LAB 2 - Analog-to-Digital Conversion (ADC)

Ana Lopes (98587)[1] and Mariana Mourão (98473)[2]

[1] *Instituto Superior Técnico,*

Integrated Master's in Biomedical Engineering,

ana.rita.santos.lopes@tecnico.ulisboa.pt

[2] *Instituto Superior Técnico,*

Integrated Master's in Biomedical Engineering,

mariana.mourao@tecnico.ulisboa.pt

Exercises:

1. Um conversor analógico-digital (ADC) pode ser modelado como dois processos: amostragem e quantização. A amostragem converte um sinal de tensão variável no tempo num sinal de tempo discreto, gerando uma sequência de números reais. A quantização consiste num mapeamento dos inputs analógicos para códigos digitais, dependendo dos seguintes parâmetros: tensão de operação do microcontrolador e resolução do ADC.

Para o simulador e *Seeeduino Nano*, a tensão de operação corresponde a 5V, sendo que para o *Seeeduino XIAO* corresponde a 3.3V. Já relativamente à resolução do ADC, para todas as placas testadas (simulador, *Seeeduino Nano* e *Seeeduino XIAO*) é, por *default*, 10 bits, significando que os pins analógicos podem detetar 2^{10} = 1024 níveis discretos (níveis de quantização) analógicos, mapeando as tensões analógicas de input entre 0V à tensão de operação (5V ou 3.3V) para códigos digitais entre 0 a 1023. De notar que a resolução do ADC pode ser modificada através do comando *analogReadResolution()*.

A quantização dos níveis de tensão (resolução entre leituras digitais) determina-se pela expressão 1, sendo que o range de níveis de quantização é computada através da expressão 2, para $valor_{ADC}$= 0 e $valor_{ADC}$ = 1023. Na tabela 1 estão dispostos estes valores para as diferentes placas testadas.

$$\text{Quantização de tensões} = \frac{tensão\ de\ operação}{1024} \qquad \text{Expressão 1}$$

$$Voltagem = valor_{ADC} * \left(\frac{tensão\ de\ operação}{1024}\right)$$ Expressão 2

Neste exercício, considera-se enquanto tensão analógica de entrada a tensão no nó de saída de um divisor de tensão, composto por duas resistências R1 = R2 = 10k ohm em série. Ora, duas resistências em série equivale a uma resistência de R = R1 + R2. Pela lei de ohm tem-se que Vin = I*R, com Vin = 3.3V. Deste modo, tem-se que

$$I = Vin/(R1 + R2)$$ Expressão 3

Por outro lado, aos terminais de R2 tem-se Vout, dada por Vout = I2*R2. Como as resistências estão em série, I = I2, pelo que combinando as equações tem-se que

$$Vout = \frac{R2}{R1+R2} * Vin$$ Expressão 4

Como R1=R2, Vout = Vin/2 = 3.3/2 = 1.65 V.

Para a tensão de entrada analógica de 1.65 V, o nível de tensão (código digital) atribuído é teoricamente computado a partir da expressão 2, resolvendo em ordem a $valor_{ADC}$ (valor inteiro). Para as placas cuja tensão de operação é 3.3V, tem-se que o código digital atribuído será 512, com o nível de tensão de 1.65V. Já para as placas cuja tensão de operação é 5V, o código digital atribuído será 338, com o nível de tensão de 1.645V.

Tabela 1- Comparação entre ambos os Seeeduinos, XIAO e NANO e com o simulador.

	Tensão de operação (V)	Resolução ADC (bits)	Quantização dos níveis de tensão (mV)	Range de níveis de quantização (V)	Código Digital atribuído ao DT	Nível de Tensão atribuído ao DT (V)
Seeeduino XIAO	3.3	10	3.22	0-3.29	512-516	1.64-1.66
Seeeduino NANO	5	10	4.88	0-4.99	344-346	1.68-1.69
Simulador	5	10	4.88	0-4.99	338	1.65

Procedendo à aquisição dos dados experimentais para uma frequência de amostragem de 1Hz, bem como configurando a comunicação série segundo o protocolo *default* SERIAL_8N1, com *baud rate* fixa de 9600 bps, obtiveram-se os dados ilustrados graficamente na figura 2, sendo que o código digital atribuído à tensão analógica de entrada (experimentalmente medido através do comando *analogRead()*) e o respetivo nível de tensão (calculado através da expressão 2) encontram-se na tabela 1. Ora, verifica-se que, ao contrário do que seria de esperar, uma sequência não constante de códigos digitais é adquirida

(excetuando para o simulador, em que o código digital atribuído é constante), ainda que rondando os códigos digitais previstos para as placas testadas. O maior desvio face à estimativa teórica refere-se ao *Seeeduino Nano*, em que seria previsto obter um código digital de 338. Isto pode dever-se ao facto da tensão de operação do *Seeeduino Nano* poder não ser exatamente 5V, devido, por exemplo, a tolerâncias e imprecisões nos componentes usados pelo sub-sistema de alimentação.

Quanto às oscilações observadas nos códigos digitais atribuídos, tem-se que, em teoria, no processo de quantização duma determinada tensão de entrada DC (constante no tempo), o código digital atribuído seria necessariamente constante. Contudo, devido a não idealidades do ADC, ou porque a tensão de entrada estaria no limiar entre atribuir a níveis de tensão adjacentes, produzem-se flutuações nos códigos digitais atribuídos. Essas não idealidades podem resultar de efeitos térmicos ou do próprio funcionamento do ADC (muitos canais adjacentes e multiplexados, sendo que o *switching* entre os canais pode introduzir pequenos artefactos na medição), fazendo com que as medições sejam constantes dentro de um determinado erro de quantização. Na figura 2, as oscilações observadas dizem respeito ao fenómeno designado ruido de quantização, o qual resulta precisamente do erro de quantização, estando tipicamente caracterizado nas especificações do microcontrolador. Os ADC são fabricados com um output ou modo de funcionamento que opera dentro de determinados valores de tolerância, sendo que em torno dessa tolerância tem se um resultado que não é constante, tendo uma pequena variação de 1 ou 2 LSBs que vai afetar o código digital que é atribuído a uma determinada tensão. Note-se que no emulador não simula estas não idealidades.

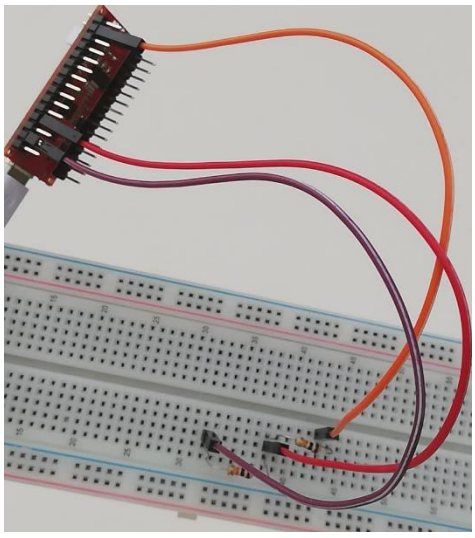

Figura 1 – Montagem do divisor de tensão com o Seeeduino Nano.

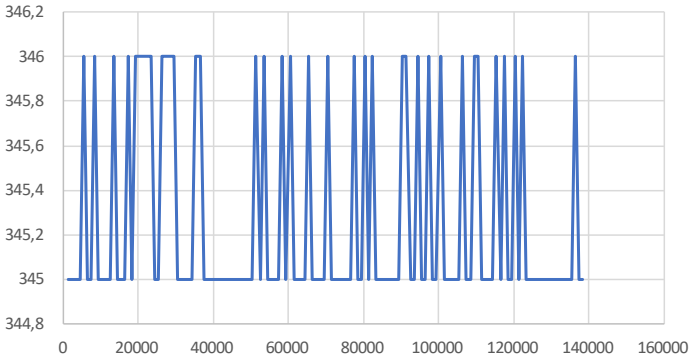

Figura 2 – Quantização da tensão de entrada analógica 1.65V, para baud rate de 9600 bps e frequência de amostragem 1Hz.

2. Recorrendo à mesma montagem do divisor de tensão, bem como mantendo a configuração da comunicação série (protocolo *default* SERIAL_8N1, transmitindo para a porta série 10 bits por caracter; *baud rate* fixa de 9600 bps, implicando 0.104 ms por cada bit transferido), o volume de dados foi controlado consoante a frequência de amostragem definida a partir da porta série, tendo sido testadas as seguintes: 1Hz, 10Hz, 100Hz e 1000Hz.

Para cada aquisição dos dados, obtém-se uma sequência formatada de valores separados por vírgula (CSV), da forma <TIME>, <A>, com <TIME> como sendo o vetor dos instantes temporais gerados com o comando *millis() (elapsed times*, em milissegundos, desde que o código começou a ser executado), e <A> o vetor dos códigos digitais atribuídos à tensão no nó de saída do divisor de tensão (tensão DC, constante no tempo). Para os cálculos posteriores, os dados impressos na porta série foram transferidos para o *Excel*. No ambiente *Excel*, a estimativa do período de amostragem a partir dos dados experimentais foi computada como a diferença entre valores consecutivos no vetor <TIME>, obtendo-se um vetor <SAMPLING_PERIODS> com N-1 intervalos temporais (em milissegundos), com N como sendo o comprimento do vetor <TIME>. De forma a obter uma estimativa do período de amostragem estatisticamente mais significativa, computou-se a média e desvio-padrão dos valores contidos no vetor <SAMPLING_PERIODS>, também tendo-se determinado os valores máximo e mínimo. Na tabela 2 apresentam-se os resultados estatísticos referidos, sendo que nas figuras 2, 3, 4 e 5 ilustra-se graficamente os dados <TIME>, <A> adquiridos com a respetiva frequência de amostragem.

Tabela 2- Resultados estatisticos extraidos a partir dos resultados experimentais, para baud rate de 9600 bps.

Frequência de amostragem (Hz)	Período de Amostragem			
	Média (ms)	Desvio-padrão (ms)	Máximo (ms)	Mínimo (ms)
1 Hz	1000	0	1000	1000
10 Hz	100.007	0.082	101	100
100 Hz	10.450	0.653	13	10
1000 Hz	10.364	1.067	13	1

De forma a analisar os resultados apresentados na tabela 2, ter-se-á de determinar a quantidade de dados que é esperada ser transmitida para a porta série (segundo a configuração estabelecida), e com isso concluir se a *baud rate* fixa de 9600 bps permite amostrar os dados à frequência de amostragem pretendida.

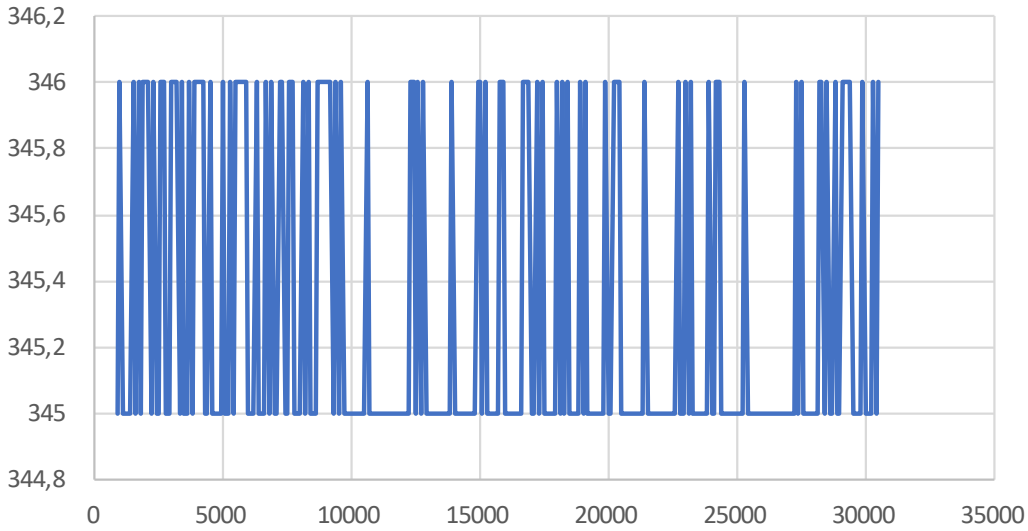

Figura 3 - Quantização da tensão de entrada analógica 1.65V, para baud rate de 9600 bps e frequência de amostragem 10Hz.

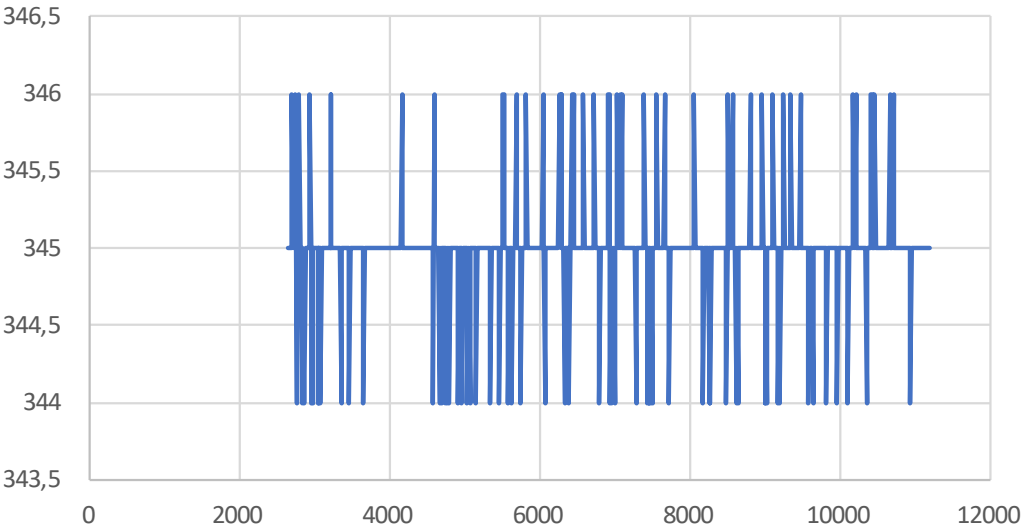

Figura 4 - Quantização da tensão de entrada analógica 1.65V, para baud rate de 9600 bps e frequência de amostragem 100Hz.

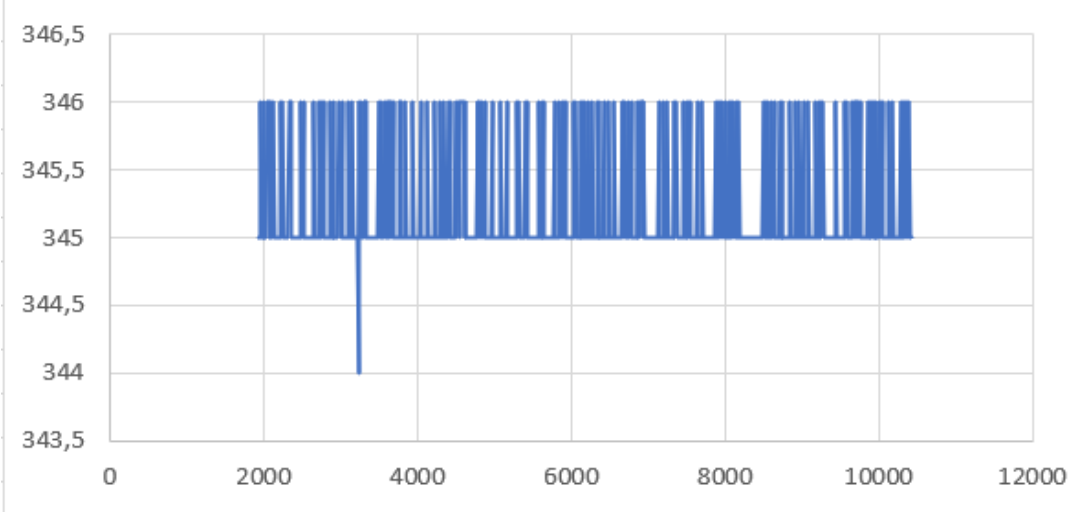

Figura 5 - Quantização da tensão de entrada analógica 1.65V, para baud rate de 9600 bps e frequência de amostragem 1000Hz.

Ora, tendo em conta o exposto no exercício 1, o código digital atribuído à tensão no nó de saída do divisor de tensão (1.65 V) ronda um valor teórico da ordem dos 10^2 (para o *Seeeduino NANO* ronda os 345 ± 1LBS), pelo que a sua representação requer 3 caracteres. Adicionalmente, e para o formato CSV pretendido, utiliza-se uma vírgula para separar as medições, representando-se por 1 caracter (‘,’).

Quanto ao *elapsed time* desde que o código começou a ser executado, pode assumir valores inteiros entre 0 a 4294967296, incrementando entre 1 a 10 caracteres a imprimir na porta série. Para além disto, o comando *Serial.println()* (executado 1 vez durante cada iteração em que se transmite os dados amostrados para a porta série) adiciona os caracteres ‘\r’ e ‘\n’. Deste modo, podem ser impressos 7 a 16 caracteres, requerendo a transmissão de 7 chars *10 bist/char = 70 bits a 16 chars*10 bits/char = 160 bits para a porta série, resultando entre 70 bits * 0.104 ms/bit = 7.28 ms (limite inferior) a 160 bits *0.104 ms/bit = 16.64 ms (limite superior) para os dados serem transmitidos. A expressão 5 generaliza o raciocínio exposto (tendo em conta o protocolo default SERIAL_8N1, estabelecendo 10 bits/char), ao considerar-se que 6 caracteres serão sempre impressos (3 caracteres do código digital; 1 caracter ‘,’; 2 caracteres ‘\r’ e ‘\n’), sendo CT o número de caracteres para a representação dos elapsed times.

$$\text{tempo de transmissão (ms)} = \frac{(6 \text{ chars} + \text{CT chars}) * 10 \text{ bits/char}}{\text{baud rate bits/s}} * 10^3 \qquad \text{Expressão 5}$$

Assim, para que teoricamente os dados sejam corretamente amostrados segundo a frequência de amostragem fs definida, a condição definida na expressão 6 tem de verificar-se, assumindo-se que as restantes operações executadas no microcontrolador são instantâneas. Se a condição não se verificar, teoricamente prevê-se que os dados serão amostrados com uma frequência de amostragem dada pela expressão 7.

tempo de transmissão <= 10^3/fs (ms) Expressão 6

frequência de amostragem (Hz) = 10^3/(tempo de transmissão) Expressão 7

Tendo em conta o limite inferior (7.28 ms) e superior (16.64 ms) para o tempo de transmissão dos dados para a porta série quando o *baud rate* é 9600 bps, teoricamente prevê-se que as frequências de amostragem 1Hz e 10Hz permitam amostrar o sinal corretamente para toda a extensão dos valores *elapsed time* (expressão 2 é sempre verificada), com as amostras temporais a distarem 1000 ms e 100 ms, respetivamente. Comparando com os resultados experimentais obtidos, constata-se que os períodos de amostragem máximo e mínimo para fs = 1Hz correspondem a 1000 ms, resultando numa média do mesmo valor, indo de encontro à previsão teórica. Já para fs = 100 Hz, verifica-se que o período de amostragem máximo corresponde a 101 ms (1 ocorrência), conduzindo a que o período de amostragem médio difira em 0.0068% do teoricamente previsto. Como tal, pode-se concluir que o experimentalmente observado foi previsto teoricamente.

Relativamente às frequências de amostragem 100Hz e 1000Hz, a Expressão 6 não é sempre verificada. Para fs = 100 Hz, a expressão 6 é verificada para o limite inferior do tempo de transmissão, não sendo verificada para o limite superior. De modo a determinar a máxima ordem de grandeza do *elapsed time* que permite teoricamente amostrar os dados corretamente, considera-se na expressão 3 que o tempo de transmissão é igual a 10^3/fs = 10 ms. Resolvendo em ordem a CT, determina-se que CT = 3 caracteres (ordem de grandeza de 10^2 ms) corresponde ao limite máximo para que o período de amostragem definido seja verificado.

Para *elapsed times* da ordem dos 10^3 ms (CT = 4), o tempo de transmissão dos dados é 10.4 ms, não se verificando a expressão 2, obtendo-se uma frequência de amostragem efetiva de 96.15 Hz (expressão 7). Para os dados experimentais obtidos, os *elapsed times* são da ordem dos 10^3 ms (CT = 4 caracteres) a 10^4 ms (CT = 5 caracteres). Como já referido, para CT = 4 o período de amostragem efetivo será de 10.40 ms, sendo que para CT = 5 será de 11.44 ms. Comparando com os resultados experimentais, constata-se que o período de amostragem médio aproxima-se das estimativas teóricas, com o valor mínimo a assumir 10 ms, tornando-se importante denotar que os instantes temporais produzidos pelo comando *millis()* são do tipo *unsigned long*, pelo que casas decimais não são representadas (truncados). Contudo, constata-se que o valor máximo corresponde a 13 ms, afastando-se das estimativas teóricas.

Já para fs = 1000 Hz, a expressão 6 nunca é verificada, pelo que teoricamente os dados não são amostrados segundo a frequência de amostragem pretendida. Pela expressão 7, os dados serão amostrados entre 60Hz (CT = 1) a 137.36Hz (CT = 10). Para os dados experimentais obtidos, os *elapsed times* são novamente da ordem dos 10^3 ms (CT = 4 caracteres) a 10^4 ms (CT = 5 caracteres), tendo associado um período de amostragem de 10.4 ms e 11.44 ms, respetivamente. Comparando com os resultados experimentais, constata-se que o período de amostragem médio se aproxima das estimativas teóricas. Contudo, constata-se que o valor mínimo corresponde a 1 ms (período de amostragem pretendido), dizendo respeito aos primeiros valores no vetor <SAMPLING_PERIODS>. Ora, isto pode dever-se ao facto do *buffer* de envio inicialmente estar a enviar os dados no tempo certo, sendo que quando os dados deixam de ser consumidos a uma velocidade suficientemente rápida, parte dos dados do *buffer*

de envio da porta série podem não ser transmitidos para o recetor, resultando na sua substituição e consequente desfasamento temporal, introduzindo-se num atraso na transmissão dos dados.

Por último, através das figuras 2, 3, 4 e 5 pode-se novamente constatar o ruído de quantização, sendo que quanto maior o volume de dados (maior frequência de amostragem), mais ocorrências de oscilações dentro do limite de tolerância do ADC (345 ± 1LBS) são verificadas.

3.

No exercício anterior, através da variação do volume de dados (pelo ajuste da frequência de amostragem) avaliou-se o desempenho do sistema é afetado. Neste exercício, pretende-se analisar outro grau de liberdade associado ao processo de transmissão de dados e consequente impacto no desempenho do sistema: taxa de transferência dos dados, através do ajuste do *baud rate* (frequência de amostragem fixa em 100Hz), testando-se para 300 bps, 9600 bps e 115200 bps. À semelhança do exercício anterior, na tabela 3 apresentam-se os mesmos resultados estatísticos referentes ao vetor <SAMPLING_PERIODS>, sendo que nas figuras 4, 6 e 7 ilustram-se graficamente os dados <TIME>, <A> adquiridos com a respetiva *baud rate*.

Tabela 3- Resultados estatisticos extraidos a partir dos resultados experimentais, para frequência de amostragem de 100Hz.

Baud rate (bits/s)	Período de Amostragem			
	Média (ms)	Desvio-padrão (ms)	Máximo (ms)	Mínimo (ms)
300 bps	353.087	70.344	367	10
9600 bps	10.450	0.653	13	10
115200 bps	10.039	0.194	11	10

Aplicando o mesmo raciocínio exposto no exercício 2, na tabela 4 expõem-se os limites inferior (CT = 1) e superior (CT = 10) do tempo de transmissão dos dados para cada *baud rate*, calculados através da expressão 5.

Tabela 3 - Limites inferior (CT = 1) e superior (CT = 10) do tempo de transmissão dos dados para cada baud rate.

Baund rate	Limite inferior (ms)	Limite superior (ms)
300 bps	233.1	523.8
9600 bps	7.28	16.64
115200 bps	0.61	1.39

Os dados referentes à *baud rate* de 9600 bps (fs = 100Hz) foram analisados no exercício 2. Para a *baud rate* de 300 bps, conclui-se que teoricamente os dados não são amostrados segundo a frequência de amostragem pretendida (expressão 6 nunca é verificada). Pela expressão 7, os dados serão amostrados entre 1.88Hz (CT = 1) a 4.29Hz (CT = 10). Para os dados experimentais obtidos, os *elapsed times* são da ordem dos 10^4 ms (CT = 5 caracteres), pelo que o período de amostragem efetivo será de 366.63 ms. Comparando com os resultados experimentais, constata-se que o período de amostragem médio se aproxima da estimativa teórica, registando-se um valor máximo de 367 ms. Semelhante ao referido no exercício 2, o valor mínimo de 10 ms (período de amostragem pretendido) corresponde aos primeiros valores no vetor <SAMPLING_PERIODS>, podendo dever-se à transmissão no tempo certo dos dados pelo *buffer* de envio, com posterior desfasamento temporal para instantes subsequentes.

Já relativamente à *baud rate* de 115200 bps, teoricamente prevê-se que permite amostrar o sinal corretamente para toda a extensão dos valores *elapsed time* (expressão 6 é sempre verificada), com as amostras temporais a distarem 10 ms. Comparando com os resultados experimentais obtidos, constata-se que o período de amostragem médio se aproxima da estimativa teórica, registando-se valores máximo e mínimo de 11 ms e 10 ms. As ocorrências de 11ms podem dever-se a oscilações estocásticas introduzidas pelo *hardware*, não sendo representativas na estimativa final do período de amostragem, com o desvio padrão duas ordens de grandeza inferior à média.

Mais uma vez, nas figuras 4, 6 e 7 constata-se que quanto maior a *baud rate*, mais ocorrências de ruído de quantização são observadas.

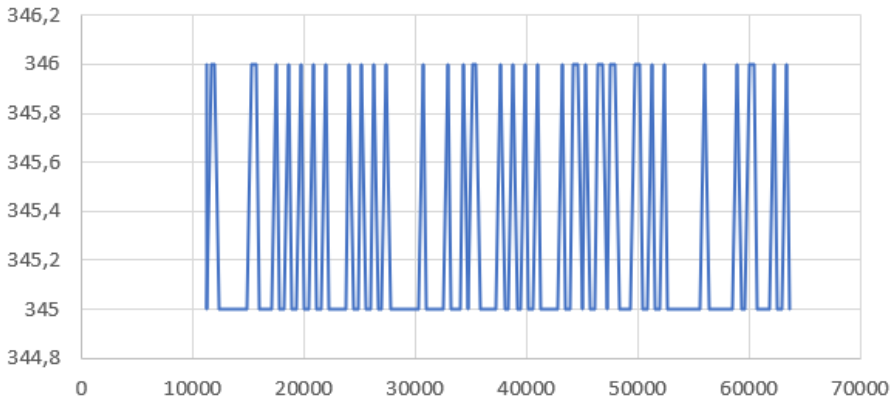

Figura 6 - Quantização da tensão de entrada analógica 1.65V, para baud rate de 300 bps e frequência de amostragem 100Hz.

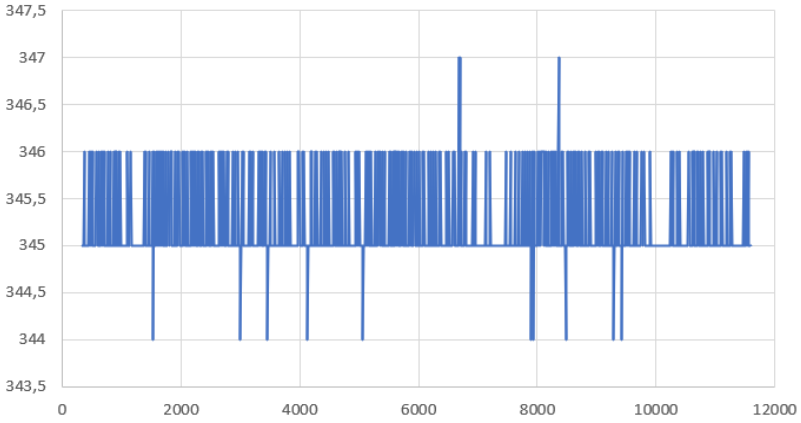

Figura 7 - Quantização da tensão de entrada analógica 1.65V, para baud rate de 115200 bps e frequência de amostragem 100Hz.

4. Neste exercício, pretende-se avaliar o impacto de uma outra variável no desempenho do sistema: o tipo de dados. Para tal, utilizou-se o sensor de temperatura TMP36, o qual permite medir uma gama de temperaturas bastante ampla (-40°C a 125°C), e com uma resolução de 0,1°C, possuindo um fator de escala de 10mV/ºC. Estes sensores fornecem uma saída de tensão que é linearmente proporcional à temperatura em grau Celsius (°C). Para a aquisição dos dados experimentais, definiu-se uma frequência de amostragem de 100Hz e baud rate de 9600 bps, ilustrando-se os dados TMP digitais e convertidos graficamente nas figuras 8 e 9, respetivamente. Os resultados estatísticos referentes ao vetor <SAMPLING_PERIODS> apresentam-se na tabela 5.

Tabela 5 - Resultados estatisticos extraidos a partir dos resultados experimentais referents aos dados TMP digitais e convertidos, para frequência de amostragem de 100Hz e baud rate de 9600 bps.

Dados TMP	Período de Amostragem			
	Média (ms)	Desvio-padrão (ms)	Máximo (ms)	Mínimo (ms)
Digitais (int data type)	10.35	0.57	13	10
Convertidos (float data type)	12.37	0.77	14	10

Para a conversão dos dados TMP, aplica-se a expressão 2 apresentada no exercício 1, de modo a determinar o nível de tensão atribuído à tensão analógica de entrada. A esta tensão subtrai-se -0.4 V, correspondente a 40ºC * 10mV/Cº. Após isto, divide-se por 10*10^-3 = 10^-2, correspondendo a multiplicar por 100, por forma a obter-se o nível de graus centigrados atribuído à tensão de entrada.

Para os dados experimentais TMP digitais (int data type), os códigos digitais atribuídos ao nível de voltagem do TMP eram da ordem dos 10^2, sendo representados por 3 caracteres. Deste modo, a expressão 5 derivada no exercício 2 aplica-se também para esta situação, novamente obtendo-se os limites inferior e superior do tempo de transmissão dos dados de 7.28 ms a 16.64 ms, respetivamente. Para os dados experimentais obtidos, os elapsed times são da ordem dos 10^3 ms (CT = 4 caracteres) a 10^4 ms (CT = 5 caracteres). Como já referido, para CT = 4 o período de amostragem efetivo será de 10.42 ms, sendo que para CT = 5 será de 11.44 ms. Comparando com os resultados experimentais, constata-se que o período de amostragem médio se aproxima das estimativas teóricas.

Já para os dados experimentais TMP convertidos (float data type), os níveis de graus celcius atribuídos à tensão de entrada são representados por 5 caracteres. Deste modo, a expressão 5 altera-se para a expressão 8.

$$\text{tempo de transmissão (ms)} = \frac{(8 \text{ chars} + \text{CT chars}) * 10 \text{ bits/char}}{\text{baud rate bits/s}} * 10^3 \qquad \text{Expressão 8}$$

Assim, para os dados do tipo float transmitidos para a porta série, os limites inferior e superior do tempo de transmissão requerido são 9.38 ms e 18.72 ms, respetivamente. Para os dados experimentais obtidos, os elapsed times são da ordem dos 10^3 ms (CT = 4 caracteres), resultando num tempo de transmissão de 12.5 ms dos dados, pelo que o período de amostragem efetivo seria de 12.5 ms, ao invés dos 10ms. Comparando com os resultados experimentais, constata-se que o período de amostragem médio se aproxima da estimativa teórica.

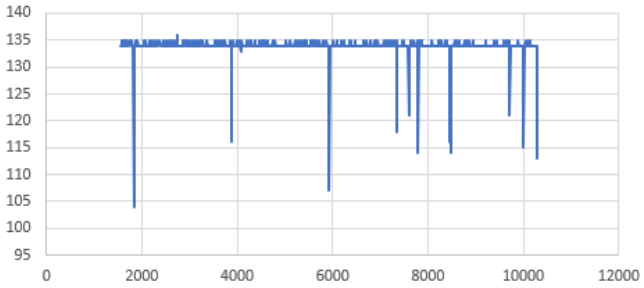

Figura 8 – Dados TMP digitais, para baud rate de 9600 bps e frequência de amostragem 100Hz.

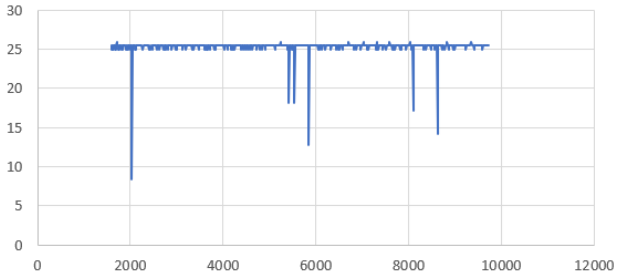

Figura 9 - Dados TMP convertidos, para baud rate de 9600 bps e frequência de amostragem 100Hz.

Deste modo, conclui-se que converter os dados TMP resulta num maior volume de informação, tendo maior impacto no desempenho do sistema, para além do maior custo computacional envolvido nas operações com floats do que com ints. Conclui-se que é vantajoso transmitir os dados enquanto códigos digitais (int data type).

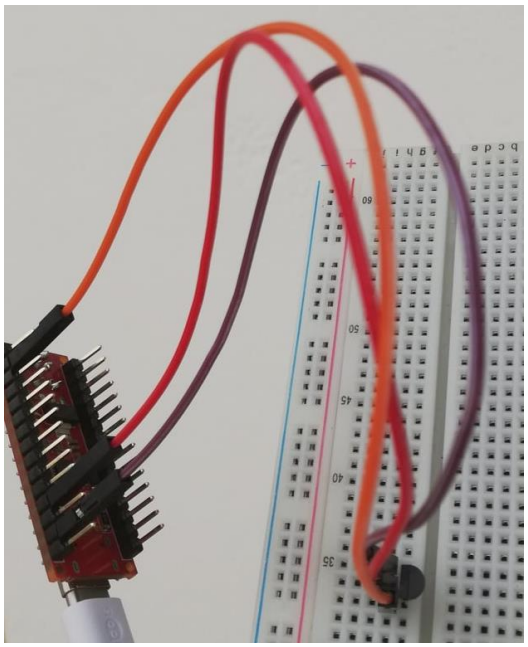

Figura 10 – Montagem para o exercício 4.

Anexos

Anexo A.1 – Código para exercicio 1, 2 e 3

```
int sensorPin = A0; // select the input pin for the potentiometer
int sensorValue = 0;  // initializes the analog value coming from the sensor, and since it is
                      // converted to a digital number from 0-1023, the data type int is the
                      // more suitable
char incoming_byte; // creates the variable for storing the char coming from the serial port

int sampling_rate = 0;
float sampling_period = 0.0;

bool system_state = 0;

unsigned long previous_millis;

unsigned long current_millis;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  if (Serial.available()>0){
    incoming_byte = Serial.read(); // read the incoming byte

    switch(incoming_byte){
      case 'S': // character for starting streaming
      system_state = 1;

      sampling_rate = Serial.parseInt();

      if (sampling_rate != 0){
        sampling_period = 1.0/sampling_rate * pow(10,3);
      }

      previous_millis = 0L;
      break;

      case 'E': // character for ending streaming
      system_state = 0;
      break;
      }
  }
```

```
  // reply only when data is received


  if (system_state && sampling_period!=0.0)
  {
    current_millis = millis();
    if (current_millis - previous_millis >= sampling_period){
      sensorValue = analogRead(sensorPin); // reads the value from the specific analog pin

      previous_millis = current_millis;

      Serial.print(current_millis);
      Serial.print(",");
      Serial.println(sensorValue);
    }
  }
}
```

Anexo A.2 - Código para exercicio 4

```
int sensorPin = A0; // select the input pin for the potentiometer
int sensorValue = 0;  // initializes the analog value coming from the sensor, and since it is
                      // converted to a digital number from 0-1023, the data type int is the
                      // more suitable
char incoming_byte; // creates the variable for storing the char coming from the serial port

int sampling_rate = 0;
float sampling_period = 0.0;

bool system_state = 0; // initializes the state of the system (0 being off and 1 being on)

float value_centigers; // valor TMP convertido

unsigned long previous_millis;

unsigned long current_millis;

bool centigers = 0; // 0 para transmitir dados TMP digitais; 1 para transmitir dados TMP convertidos

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  if (Serial.available()>0){
    incoming_byte = Serial.read(); // read the incoming byte

    switch(incoming_byte){
      case 'S': // character for starting streaming
      system_state = 1;

      sampling_rate = Serial.parseInt();

      if (sampling_rate != 0){
        sampling_period = 1.0/sampling_rate * pow(10,3);
      }

      previous_millis = 0L;
      break;

      case 'E': // character for ending streaming
      system_state = 0;
      break;
```

```
      }
  }
  // reply only when data is received


  if (system_state && sampling_period!=0.0)
  {
    current_millis = millis();
    if (current_millis - previous_millis >= sampling_period){
      sensorValue = analogRead(sensorPin); // reads the value from the specific analog pin

      previous_millis = current_millis;

      Serial.print(current_millis);
      Serial.print(",");

      if (centigers){
        value_centigers = (sensorValue*(5/pow(2,10)) - 0.4)*100;
        Serial.println(value_centigers);
      }
      else{
        Serial.println(sensorValue);
      }
    }
  }
}
```