MIEEC

SBMI

Braço de Robô em configuração *Palletizer*

2020/2021

André Enes, up201806669

Pedro Ramadas, up201806142

Introdução

O projeto que escolhemos foi o "Braço de Robô em configuração "*Palletizer*", que consiste num dispositivo que organiza 3 objetos de acordo com a sua cor. O braço recolhe uma peça, de seguida leva a peça a um sensor que após determinar a sua cor, deixa a peça no sítio correspondente. Possui igualmente um LCD que permite ao utilizador verificar a cor analisada, bem como a tarefa que o utilizador está a realizar. As posições das peças, bem como a posição do sensor e das cores podem ser definidas através de três botões. Depois da primeira configuração, podemos voltar a usar os valores das posições configurados anteriormente, dado que estão guardados na memória não volátil do microprocessador.

Materiais utilizados

- Sensor de cor TCS3200;
- 4 servos SG90;
- LCD 16x02;
- Braço robótico (https://www.thingiverse.com/thing:3117691);
- 3 botões;
- *Breadboard;*
- Cabos de ligação;
- Atmega328p incluído na placa Arduino Uno;
- 4 transístores NPN C0913;
- Resistências;

Componentes

Servo motores

Segundo a *datasheet* dos SG90, o ângulo para qual o motor se posiciona é controlado através de um certo *duty-cycle* de uma onda PWM de 50Hz, ou seja, se for transmitido uma onda PWM de 50 Hz com um *duty-cycle* de 1,5 ms, o motor move-se para a posição 0°.

Para se obter um período de 20 ms em modo PWM, é necessário utilizar o timer 1, sendo este de 16 bits. Escolhemos utilizar o modo *Fast PWM*, ao invés do *Phase Correct PWM*, uma vez que a implementação é mais simples e o *Phase Correct PWM* é mais indicado para motores onde é necessário controlar a velocidade do mesmo, ao contrário dos servo motores que a onda PWM só indica a posição para onde o motor deve rodar.

O facto de haver 4 motores implica um obstáculo: só existem duas saídas associadas à geração de ondas PWM pelo timer 1. Para superar o desafio, utilizamos 4 transístores NPN C0913, ligando 1 saída à base de 2 transístores com uma resistência de 500Ω, de forma a que quando as saídas PB4 e PB5 são acionadas, passe uma corrente na base I_B = 50mA, fazendo que o

transístor, segundo a *datasheet*, entre na zona de saturação e funcione como um curto circuito entre o motor e a saída que transmite a onda PWM.

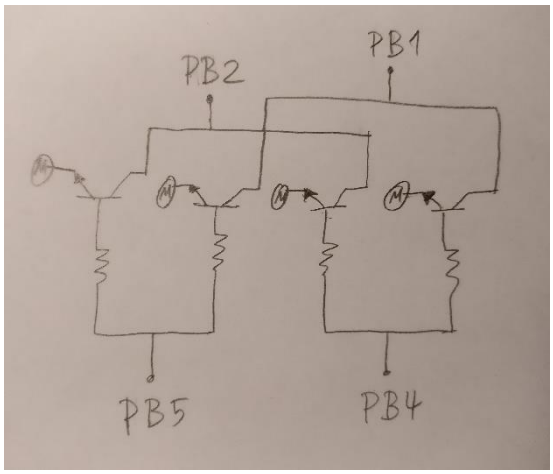


Figura 1 – esquema de ligação dos servo motores e transístores

Esta disposição torna o programa mais complexo, uma vez que é necessário desligar a função de PWM quando se troca de linha do transístor, para o motor que não se pretende controlar, não rode. A vantagem desta montagem é que se usam apenas 4 saídas, enquanto que se usássemos 1 saída por transístor, seriam sempre, no mínimo, utilizadas 5 saídas. Existia a alternativa de ligar 1 saída por transístor e as duas saídas de PWM do timer 1, mas isso ocuparia 6 saídas e tendo em conta que, neste projeto, é necessário o sensor de cor e o lcd (sem i2c, no nosso caso), decidimos poupar saídas para utilizar noutros componentes.

```
uint8_t garra_roda(int8_t pos) ///////////////rotação de baixo
{
    int angle = 0;
    angle = (200 / 9) * pos + 2999.5;
    if ((((!t_check_roda) && (!t_check_sobe) && (!t_check_estica) && (!t_check_trinca)
))
    {
        tim2_servo_inif();
        t_check_roda = 1;
    }

    if (tim2_servo_checkf() && t_check_roda == 1)
    {
        TCCR1A &= ~(1 << COM1B1); //ao mudar a linha dos transistores, se não fizesse
 isto, o outro motor mexia se
        TCCR1A |= (1 << COM1A1);

        servo_deg(angle, 1);
        t_check_roda = 0;
        return 1;
    }
    return 0;}
void servo_deg(int16_t angle, uint8_t n)
```

```
{
    if (n == 1)
    {
        PORTB &= ~(1 << PB5); //desliga linha dos servo3 e 4
        PORTB |= (1 << PB4);  //liga linha dos servo1 e 2
        OCR1A = angle;
    }
    else if (n == 2)
    {
        PORTB &= ~(1 << PB5);
        PORTB |= (1 << PB4);
        OCR1B = angle;
    }
    else if (n == 3)
    {
        PORTB &= ~(1 << PB4);
        PORTB |= (1 << PB5);
        OCR1A = angle;
    }
    else if (n == 4)
    {
        PORTB &= ~(1 << PB4);
        PORTB |= (1 << PB5);
        OCR1B = angle;
    }
}
```

A função garra_roda() controla o servo que roda a plataforma de baixo, sendo que para os outros servos, existem funções que os controlam. Para tal, converte o valor dado para um número que indique o *duty cycle* correto para a onda *PWM*, verifica que não existem outros servo motores em movimento, espera um determinado tempo definido pela função tim2_servo_inif() (neste caso, 1,5 s), desativa o modo *PWM* para a linha que não está em uso (se este passo não fosse realizado, o outro motor que esteja na mesma linha do transístor poderá se deslocar, sem que seja dada indicação) e chama a função servo_deg(). Já a servo_deg() seleciona a linha correta dos transístores e coloca o *duty cycle* correto na saída PWM que, por sua vez, roda o servo motor para o ângulo pretendido.

```
void vai_coordenadas(uint8_t roda, uint8_t sobe, uint8_t estica, uint8_t trinca)
{
    if (!pos1)
    {
        garra_roda(roda);
        if (t_check_roda == 0)
        {
            pos1 = 1;
        }
    }
    else if (!pos2 && pos1)
    {
        garra_sobe(sobe);
        if (t_check_sobe == 0)
        {
            pos2 = 1;
        }
    }
    else if (!pos3 && pos2)
    {
        garra_estica(estica);
        if (t_check_estica == 0)
        {
            pos3 = 1;
        }
    }
    else if (!pos4 && pos3)
    {
        if (trinca != 91)
            garra_trinca(trinca);
        if (t_check_trinca == 0 || trinca == 91)
        {
            pos4 = 1;
        }
    }
}
```

Para o braço se dirigir até uma certa posição, é utilizada a função acima. Nos parâmetros são enviadas as coordenadas criadas por nós, que consistem em ângulos da base, do mecanismo que faz subir o braço, do que o faz esticar e o de fechar a garra (trincar). Com esta informação, o braço usa outras funções já descritas para mover os motores para a posição correta, igualando as variáveis pos a 1 após cada motor tiver completado o movimento. Estas variáveis são utilizadas pela FSM[0] para verificar o fim dos movimentos. O número 91 é para esse motor não mudar de posição, sendo 91 um ângulo não aceitável. Também é usada uma função chamada regressa_coordenadas(), cujo propósito é movimentar o braço para a posição standard, através da sequência inversa de movimentos da vai_coordenadas().

EEPROM

```
/*  eeprom addresses

    0 --> sobe_sensor
    1 --> roda_sensor
    2 --> estica_sensor
    3 --> sensor_def
    4 --> sobe_pos1
    5 --> roda_pos1
    6 --> estica_pos1
    7 --> pos1_def
    8 --> sobe_pos2
    9 --> roda_pos2
    10 --> estica_pos2
    11 --> pos2_def
    12 --> sobe_pos3
    13 --> roda_pos3
    14 --> estica_pos3
    15 --> pos3_def
    16 --> sobe_red
    17 --> roda_red
    18 --> estica_red
    19 --> red_def
    20 --> sobe_green
    21 --> roda_green
    22 --> estica_green
    23 --> green_def
    24 --> sobe_blue
    25 --> roda_blue
    26 --> estica_blue
    27 --> blue_def
    28 --> tudo_definido
*/
```

A lista apresentada mostra as posições de memória que utilizámos para cada parâmetro guardado na memória *eeprom*: a posição do sensor de cor, a posição da 1ª, 2ª e 3ª peças, as posições do depósito das peças vermelha, verde e azul e se estas posições já se encontram na memória.

Sensor de cor

Para usar o sensor de cor previamente mencionado, foi necessário usar 8 das 10 entradas disponíveis. As entradas Vcc e Gnd são de alimentação, enquanto que as entradas S0 e S1 permitem definir o escalonamento da potência de saída e S2 e S3 definem a cor a ser analisada. Neste projeto, definimos S0 a “High” e S1 a “Low” pois foi onde obtivemos as melhores medições. Para medir as cores, recorremos ao timer2 e ao timer0 do microprocessador, no modo “External clock source on T0 pin. Clock on rising edge”, pelo que a saída out foi ligada à

porta T0 (PD4). Como o sensor de cor envia uma onda quadrada com uma frequência que depende da cor do objeto, é possível relacionar o valor de *Rising edges* obtidos num intervalo de tempo com o valor RGB da cor do objeto.

Assim, para verificarmos a cor de um objeto, colocamos o sensor no modo de leitura da cor verde (S2 e S3 a "High"), fazemos 10 medições do valor no counter 0, cada uma de 8ms em 8ms. Guardamos o maior valor obtido. Procedemos de igual modo para a cor azul e vermelha. No final, comparamos os valores máximos obtidos. O maior valor entre estes diz-nos a cor do objeto.

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

Figura 2 Entradas S2 e S3

S0	S1	OUTPUT FREQUENCY SCALING (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

Figura 3 Entradas S0 e S1

O excerto abaixo demonstra o código efetuarmos uma medição de uma cor:

```
void sensor_cor()
{
    ler_vermelho(); //Tal como mencionado anteriormente, é necessário medir o valor RGB de cada objeto, pelo que cada cor é medida individualmente
    while (aux2 != 2) ;//Quando acaba a medição da cor vermelha, começa a medição da cor verde
    ler_verde();
    while (aux2 != 3) ;
    ler_azul();
    while (aux2 != 0) ;
}
void ler_vermelho()
{
    PORTD &= ~(1 << PD3); //Configura o sensor para ler a cor vermelha
    PORTD &= ~(1 << PD5);
    aux2 = 1;
    aux = 0;
    while (aux != 10) ; //A esta variável é somado 1 a cada overflow do timer 0 (8 ms), e é guardado o valor máximo do timer 0 (neste caso, valor da cor vermelha) durante 10 ciclos do timer 1.
    if (aux == 10)
    {
        aux2 = 2; //Esta variável fica a 1 no fim das 10 medições
    }
}
```

Procedemos de igual modo para as cores verde e azul. No final, comparamos o valor máximo de cada cor e o máximo indica a cor do objeto.

LCD

Em relação ao LCD, utilizamos um LCD do tipo 16X02, ou seja, 16 colunas e 2 linhas. Para usarmos este componente, usamos os seguintes pins: D4, D5, D6, D7, RS e E0. Os pins de D4 a D7 são usados para trocar dados de 4 bits entre o microcontrolador e o LCD, o pin RS que indica se o microcontrolador está a enviar comandos ou se está a enviar dados, e por último E0 que permite ao LCD saber quando é que deve processar os dados recebidos nos pins previamente mencionados. Como o funcionamento deste componente pode ser bastante complicado, recorremos a uma biblioteca AVR. Esta inclui funções simples como limpar o ecrã, escolher a célula onde queremos escrever, entre outras. Em relação ao circuito usado, está incluído na secção "Anexos" a montagem do circuito. Todos estes pins foram ligados às portas analógicas do microprocessador por uma questão de organização.

Em relação ao código utilizado, existem 3 funções na biblioteca utilizada que simplificam o seu uso do LCD:

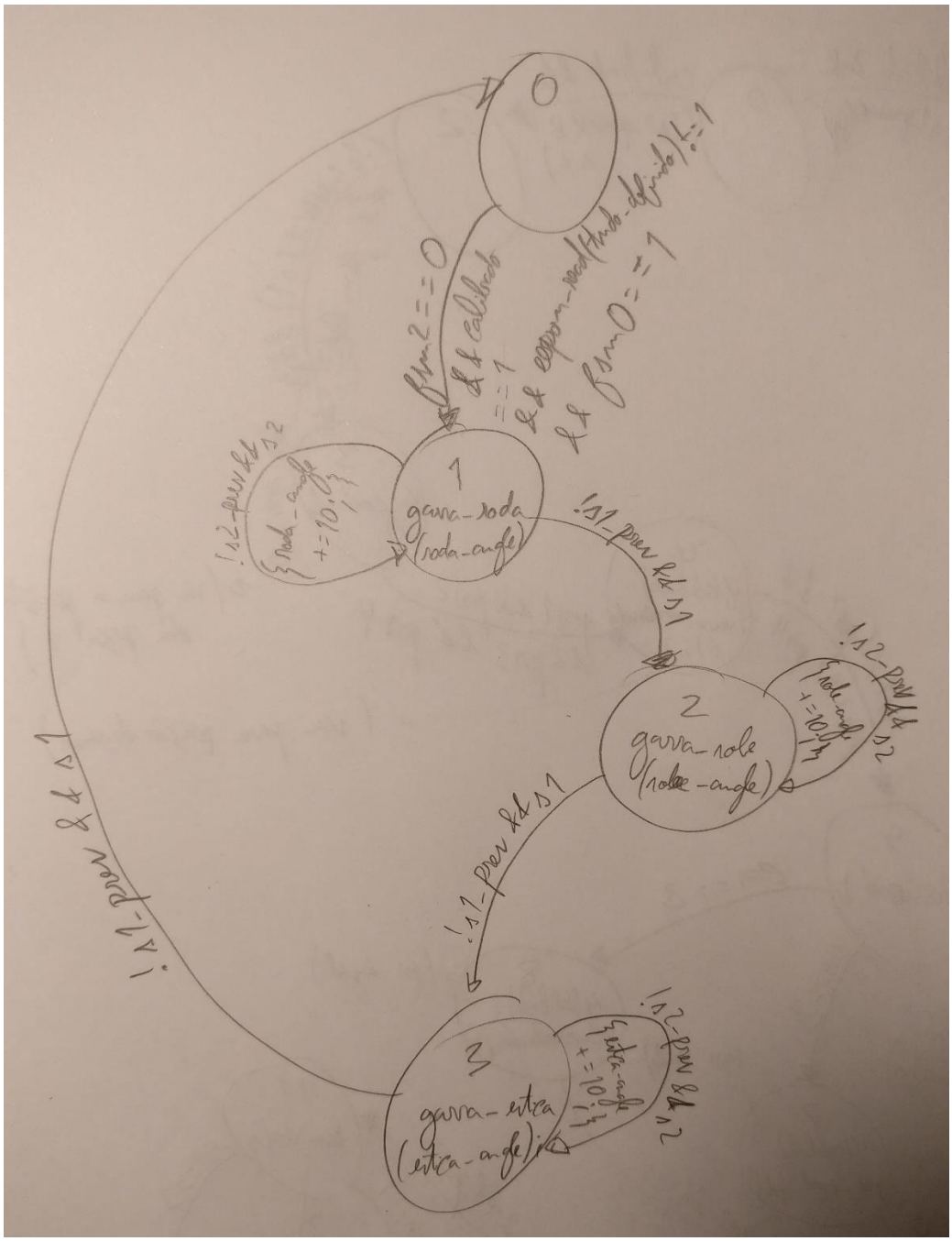
```
lcd_init();
lcd_xy(x, y);
lcd_puts("String");
```

A primeira função referida "limpa" o LCD, pelo que deve ser utilizada antes escrevermos algo lá.

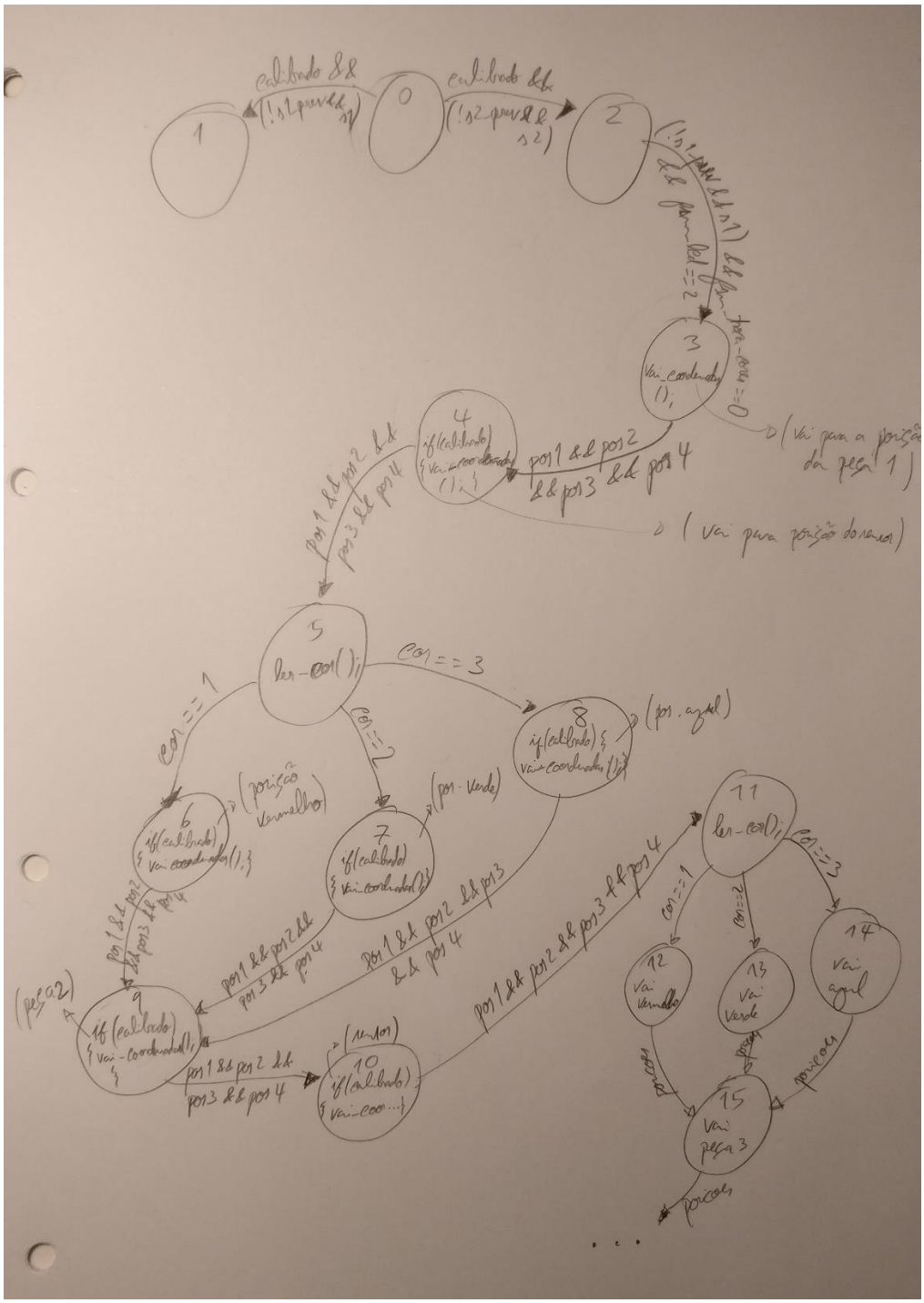
A segunda indica ao LCD em que linha e coluna é que queremos escrever o texto, sendo o 'x' a coluno e 'y' a linha.

Por último, a terceira função serve para enviarmos uma *string* para o LCD.

Máquinas de estado



Esta máquina de estados demonstra a calibração do braço. Quando a etapa 1 se encontra ativa, cada clique no botão S2 aumenta o valor do ângulo do servo da base em 10º, sendo que a gama de valores vai de -90º a 90º. Quando atinge o máximo, regressa ao valor mínimo. Quando estivermos satisfeitos com a posição do braço, carregamos no botão S1, e a etapa 2 fica ativa. Tal como na etapa anterior, cada clique no botão S2 aumenta o valor do ângulo em 10º, mas desta vez é o servo responsável pela altura do braço. A gama de valores vai de -25º a 90º, pois valores inferiores a -25 podem estragar o braço. Por último, quando o botão S1 é carregado, passamos à última calibração, neste caso do servo responsável por esticar o braço. A gama de valores deste servo é de -35º a 70º.



A máquina de estados acima representa o funcionamento geral do braço robótico, após a configuração de todas as posições. O braço inicia o programa numa posição neutra a que chamamos de posição standard. Se for pressionado o botão 1, o braço, por esta ordem, roda, sobe e estica até a peça 1, fecha a garra, retorna à posição standard pela ordem inversa do movimento anterior, vai até ao sensor (pela mesma ordem que foi até à peça 1), lê a cor e consoante o resultado, volta à posição standard e dirige-se para a posição da cor respetiva.

Procedimento

Inicialização

Quando ligamos o microprocessador, aparece-nos um menu que indica se queremos configurar as posições (do sensor, peças e para onde as queremos levar) ou se queremos usar posições previamente configuradas. Se queremos novas posições carregamos no botão 1, senão carregamos no botão 2.

Figura 4 - Primeiro menu do LCD

Ao configurar novas posições, o botão 2 controla o ângulo de cada servo (cada clique aumenta o valor do ângulo até 90º, caso exceda este valor, passa para -90º) e o botão 1 indica que o valor do ângulo já está definido e podemos passar ao próximo.

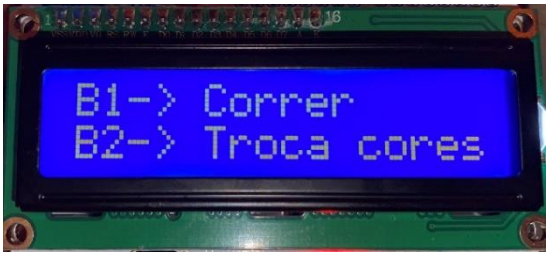


Figura 5 - Segundo menu do LCD

De seguida, depois de configurarmos todas as posições ou de termos escolhido a opção de memória, podemos trocar as posições de destino dos objetos, isto é, podemos: trocar a cor as cores entre si (a posição azul passa a ser a verde, e vice-versa, por exemplo) carregando no botão S2, começar a ordenar os objetos (botão S1), ou então efetuarmos uma medição de cor (botão S3). Neste último modo, cada clique no botão S3 efetua uma medição, e quando quisermos regressar ao menu anterior, é necessário pressionar o botão S1.

Após carregarmos no botão 1, o LCD indica que o programa está a correr. O braço começa numa posição central e vai à posição 1 buscar a primeira peça. De seguida retoma à posição inicial e leva a peça ao sensor de cor. Após ser determinada a cor, o braço vai para a posição central, e de seguida para a posição lida e larga a peça. Este procedimento é repetido para mais duas peças. No final, retoma para o segundo menu previamente descrito.

Bibliografia

1. Slides da cadeira “SBMI”
2. https://www.avrfreaks.net/forum/tutc-lcd-tutorial-1001 - Website onde se encontra um tutorial e a biblioteca usada para o LCD
3. https://www.mouser.com/catalog/specsheets/tcs3200-e11.pdf - Datasheet do sensor de cor
4. http://www.agspecinfo.com/pdfs/J/JHD162A.PDF - Datasheet do LCD
5. http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf - Datasheet dos servos

Autores

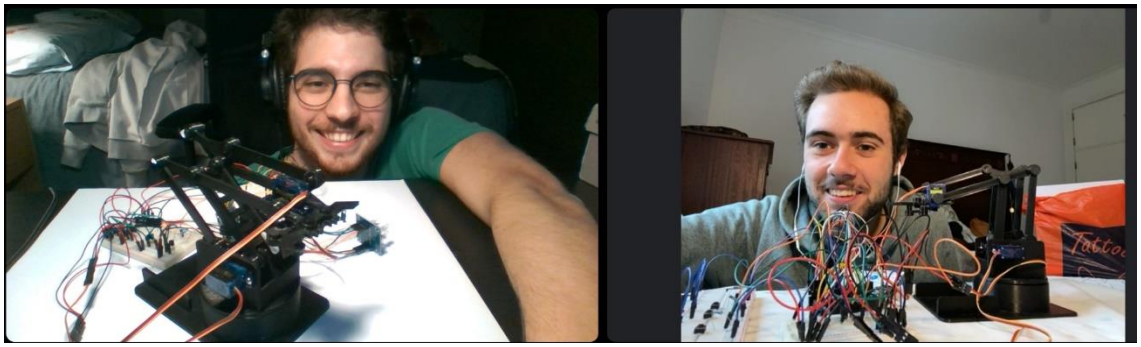


Figura 6 - André Enes e Pedro Ramadas

6. Anexos

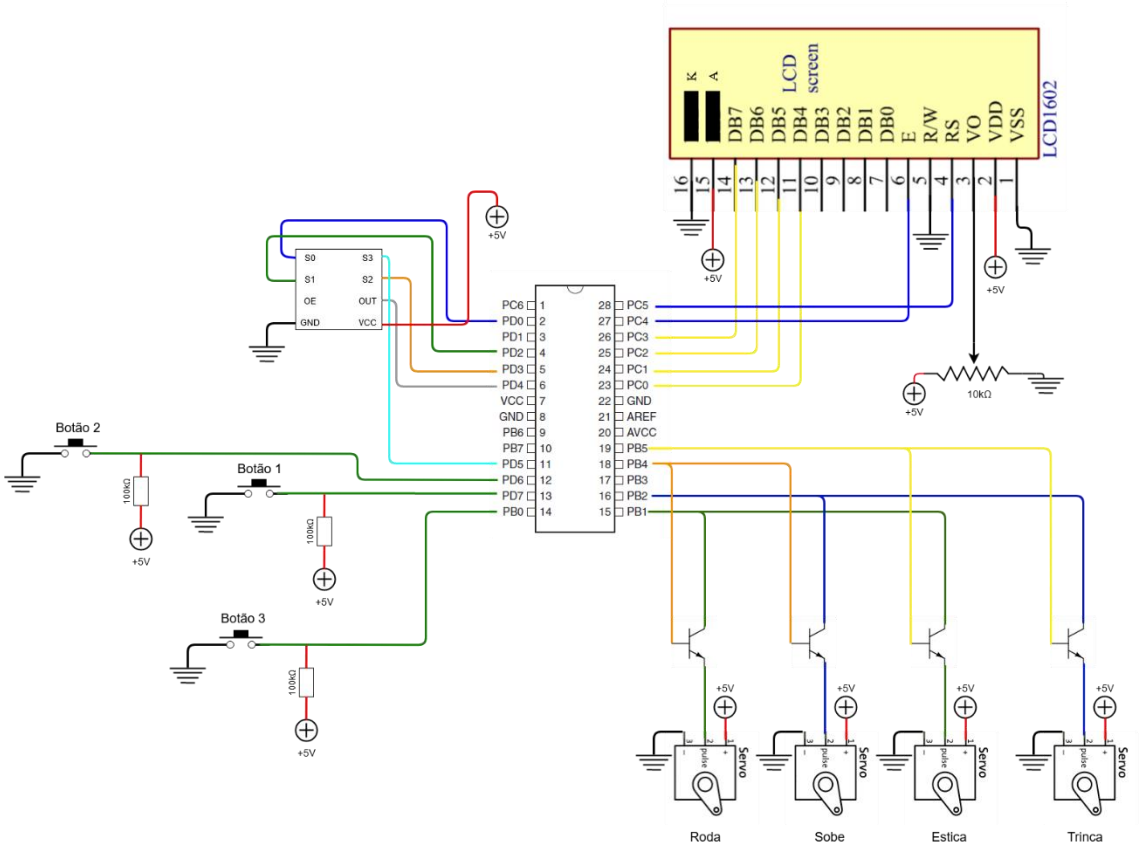


Figura 7 - Esquemático da montagem

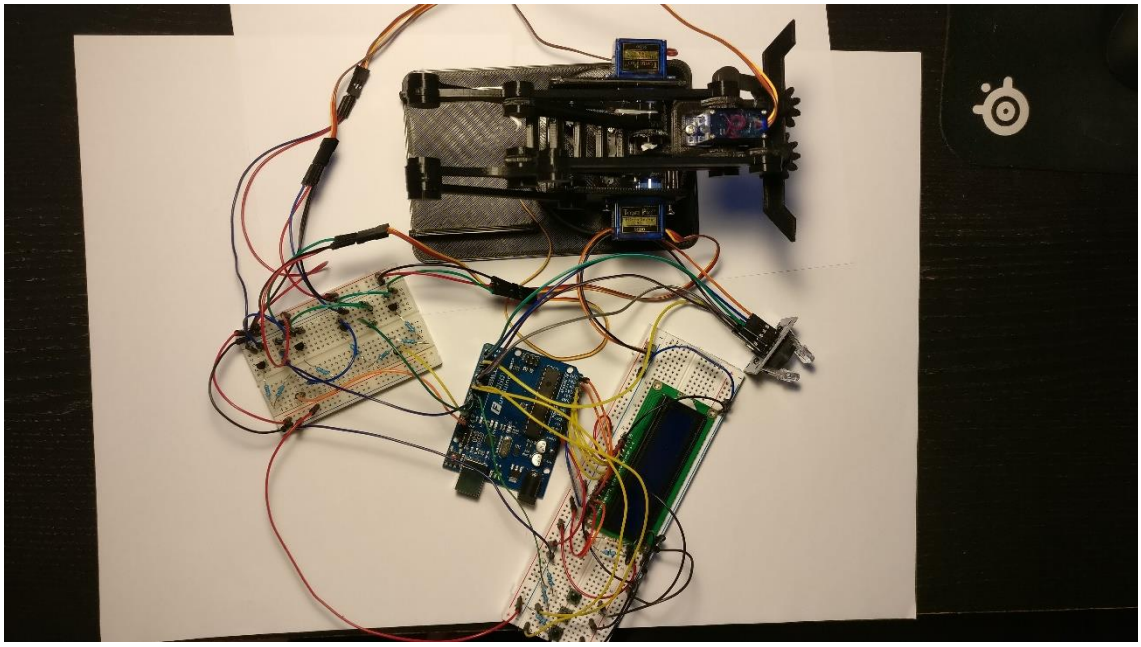


Figura 8 - Montagem do circuito