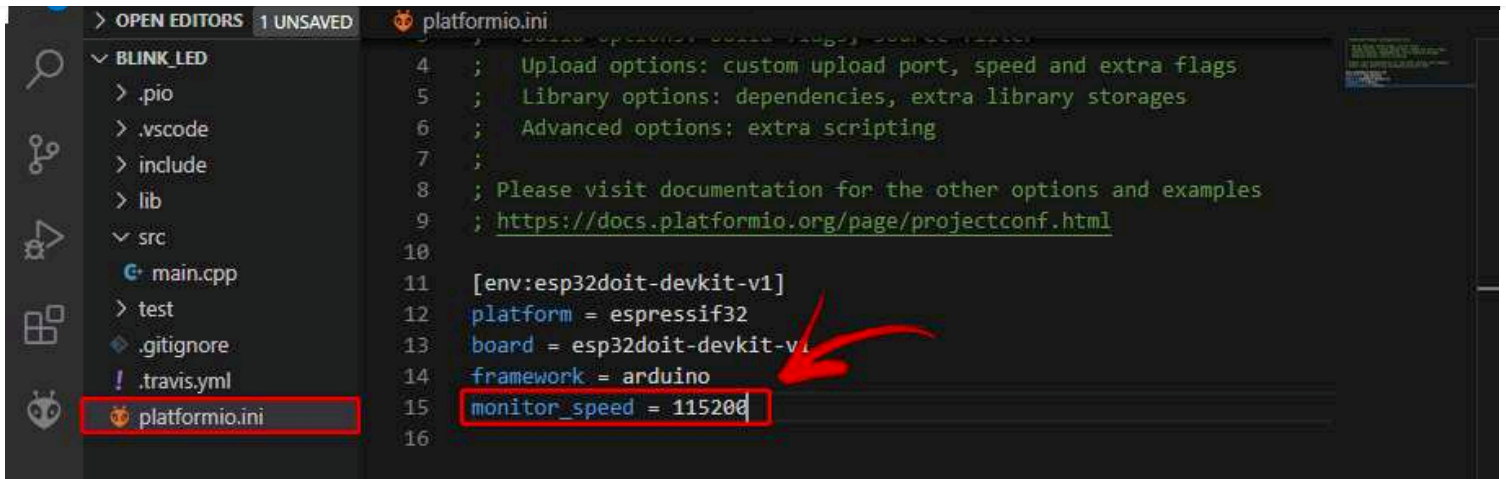
Changing the Serial Monitor Baud Rate – PlatformIO IDE

The default baud rate used by PlatformIO is 9600. However, it is possible to set up a different value as mentioned previously. On the File Explorer, under your project folder, open the *platformio.ini* file and add the following line:

```
monitor_speed = baud_rate
```

For example:

```
monitor_speed = 115200
```



The screenshot shows the PlatformIO IDE interface. The left sidebar displays a file explorer with the following structure:

- BLINK_LED
 - .pio
 - .vscode
 - include
 - lib
 - src
 - main.cpp
 - test
 - .gitignore
 - .travis.yml
- platformio.ini

The main editor window shows the contents of the `platformio.ini` file:

```
4 ; Upload options: custom upload port, speed and extra flags.
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:esp32doit-devkit-v1]
12 platform = espressif32
13 board = esp32doit-devkit-v1
14 framework = arduino
15 monitor_speed = 115200
16
```

A red arrow points to the `monitor_speed = 115200` line, and a red box highlights this line.

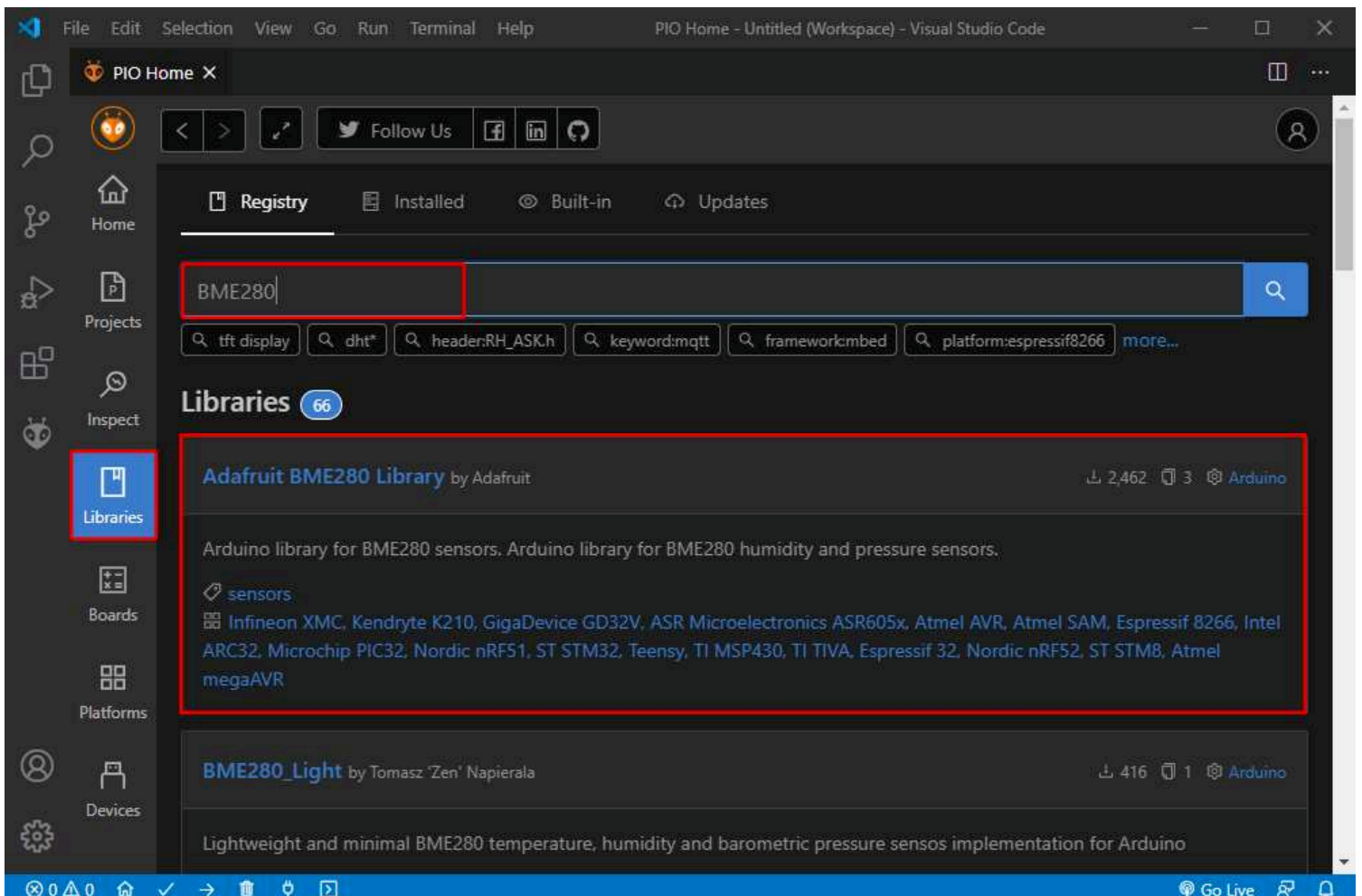
After that, save that file.

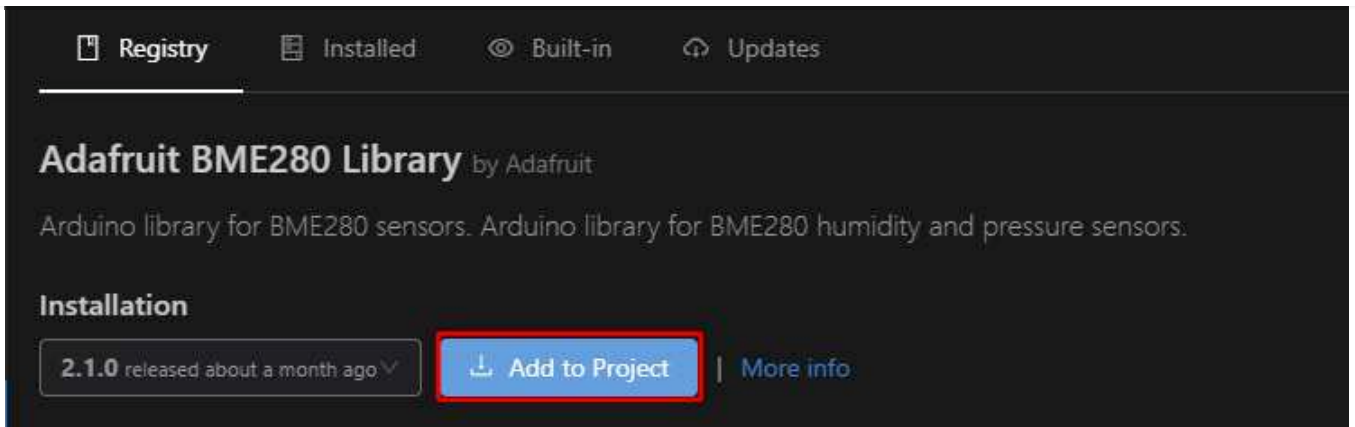
Installing ESP32/ESP8266 Libraries on PlatformIO IDE

Follow the next procedure if you need to install libraries in PlatformIO IDE.

Click the **Home** icon to go to PlatformIO Home. Click on the **Libraries** icon on the left side bar.

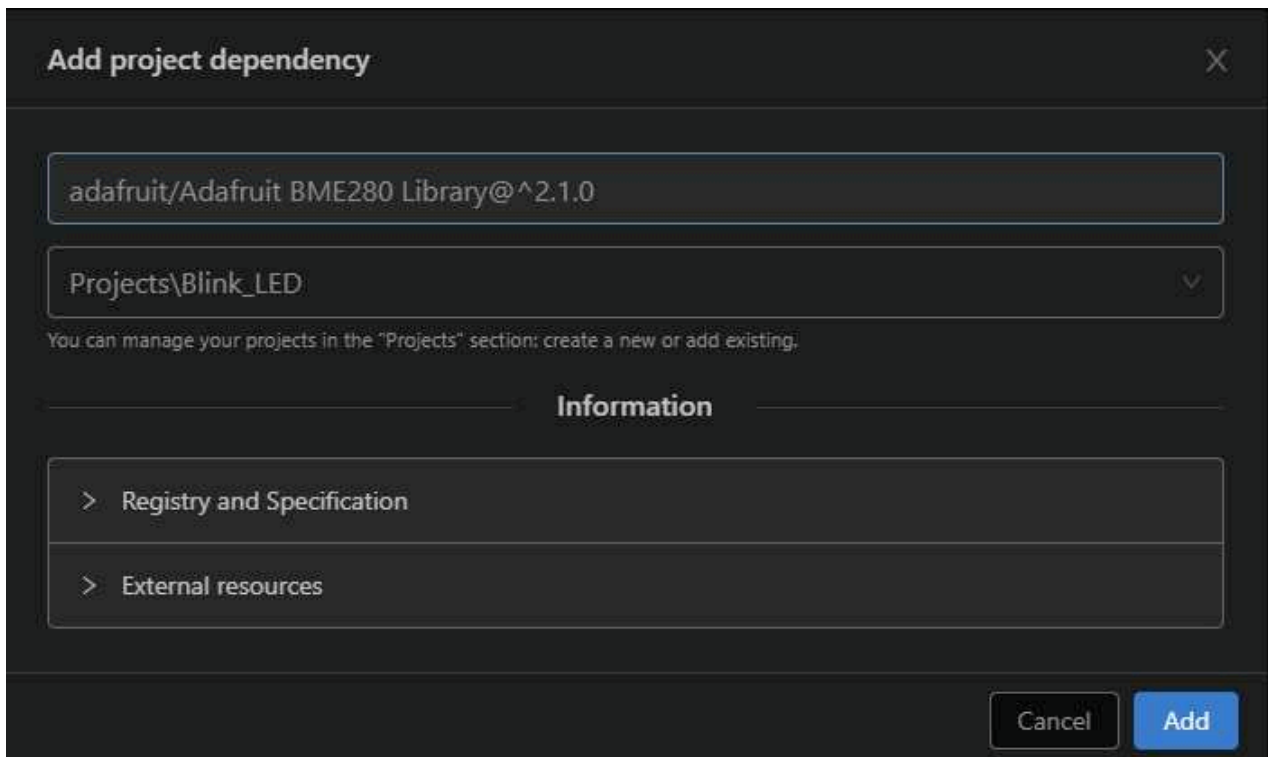
Search for the library you want to install. For example *Adafruit_BME280*.





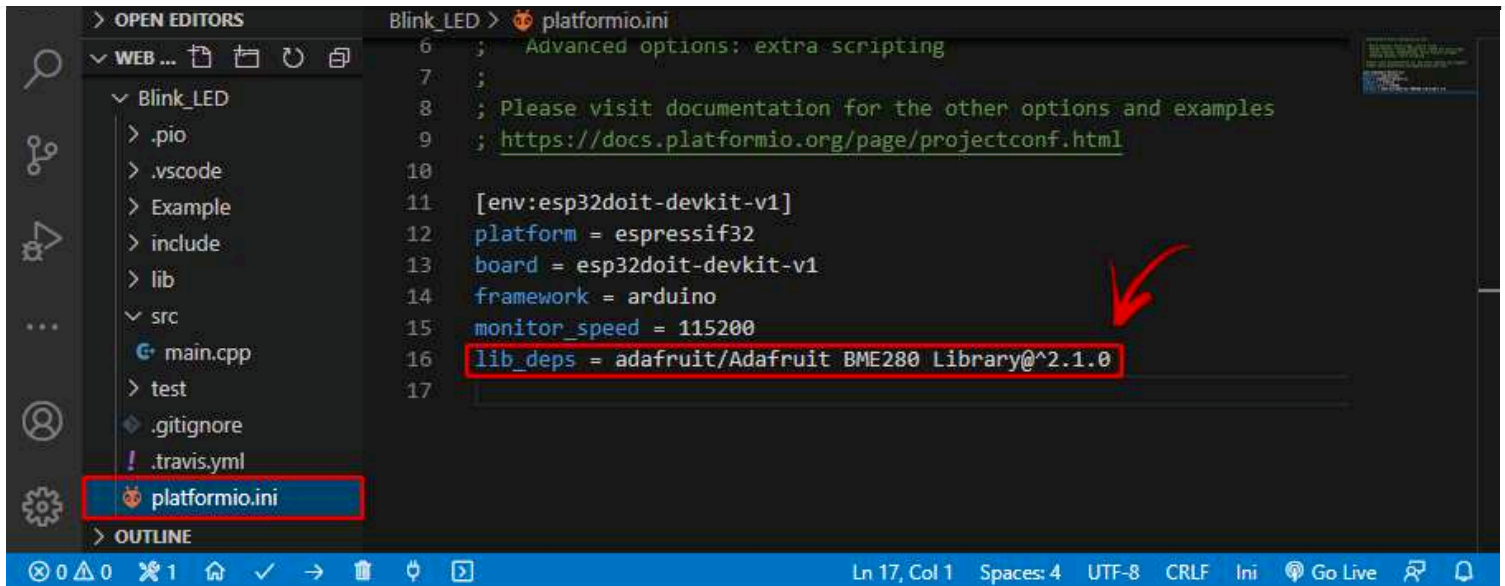
The screenshot shows the 'Registry' tab in the Arduino IDE. The library 'Adafruit BME280 Library' by Adafruit is displayed. Below the library name, there is a description: 'Arduino library for BME280 sensors. Arduino library for BME280 humidity and pressure sensors.' Under the 'Installation' section, the version '2.1.0' is shown with a dropdown arrow. A blue button labeled 'Add to Project' with a download icon is highlighted with a red border. To the right of this button is a link for 'More info'.

Select the project where you want to use the library.



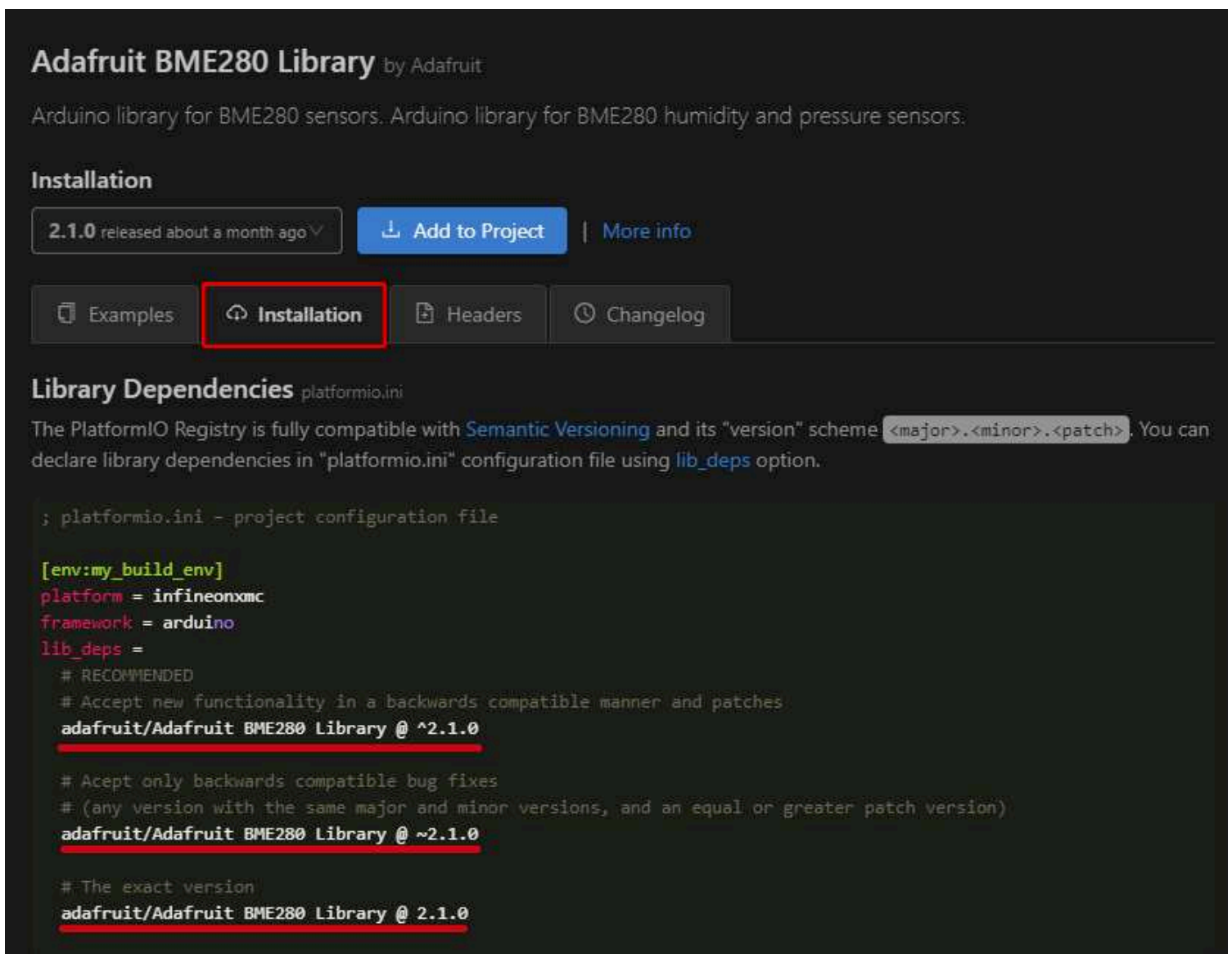
The screenshot shows the 'Add project dependency' dialog box. The title bar says 'Add project dependency' with a close button (X) on the right. The main area contains a text input field with the text 'adafruit/Adafruit BME280 Library@^2.1.0'. Below this is a dropdown menu showing 'Projects\Blink_LED'. A note below the dropdown reads: 'You can manage your projects in the "Projects" section: create a new or add existing.' Underneath is an 'Information' section with two expandable items: '> Registry and Specification' and '> External resources'. At the bottom right, there are two buttons: 'Cancel' and 'Add'.

This will add the library identifier using the `lib_deps` directive on the `platformio.ini` file. If you open your project's `platformio.ini` file, it should look as shown in the following image.



```
> OPEN EDITORS      Blink_LED > platformio.ini
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:esp32doit-devkit-v1]
12 platform = espressif32
13 board = esp32doit-devkit-v1
14 framework = arduino
15 monitor_speed = 115200
16 lib_deps = adafruit/Adafruit BME280 Library@^2.1.0
17
```

Alternatively, on the library window, if you select the **Installation** tab and scroll a bit, you'll see the identifier for the library. You can choose any of those identifiers depending on the options you want to use. The library identifiers are highlighted in red.



Adafruit BME280 Library

by Adafruit

Arduino library for BME280 sensors. Arduino library for BME280 humidity and pressure sensors.

Installation

2.1.0 released about a month ago ▾ [Add to Project](#) | [More info](#)

[Examples](#) **[Installation](#)** [Headers](#) [Changelog](#)

Library Dependencies

platformio.ini

The PlatformIO Registry is fully compatible with [Semantic Versioning](#) and its "version" scheme `<major>.<minor>.<patch>`. You can declare library dependencies in "platformio.ini" configuration file using `lib_deps` option.

```
; platformio.ini - project configuration file

[env:my_build_env]
platform = infineonxmc
framework = arduino
lib_deps =
# RECOMMENDED
# Accept new functionality in a backwards compatible manner and patches
adafruit/Adafruit BME280 Library @ ^2.1.0

# Accept only backwards compatible bug fixes
# (any version with the same major and minor versions, and an equal or greater patch version)
adafruit/Adafruit BME280 Library @ ~2.1.0

# The exact version
adafruit/Adafruit BME280 Library @ 2.1.0
```

Then, go to the `platformio.ini` file of your project and paste the library identifier into that file, like this:

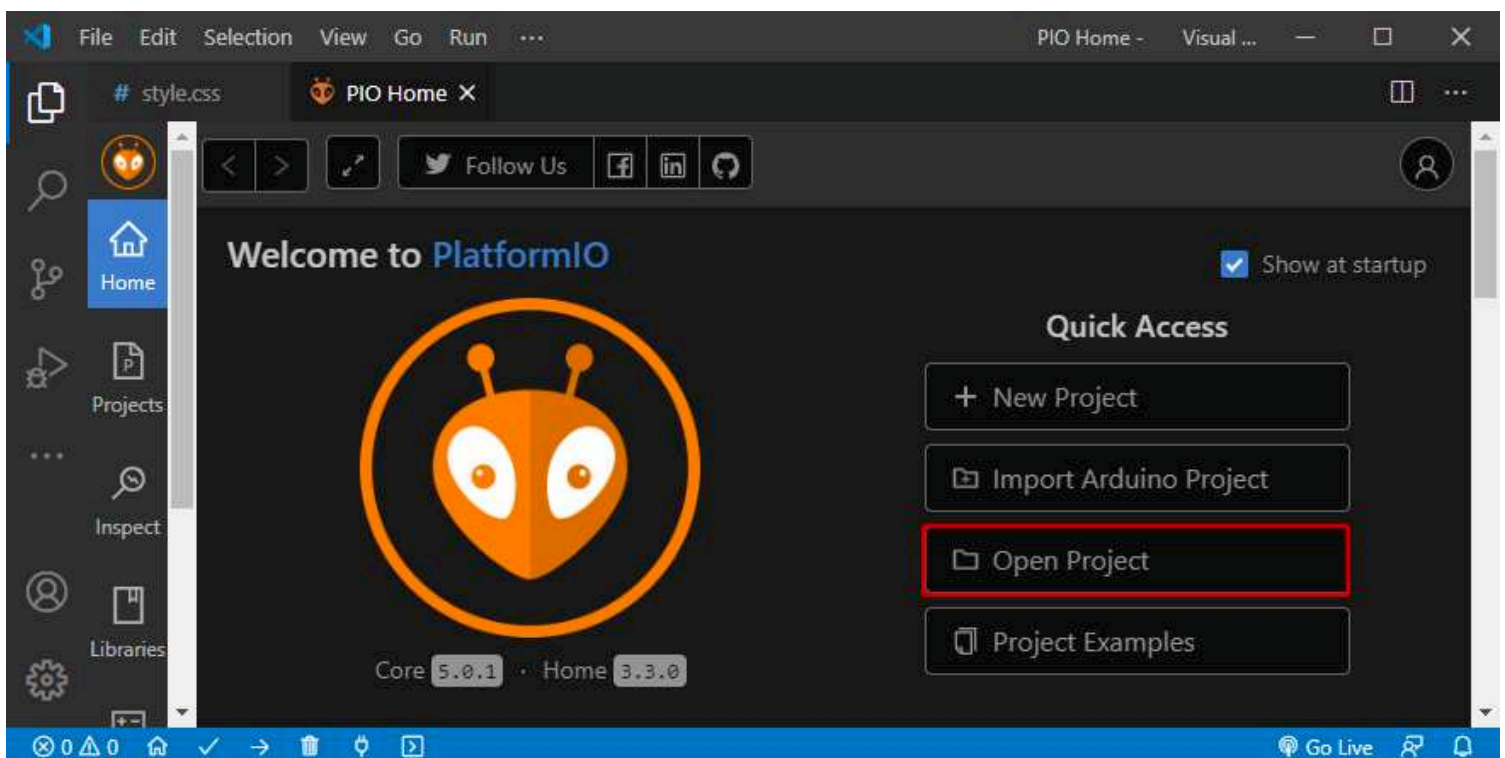
If you need multiple libraries, you can separate their name by a coma or put them on different lines. For example:

```
lib_deps =
  arduino-libraries/Arduino_JSON @ 0.1.0
  adafruit/Adafruit BME280 Library @ ^2.1.0
  adafruit/Adafruit Unified Sensor @ ^1.1.4
```

PlatformIO has a built-in powerful Library Manager, that allows you to specify custom dependencies per project in the Project Configuration File *platformio.ini* using `lib_deps`. This will tell PlatformIO to automatically download the library and all its dependencies when you save the configuration file or when you compile your project.

Open a Project Folder

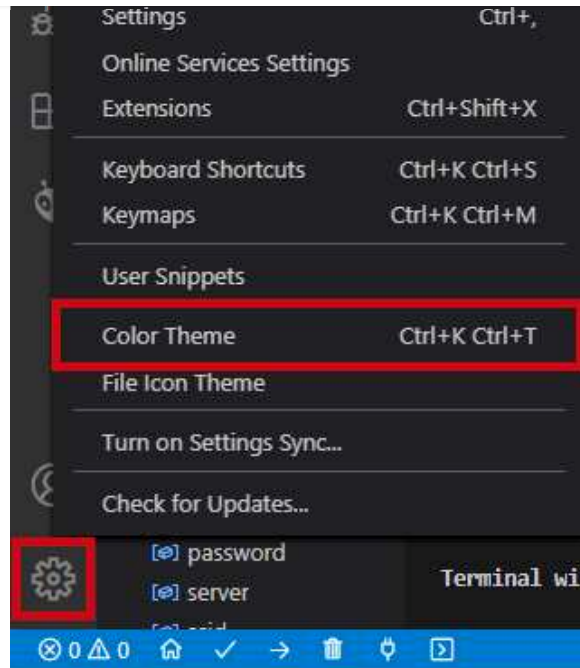
To open an existing project folder on PlatformIO, open VS Code, go to PlatformIO Home and click on **Open Project**. Navigate through the files and select your project folder.



PlatformIO will open all the files within the project folder.

VS Code Color Themes

VS Code lets you choose between different color themes. Go to the **Manage** icon and select **Color Theme**. You can then select from several different light and dark themes.



Shortcuts' List

For a complete list of VS Code shortcuts for Windows, Mac OS X or Linux, check the next link:

- [VS Code Keyboard Shortcuts Reference](#).

Wrapping Up

In this tutorial you've learned how to install and prepare Visual Studio Code to work with the ESP32 and ESP8266 boards. VS Code with the PlatformIO IDE extension is a great alternative to the classical Arduino IDE, especially when you're working on more advanced sketches for larger applications.

Here's some of the advantages of using VS Code with PlatformIO IDE over Arduino IDE:

- It detects the COM port your board is connected to automatically;
- VS Code IntelliSense: Auto-Complete. IntelliSense code completion tries to guess what you want to write, displaying the different possibilities and provides insight into the parameters that a function may expect;
- Error Highlights: VS Code + PIO underlines errors in your code before compiling;
- Multiple open tabs: you can have several code tabs open at once;
- You can hide certain parts of the code;
- Advanced code navigation;
- And much more...

If you're looking for a more advanced IDE to write your applications for the ESP32 and ESP8266 boards, VS Code with the PlatformIO IDE extension is a great option.

We hope you've found this tutorial useful. If you like ESP32 and ESP8266, check the following resources:

