Utilizando o Arduino como controlador PID para o EV3 LEGO

Ariel Lima Andrade

Fevereiro de 2019

Resumo

Este projeto é uma integração do Arduino (plataforma de prototipagem eletrônica de hardware livre) com o Lego Mindstorms EV3 (plataforma de programação para desenvolvimento de robôs LEGO). O EV3 possui a interface de comunicação I2C, assim como o Arduino, possibilitando assim a integração destas plataformas. Neste projeto, o Arduino tem a função de um controlador PID, com o objetivo de controlar o posicionamento do eixo de um motor, no qual estarão acopladas vigas para sustentar algum objeto de forma que o eixo sempre esteja na posição desejada independentemente da massa do objeto (respeitando as limitações do motor).
Palavras-chave: PID, I2C, Arduino, LEGO EV3.

1 Introdução

A programação do Lego Mindstorms EV3 é feita em blocos, por ser uma linguagem de alto nível, deixa a desejar quando se trata de algoritmos mais complexos e projetos grandes (difícil visualização e compreensão do código). Com isso, a integração com o Arduino permite com que ampliemos a gama de ferramentas para os projetos de robótica com LEGO. Utilizando a comunicação I2C, esta comunicação se torna possível.
Vantagens:

- Utilização livre de sensores/atuadores;
- Aumenta o limite de sensores/atuadores no projeto (EV3 só permite 4 sensores e 4 atuadores);
- Programação em C/C++;
- Algoritmos mais extensos e complexos são viáveis;
- Utilização de 2 microcontroladores;

Desvantagens:

- Por utilizar 2 microcontroladores, requer mais espaço físico;
- Não permite a utilização de sensores I2C no Arduino;
- Requer um gerenciamento de processamento eficiente;

2 Comunicação I2C

I2C (Inter-Integrated Circuit) é um protocolo de comunicação serial síncrona para conectar dispositivos de baixa velocidade, como microcontroladores, EEPROMs, conversores AD e DE, interfaces I/O e outros periféricos em sistemas embarcados. O protocolo I2C é popular por ser simples de ser usado [1].
O barramento I2C é composto de dois fios, SDA e SCL, e alimentação (VDD), típicamente de 3.3V ou 5V. Os fios de comunicação possuem pull-ups, como pode ser visto na figura abaixo:

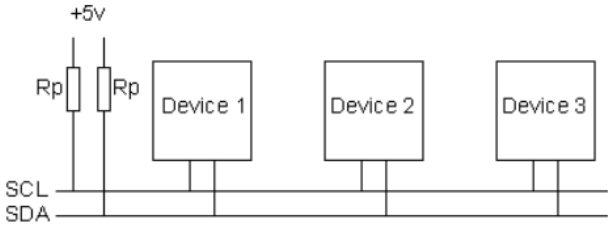

Figura 1: Circuito I2C

O número de nós em um único barramento é limitado tanto pelo tamanho do endereço, que pode ser de 7 bits, 10 bits e até 16 bits; como por restrição de espaço, já que não se pode ultrapassar poucos metros de fios, pois a capacitância total máxima, algo em torno de 400pf, impede o funcionamento correto do barramento.
O protocolo I2C tem dois tipos de dispositivos: Master e Slave. Onde o Master (mestre em inglês), é a unidade de controle responsável por coordenar todos os periféricos (slaves, escravos em inglês). A linha SCL é responsável pelo clock do barramento, e a linha SDA pela transmissão de dados.

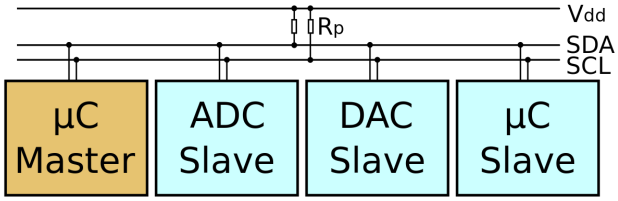

Figura 2: Barramento I2C

Como se pode perceber, no estado neutro do barramento I2C são mantidos o valor digital alto em ambas as linhas de comunicação, para se iniciar a comunicação, SDA é trazido para o valor digital baixo pelo mestre. Para escrever dados no barramento, SCL pulsa, e a cada pulso, o valor em SDA é lido como um bit, começando do bit mais significativo.
Logo após SDA ser trazida pra baixo, o mestre escreve o endereço do dispositivo que ele deseja se comunicar, por exemplo 0xC0, caso o dispositivo exista, ele responderá como um ACK, um pulso na linha SCL. Então começa a transferência de dados, o mestre escreve o endereço do registrador no escravo que ele deseja ler ou escrever (R/W) e opera então, em

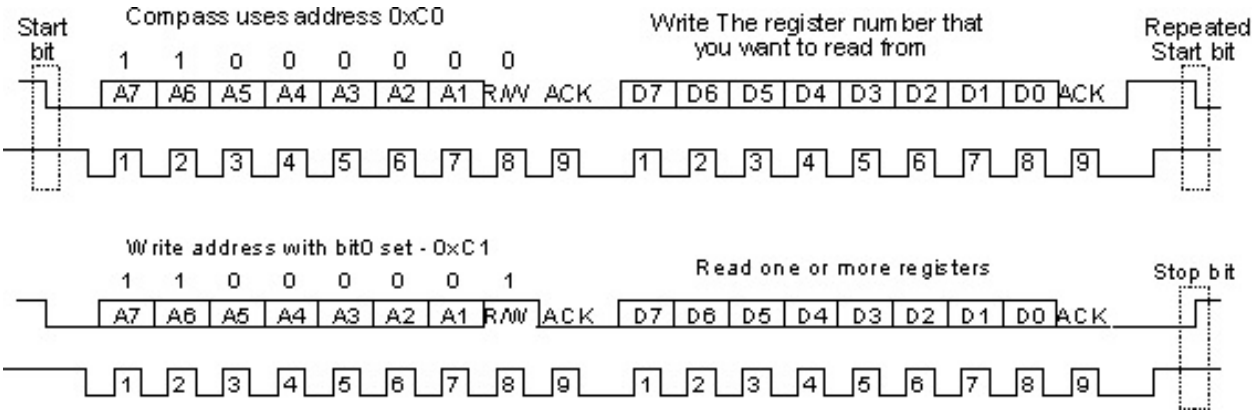

Figura 3: Funcionamento do protocolo

sequência, podendo ler/escrever um ou mais registrador [2].
Para utilizar a comunicação via I2C no Arduino, é necessário utilizar a biblioteca Wire. Os barramentos SDA (linha de dados) e SCL (linha de clock) podem ser encontrados nas versões UNO e Nano, nas portas A4 e A5, respectivamente. No Lego Mindstorms EV3, é preciso importar um bloco de comunicação I2C para o software de programação.

3 Controle PID

Proporcional-Integral-Derivativo (PID) é o algoritmo de controle mais usado na indústria e tem sido utilizado em todo o mundo para sistemas de controle industrial. A popularidade de controladores PID pode ser atribuída em parte ao seu desempenho robusto em uma ampla gama de condições de funcionamento e em parte à sua simplicidade funcional, que permite aos engenheiros operá-los de uma forma simples e direta.
Como o nome sugere, o algoritmo PID é composto por três coeficientes: proporcional, integral e derivativo, que são variados para obter a resposta ideal. A ideia básica por trás de um controlador PID é ler um sensor, calcular a resposta de saída do atuador através do cálculo proporcional, integral e derivativo e então somar os três componentes para calcular a saída.

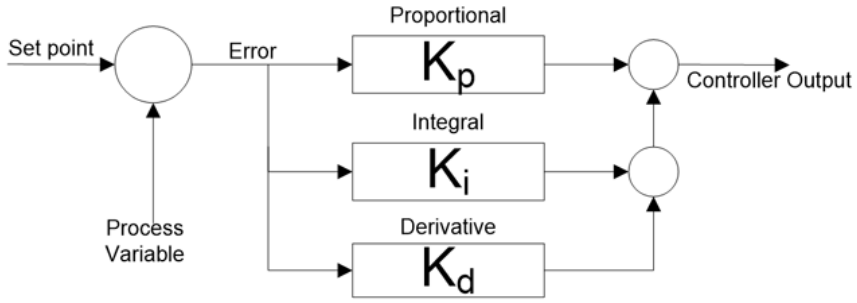

Figura 4: Diagrama de um algoritimo de controle PID

3.1 Resposta Proporcional

A componente proporcional depende apenas da diferença entre o ponto de ajuste e a variável de processo. Esta diferença é referida como o termo de erro. O ganho proporcional (Kc) determina a taxa de resposta de saída para o sinal de erro. Por exemplo, se o termo de erro tem uma magnitude de 10, um ganho proporcional de 5 produziria uma resposta proporcional de 50. Em geral, aumentando o ganho proporcional irá aumentar a velocidade da resposta do sistema de controle. No entanto, se o ganho proporcional é muito grande, a variável de processo começará a oscilar. Se Kc é aumentado ainda mais, as oscilações ficarão maior e o sistema ficará instável e poderá oscilar até mesmo fora de controle.

3.2 Resposta Integral

A componente integral soma o termo de erro ao longo do tempo. O resultado é que mesmo um pequeno erro fará com que a componente integral aumente lentamente. A resposta integral irá aumentando ao longo do tempo a menos que o erro seja zero, portanto, o efeito é o de conduzir o erro de estado estacionário para zero. O Steady-State de erro é a diferença final entre as variáveis do processo e do set point. Um fenômeno chamado windup integral ocorre quando a ação integral satura um controlador, sem que o controlador ajuste o sinal dê erro para zero.

3.3 Derivada de Resposta

A componente derivada faz com que a saída diminua se a variável de processo está aumentando rapidamente. A derivada de resposta é proporcional à taxa de variação da variável de processo. Aumentar o parâmetro do tempo derivativo (Td) fará com que o sistema de controle reaja mais fortemente à mudanças no parâmetro de erro aumentando a velocidade da resposta global de controle do sistema. Na prática, a maioria dos sistemas de controle utilizam o tempo derivativo (Td) muito pequeno, pois a derivada de resposta é muito sensível ao ruído no sinal da variável de processo. Se o sinal de feedback do sensor é ruidoso ou se a taxa de malha de controle é muito lenta, a derivada de resposta pode tornar o sistema de controle instável [3].

4 Materiais utilizados

Hardware

- 1x Arduino (pode ser qualquer versão, porém nesse projeto foi utilizado o Nano);
- 1x Lego Mindstorms EV3 + motor (com feedback do encoder) + estrutura + 2 botões;
- 1x Protoboard;
- 1x Display LCD 16x2;
- 1x Placa ilhada para circuitos;

- 1x Borne para eletrônica com 4 entradas;
- 1x Barra de pinos com 4 pinos;

Software

- EV3 Mindstorms - Software de programação;
- Arduino IDE;
- Bibliotecas para Arduino: Wire, LiquidCrystal e PIDLibrary (Brett Beauregard);
- Blocos I2C para o Mindstorms;

5 Montagem do circuito

5.1 Circuito da comunicação I2C

Para a conexão do EV3 com o Arduino, foi feita uma placa de circuito para facilitar a conversão do cabo do EV3 para a barra de pinos, assim possibilitando a conexão com o Arduino no barramento I2C. Nesse caso, não foram utilizados resistores de pull-up, porém caso a comunicação I2C apresente falhas, é necessário adicionar resistores de pull-up de 47k nas linhas SDA e SCL, conforme a figura abaixo:

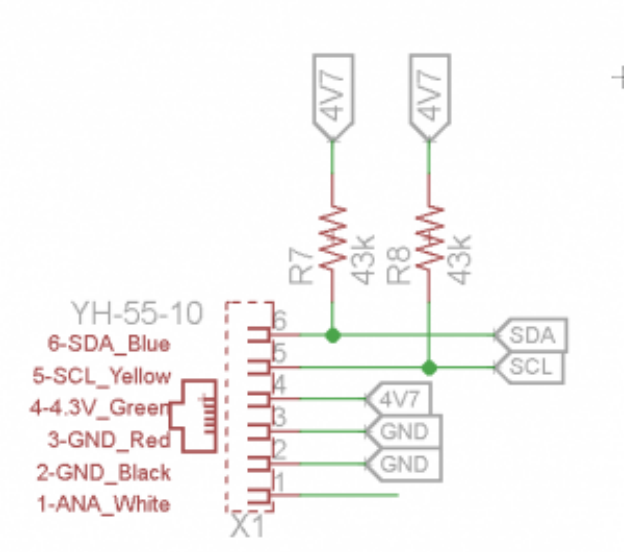

Figura 5: Conexão I2C utilizando resistores pull-up

Utilizando a placa ilhada para circuitos, o borne com 4 entradas e a barra de 4 pinos, constrói-se o circuito da figura abaixo. Dentro do cabo do EV3 há 6 fios, não utiliza-se o fio branco. Os fios vermelho e preto são ligados ao GND, o fio verde é ligado ao VCC (5V), o fio azul é conectado no SDA (A4) e o fio amarelo no SCL (A5).

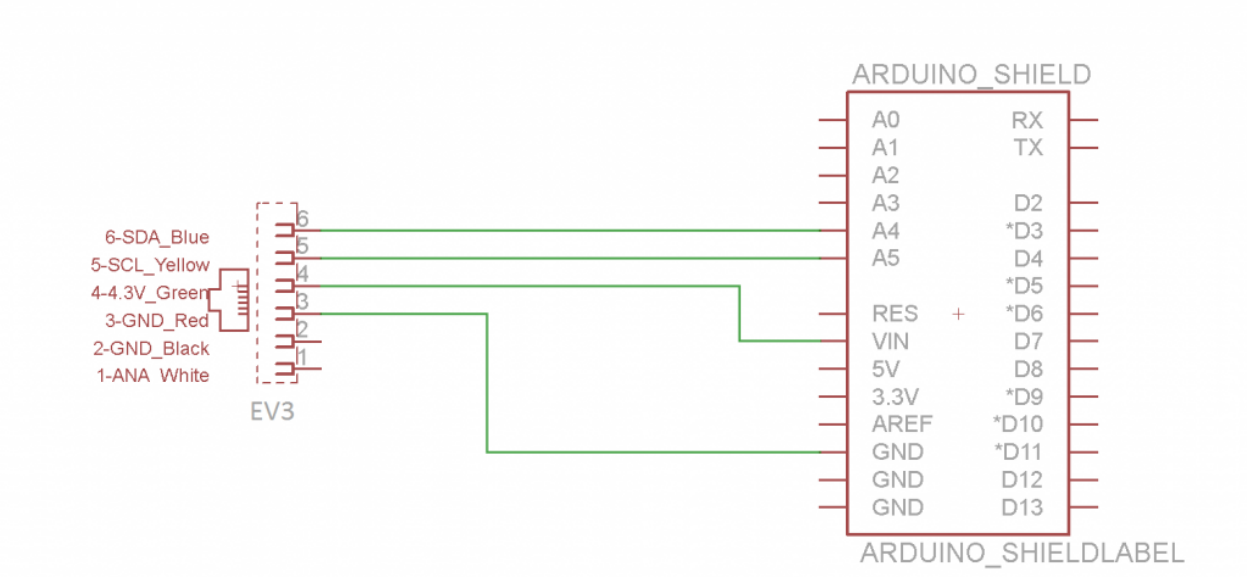

Figura 6: Conexão com o Arduino sem resistores pull-up

5.2 Circuito display LCD 16x2

O display LCD 16x2 utiliza o controlador HD44780, que permite a conexão com o vários microcontroladores, inclusive o Arduino. A conexão foi feita conforme a tabela abaixo.

Conexões LCD - HD44780	
Pino LCD	Pino Arduino
1 - Vss	GND
2 - Vdd	Vcc 5V
3 - V0	GND
4 - RS	Porta 12
5 - RW	GND
6 - E	Porta 11 Arduino
7 - D0	Não conectado
8 - D1	Não conectado
9 - D2	Não conectado
10 - D3	Não conectado
11 - D4	Porta 5 Arduino
12 - D5	Porta 4 Arduino
13 - D6	Porta 3 Arduino
14 - D7	Porta 2 Arduino
15* - A	Vcc 5V
16 - K	GND

* Resistor de 360 ohm para limitar corrente.

A imagem abaixo mostra o circuito final:

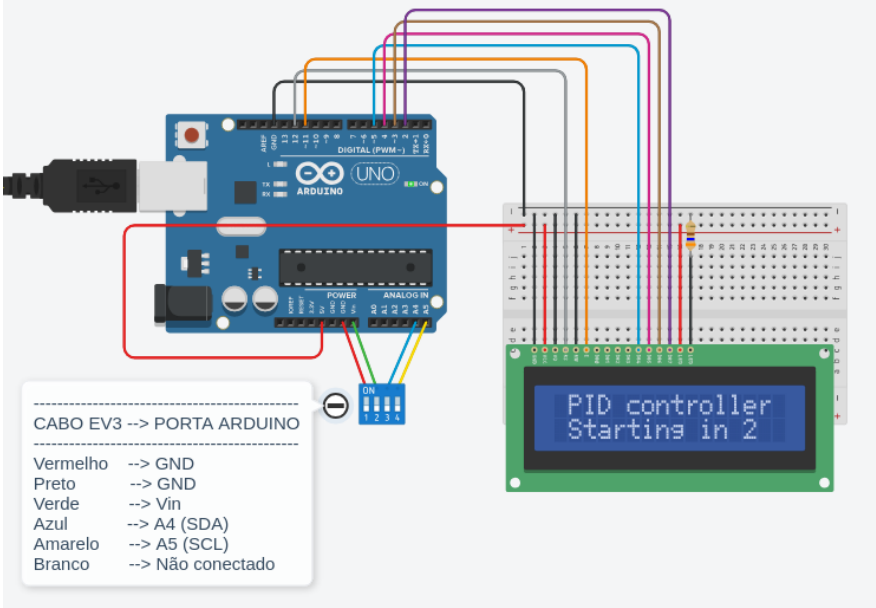

Figura 7: Circuito Arduino

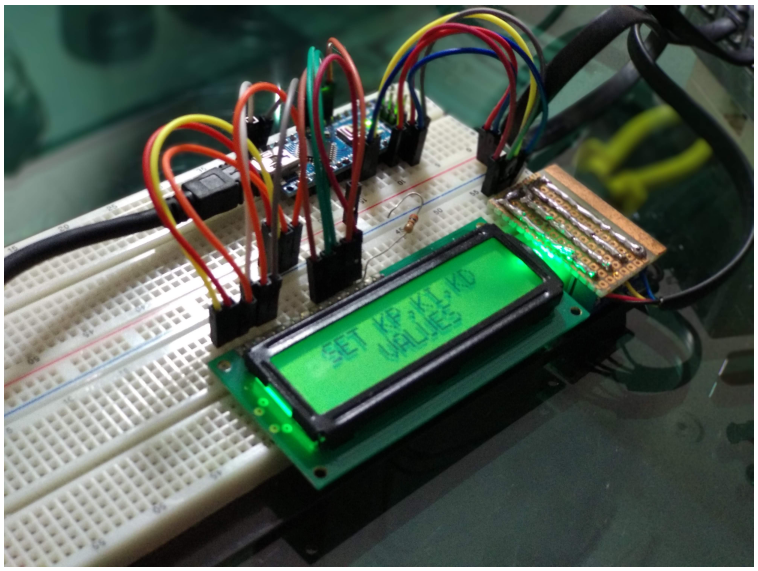

Figura 8: Circuito físico na protoboard

5.3 Conexão dos dispositivos no EV3

Foram utilizados no EV3 dois botões (sensores de toque) e um motor (com feedback). Os dois botões foram conectados nas portas 2 e 3, o cabo que conecta ao Arduino via I2C deve ser conectado na porta 1 e motor na porta de saída A. A imagem abaixo mostra o bloco EV3 com seus dispositivos e estrutura feita com peças LEGO.

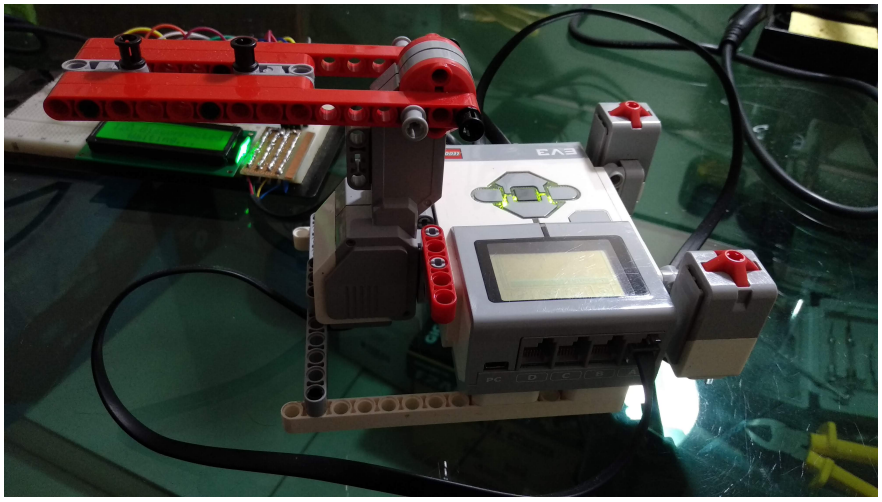

Figura 9: Dispositivos no EV3 e estrutura

6 Programação

6.1 Mindstorms EV3

Para a utilização da comunicação I2C no Mindstorms EV3, é necessário instalar as bibliotecas (blocos) "Dexter.ev3b"e "Mindsensors.ev3b", disponíveis em https://github.com/arielima97/PID-controller-for-EV3-with-arduino/tree/master/EV3/libraries. Estas bibliotecas devem ser importadas no software de programação Mindstorms EV3.

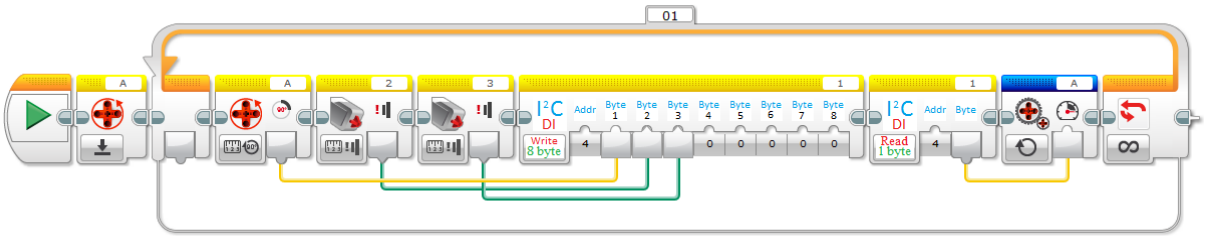

Figura 10: Programação do EV3

A figura acima mostra a programação do EV3, que funciona da seguinte forma: inicialmente a posição desejada do eixo acoplado ao motor é determinada (reiniciando o enconder do motor, definindo o ponto atual para de seja dado um feedback igual a 0), após essa definição, o programa entra em um loop infinito em que coleta o a posição do motor em graus e o estado dos botões (se está pressionado ou não) e envia essas informações através do bloco I2C na função "Write 8 byte", a qual permite o envio de até 8 bytes para o dispositivo desejado (no caso o Arduino). O primeiro parâmetro deste bloco é o endereço do dispositivo em que se quer enviar a informação, neste caso definimos o Arduino com o endereço 0x04, os demais campos são os bytes a serem enviados. Neste caso é necessário somente os três

primeiros bytes. Em seguida, utilizando o mesmo bloco I2C, porém com a função "Read 1 byte"(solicita ao Arduino um byte e lê esse valor), lemos o valor enviado pelo Arduino que corresponde à resposta do algoritmo PID e aplicamos este valor diretamente num bloco para acionar o motor com uma força proporcional à resposta obtida pelo Arduino.

6.2 Programação Arduino

Nas linhas de 1 a 3 do código abaixo, inserimos as bibliotecas. As bibliotecas Wire e LiquidCrystal já estão disponíveis no software Arduino IDE e a biblioteca PID está disponível em https://github.com/arielima97/PID-controller-for-EV3-with-arduino/tree/master/arduino/libraries. Nas linhas 6 e 7, determinamos o endereço do Arduino para a comunicação I2C. Nas linhas de 11 a 17, criamos as variáveis dos parâmetros para o algoritmo PID e na linha 19 instanciamos um objeto PID com todos estes parâmetros. Na linha 23 instanciamos um objeto display LCD onde entramos com os pinos em que conectamos o LCD no Arduino (seguindo a sequência: RS, E, D4, D5, D6 e D7). Nas linhas de 26 a 29, criamos variável para receber o valor do feedback do motor, uma flag para identificar se as constantes do PID já foram inseridas pelo usuário, um vetor contendo os dígitos determinado pelo usuário no display LCD e as variáveis de estado dos botões do EV3. Na linhas de 26 a 35 determinados algumas funções que serão utilizadas:

- Linha 31 - DTF: Função que recebe um vetor com os dígitos selecionados pelo usuário no display LCD e converte para um número em ponto flutuante.
- Linha 32 - set_const: Função que executa a interface para aquisição das constantes KP, KI e KD determinadas pelo usuário através do display LCD e botões do EV3.
- Linha 33 - update_const: Função que atualiza o valor atual da constante exibida na tela. Utilizada após o usuário pressionar um botão.
- Linhas 34 e 35 - check_bX: Checa se o botão X (1 ou 2) foi pressionado.

```
1  #include <PID_v1.h>
2  #include <Wire.h>
3  #include <LiquidCrystal.h>
4
5  //-----------I2C PARAMETERS-------------
6  #define I2C_ADDRESS 0x04 //Define I2C address.
7  bool I2C_connected = 0; //Check if I2C is connect.
8  //---------------------------------------
9
10 //-----------PID PARAMETERS-------------
11 double KP = 0.0;   //Proportional constant
12 double KI = 0.0;       //Integral constant
13 double KD = 0.0;      //Derivative constant
14
15 double Setpoint = 0; //Reference position. PID controls the motor ↩
      to reach this position.
```

```
16 double Input;        //Motor encoder feedback.
17 double Output;       //Motor power.
18
19 PID myPID(&Input, &Output, &Setpoint, KP, KI, KD, DIRECT); //PID ↩
      library. It sets the Output variable based on the error (↩
      Setpoint - Input);
20 //---------------------------------------
21
22 //-----------LCD PARAMETERS-------------
23 LiquidCrystal LCD(12, 11, 5, 4, 3, 2); //Display LCD library.
24 //---------------------------------------
25
26 int encoder_feedback = 0; //Motor encoder feedback.
27 bool flag = 0; //Flag to verify whether the constants were defined ↩
      or not.
28 int dig[6] = {}; //Digits on LCD screen.
29 byte b1, b2; //Buttons status.
30
31 float DTF(int *v); //Digits on LCD to float number.
32 void set_const(int *skip); //Setting KP, KI and KD, input by the ↩
      user.
33 void update_const(int n); //Update the screen with the actual ↩
      constant value.
34 void check_b1(); //Check if button 1 was pressed.
35 void check_b2(); //Check if button 2 was pressed.
```

No trecho do código abaixo descrevemos a função setup(), onde iniciamos o display LCD, e a comunicação I2C com suas determinadas funções que dever ser executadas quando um dado é recebido e solicitado. Em seguida é iniciada uma interface para o usuário sinalizando que o controlador está iniciando e verificando se o cabo de conexão do Arduino com o EV3 está conectado. Iniciamos também a porta serial e colocamos o PID em modo automático e definimos os limites da sua resposta (de -100 a 100 pois esses são os valores limites para força do motor no EV3.

```
37 void setup()
38 {
39     LCD.begin(16, 2);                    //Setting up the LCD 16x2.
40     Wire.begin(I2C_ADDRESS);             //Defining Arduino I2C ↩
          address.
41     Wire.onReceive(readData);            //When receive data from ↩
          master, the function readData runs.
42     Wire.onRequest(writeData);           //When the master requests ↩
          data, the function writeData runs.
43
44     byte i = 0;                         //Set starting time, in ↩
          seconds.
45     for(i; i > 0; i--)
```

```
46      {
47          LCD.setCursor(0, 0);
48          LCD.print(" PID controller ");
49          LCD.setCursor(0, 1);
50          LCD.print(" Starting in");
51          LCD.setCursor(13,1);
52          LCD.print(i);
53          delay(1000);
54          LCD.clear();
55      }
56
57      if(!I2C_connected)                          //Checking if I2C is ↩
            connected.
58      {
59          while(I2C_connected == false)
60          {
61              LCD.setCursor(0, 0);
62              LCD.print("I2C disconnected");
63              LCD.setCursor(0, 1);
64              LCD.print("   Waiting...   ");
65          }
66          LCD.clear();
67          I2C_connected = true;
68      }
69      else
70          I2C_connected = true;
71
72      Serial.begin(9600);                 //Starting Serial Monitor.
73      myPID.SetMode(AUTOMATIC);           //Starting PID control.
74      myPID.SetOutputLimits(-100, 100); //Setting limits to PID ↩
            Output (from -100 to 100, because that's the limit in EV3).
75  }
```

O trecho do código mostrado abaixo é a função loop(), onde em sua primeira execução é iniciada a interface com o usuário para que este possa inserir os parâmetros KP, KI e KD. Após a definição dos parâmetros, o algoritmo PID é iniciado e executado em tempo real e o display LCD exibe a resposta do controle PID e a posição do eixo do motor.

```
78  void loop() {
79      set_const(&flag); //Setting constants.
80      myPID.SetTunings(KP, KI, KD, P_ON_E); //Applying the new ↩
            constants to PID algorithm.
81      Input = encoder_feedback; //Assigning motor encoder feedback to↩
             PID Input.
82      myPID.Compute(); //Computing Output based on the error.
83
84
```

```
85      //Shows the position and the output on the screen.
86      Serial.println("Running");
87      LCD.setCursor(0, 0);
88      LCD.print("Power: ");
89      LCD.setCursor(11, 0);
90      LCD.print(Output);
91      LCD.setCursor(0, 1);
92      LCD.print("Position: ");
93      LCD.setCursor(11, 1);
94      LCD.print((float)encoder_feedback);
95  }
```

As funções das linhas 31 a 35 não serão descritas pois são apenas funções de interface com o usuário, o que não é o objetivo principal deste projeto.

7 Funcionamento

Inicialmente o programa verifica se a conexão I2C foi estabelecida, ou seja, se o cabo foi conectado ao EV3 e a programação no EV3 foi executada. Após a confirmação da conexão, pressionar o botão 1 do EV3 (botão da porta 2 do EV3) para iniciar a seleção do parâmetro KP, o botão 1 altera o número da correspondente casa decimal sinalizada, para confirmar o número pressionar o botão 2 (botão da porta 3 do EV3). Os mesmos procedimentos para a seleção dos parâmetros KI e KD. Ao finalizar o parâmetro KD, o PID já entra em execução controlando o eixo do motor conforme os parâmetro utilizados. Nota-se que a posição do eixo é determinada pela posição em que o eixo estava quando a programação do EV3 foi iniciada, ou seja, para alterar este ponto, basta sair da programação do EV3, mover o eixo para a posição desejada e assim iniciar a programação do EV3 novamente. Para alterar os parâmetros do PID novamente, basta pressionar o botão de reset do Arduino e parametrizar novamente.

8 Conclusão

Este projeto é uma demonstração de como se pode integrar o Arduino com o EV3 LEGO através da comunicação I2C. Neste projeto o Arduino atuou como um controlador PID para o EV3, esta ferramenta pode ser utilizada em aplicações como robôs seguidores de linha, movimentação e orientação em robótica ou robôs equilibristas. A utilização de um Arduino junto com o EV3 possibilita a criação de estruturas mecânicas de forma simples utilizando as peças LEGO, o desenvolvimento de algoritmos mais complexos, e a expansão na quantidade e tipos de sensores/atuadores suportados pelo EV3, pois podemos usar qualquer dispositivo que se conecte ao Arduino. Com isso, até mesmo projetos envolvendo IoT com o EV3 são possíveis através da integração Arduino/EV3.

Referências

[1] *I2C Bus, Interface and Protocol*, disponível em `https://i2c.info/`.

[2] *Protocolo I2C*, disponível em `http://www.univasf.edu.br/řomulo.camara/novo/wp-content/uploads/2013/11/barramento-e-Protocolo-I2C.pdf`.

[3] *Explicando a Teoria PID*, disponível em `http://www.ni.com/pt-br/innovations/white-papers/06/pid-theory-explained.html`