1. //Власов Роман Евгеньевич
2. //группа 250541
3. //Микропроцессорное устройство распознавания речи
4. // Определение макроса uspeech\_h
5. #define uspeech\_h
6. // Подключение заголовочного файла Arduino.h для доступа к функциям платформы Arduino
7. #include "Arduino.h"
8. // Подключение библиотеки для работы со строками
9. #include <string.h>
10. // Подключение математической библиотеки для использования математических функций
11. #include <math.h>
12. // Подключение стандартной библиотеки для общего функционала
13. #include <stdlib.h>
14. // Определение порога тишины
15. #define SILENCE 2000
16. // Определение константы F\_DETECTION для обнаружения фонемы /f/
17. #define F\_DETECTION 3
18. // Определение константы F\_CONSTANT (порог для /f/)
19. #define F\_CONSTANT 350
20. // Определение константы MAX\_PLOSIVETIME (максимальное время для определения взрывного звука)
21. #define MAX\_PLOSIVETIME 1000
22. // Определение константы PROCESS\_SKEWNESS\_TIME (период анализа данных)
23. #define PROCESS\_SKEWNESS\_TIME 15
24. //Класс для распознавания звука
25. class signal {
26. public:
27. // Объявление массива для хранения аудиоданных (буфер из 32 выборок)
28. int arr[32];
29. // Объявление переменной для средней мощности сигнала
30. int avgPower;
31. // Объявление тестового коэффициента для отладки
32. int testCoeff;
33. // Объявление минимального значения громкости, после которого сигнал считается готовым к распознанию
34. int minVolume;
35. // Объявление порога для распознавания фонемы /f/ (настраиваемый)
36. int fconstant;
37. // Объявление порога для распознавания фонем, таких как /ee/ или /i/
38. int econstant;
39. //Объявление порога для распознавания фонем, таких как /a/, /o/, /r/, /l/
40. int aconstant;
41. // Объявление порога для распознавания фонем, таких как /z/, /v/ или /w/
42. int vconstant;
43. // Объявление порога для распознавания фонем, таких как /sh/ или /ch/ (значения выше – фонема /s/)
44. int shconstant;
45. // Объявление флага, разрешающего распознавание фонемы /f/
46. bool f\_enabled;
47. // Объявление коэффициента усиления для регулировки чувствительности
48. int amplificationFactor;
49. // Объявление порога мощности микрофона (значения ниже игнорируются)
50. int micPowerThreshold;
51. // Объявление масштабного коэффициента для входного сигнала
52. int scale;
53. // Объявление переменной для хранения распознанной фонемы (символ)
54. char phoneme;
55. // Объявление конструктора класса, принимающего номер порта
56. signal(int port);
57. // Объявление переменной для хранения мощности микрофона
58. int micPower;
59. // Объявление метода для выборки звука
60. void sample();
61. // Объявление метода для определения максимальной мощности сигнала
62. unsigned int maxPower();
63. // Объявление метода для вычисления общей мощности сигнала
64. unsigned int power();
65. // Объявление метода для вычисления отношения сигнал/шум (SNR)
66. int snr(int power);
67. //Объявление метода для калибровки микрофона (определение фонового уровня)
68. void calibrate();
69. // Объявление метода для распознавания фонемы (возврат символа)
70. char getPhoneme();
71. // Объявление переменной для хранения значения калибровки (средний уровень фонового шума)
72. int calib;
73. private:
74. // Объявление переменной для хранения номера порта, к которому подключён микрофон
75. int pin;
76. // Объявление переменной для хранения времени (миллисекунды) – может использоваться для отладки или тайминга
77. int mil;
78. // Объявление переменной для хранения позиции в массиве с максимальной амплитудой
79. int maxPos;
80. // Объявление флага, указывающего на наличие тишины
81. bool silence;
82. // Объявление массива для хранения истории коэффициентов (используемого для фильтрации шума)
83. unsigned int overview[7];
84. // Объявление приватного метода для вычисления «сложности» сигнала (отношение суммы модулей производных к мощности)
85. unsigned int complexity(int power);
86. };
87. //Класс для накопления статистических показателей
88. class statCollector {
89. public:
90. // Объявление переменных для количества выборок, среднего значения и моментов 2-го, 3-го и 4-го порядка
91. int n, mean, M2, M3, M4;
92. // Объявление конструктора класса
93. statCollector();
94. // Объявление метода для вычисления куртоза (остроты распределения)
95. int kurtosis();
96. // Объявление метода для вычисления асимметрии распределения
97. int skew();
98. // Объявление метода для получения среднего значения
99. int \_mean();
100. //Объявление метода для получения стандартного отклонения(возвращается M2)
101. int stdev();
102. // Объявление метода для накопления статистических данных (обновление моментов) с новым значением x
103. void collect(int x);
104. };
105. //Класс для аккумуляции фонем (слогов)
106. class syllable {
107. public:
108. // Объявление аккумуляторов для счёта появлений фонем: f, e, o, s, h, v
109. int f, e, o, s, h, v;
110. // Объявление переменных для хранения максимальных позиций (или показателей) для каждой фонемы
111. int maxf, maxe, maxo, maxs, maxh, maxv;
112. // Объявление переменных для модальности каждой фонемы (индикатор наличия двух пиков в распределении)
113. int modalityf, modalitye, modalityo, modalitys, modalityh, modalityv;
114. // Объявление переменной для хранения длины произнесённого слога (количество фонем)
115. int length;
116. // Объявление счётчика взрывных звуков (плозивов)
117. int plosiveCount;
118. // Объявление конструктора класса syllable
119. syllable();
120. // Объявление метода для сброса аккумуляторов (вызывается при обнаружении тишины)
121. void reset();
122. // Объявление метода для классификации символа (фонемы) с обновлением аккумуляторов в зависимости от входного символа
123. void classify(char c);
124. // Объявление метода для возврата аккумуляторов в виде указателя на массив int
125. int\* tointptr();
126. // Объявление метода для отладочного вывода накопленных данных на Arduino
127. void debugPrint();
128. // Объявление метода для расчёта «расстояния» (схожести) между двумя слогами
129. void distance(syllable s);
130. private:
131. // Объявление временных аккумуляторов для промежуточного подсчёта фонем
132. char cf, ce, co, cs, ch, cv;
133. // Объявление переменных для хранения предыдущих значений временных аккумуляторов (для вычисления пиков и модальностей)
134. char prevf, preve, prevo, prevs, prevh, prevv;
135. // Объявление переменной для хранения текущего пика (фонемы) и флага ожидания пробела (для разделения слогов)
136. char currPeak, expectSp;
137. // Объявление переменной для хранения времени последнего обновления (определение интервала между слогами)
138. unsigned long lastTime;
139. };
140. #include "uspeech.h"
141. // Реализация метода распознавания фонемы getPhoneme класса signal
142. char signal::getPhoneme() {
143. // Выполнение выборки звуковых данных
144. sample();
145. // Вычисление суммарной мощности (энергии) сигнала
146. unsigned int pp = power();
147. // Сравнение вычисленной мощности с пороговым значением тишины
148. if (pp > SILENCE) {
149. // Вычисление "сложности" сигнала
150. int k = complexity(pp);
151. // Сдвиг значений в истории коэффициентов (низкочастотная фильтрация)
152. overview[6] = overview[5];
153. overview[5] = overview[4];
154. overview[4] = overview[3];
155. overview[3] = overview[2];
156. overview[2] = overview[1];
157. overview[1] = overview[0];
158. // Запись нового значения сложности в начало массива истории
159. overview[0] = k;
160. // Инициализация переменной для расчёта среднего коэффициента
161. int coeff = 0;
162. // Итерация по элементам массива для суммирования значений
163. for (uint8\_t f = 0; f < 6; f++) {
164. coeff += overview[f];
165. }
166. // Вычисление среднего коэффициента фильтрации
167. coeff /= 7;
168. // Расчёт мощности микрофона с экспоненциальным сглаживанием
169. micPower = 0.05 \* maxPower() + (1 - 0.05) \* micPower;
170. // Сохранение вычисленного коэффициента для отладки
171. testCoeff = coeff;
172. // Классификация фонемы на основе среднего коэффициента
173. if (coeff < econstant) {
174. // Выбор фонемы 'e'
175. phoneme = 'e';
176. } else if (coeff < aconstant) {
177. // Выбор фонемы 'o'
178. phoneme = 'o';
179. } else if (coeff < vconstant) {
180. // Выбор фонемы 'v'
181. phoneme = 'v';
182. } else if (coeff < shconstant) {
183. // Выбор фонемы 'h'
184. phoneme = 'h';
185. } else {
186. // Выбор фонемы 's'
187. phoneme = 's';
188. }
189. // Проверка разрешения распознавания фонемы /f/
190. if (f\_enabled) {
191. // Сравнение мощности микрофона с порогом для фонемы /f/
192. if (micPower > fconstant) {
193. // Возврат фонемы 'f'
194. return 'f';
195. }
196. }
197. // Возврат распознанной фонемы
198. return phoneme;
199. }
200. else {
201. // Обнуление мощности микрофона при отсутствии звука
202. micPower = 0;
203. // Обнуление тестового коэффициента
204. testCoeff = 0;
205. // Возврат пробела (отсутствие звука)
206. return ' ';
207. }
208. }
209. #include "uspeech.h"
210. // Реализация конструктора класса signal
211. signal::signal(int port) {
212. // Присвоение номера порта для микрофона
213. pin = port;
214. // Инициализация порога для фонемы /f/ значением константы F\_CONSTANT
215. fconstant = F\_CONSTANT;
216. // Установка порога для фонем 'e'
217. econstant = 2;
218. // Установка порога для фонем 'o'
219. aconstant = 4;
220. // Установка порога для фонем 'v'
221. vconstant = 6;
222. // Установка порога для фонем 'h'
223. shconstant = 10;
224. // Установка коэффициента усиления для вычисления сложности сигнала
225. amplificationFactor = 10;
226. // Задание порога, ниже которого мощность микрофона считается слишком низкой
227. micPowerThreshold = 50;
228. // Установка масштаба для входного сигнала
229. scale = 1;
230. }
231. // Реализация метода калибровки микрофона на основе усреднения фонового уровня шума
232. void signal::calibrate(){
233. // Инициализация переменной калибровки нулем
234. calib = 0;
235. // Инициализация переменной для суммирования измерений
236. uint32\_t samp = 0;
237. // Цикл для сбора 10 000 выборок фонового шума
238. for (uint16\_t ind = 0; ind < 10000; ind++){
239. // Считывание значения с аналового входа, масштабирование и накопление
240. samp += analogRead(pin) \* scale;
241. }
242. // Вычисление среднего значения фонового шума
243. calib = samp / 10000;
244. }
245. // Реализация метода выборки звукового сигнала с вычитанием калибровочного значения
246. void signal::sample(){
247. // Инициализация счётчика выборки
248. int i = 0;
249. // Цикл для сбора 32 значений с микрофона
250. while (i < 32){
251. // Считывание значения с аналога, масштабирование, вычитание калибровки и запись в массив
252. arr[i] = (analogRead(pin) \* scale - calib);
253. // Инкремент счётчика
254. i++;
255. }
256. }
257. // Реализация метода для вычисления общей мощности сигнала
258. unsigned int signal::power(){
259. // Инициализация переменной для накопления суммы модулей значений сигнала
260. unsigned int j = 0;
261. // Инициализация счётчика
262. uint8\_t i = 0;
263. // Цикл по всем элементам массива с выборками
264. while (i < 32){
265. // Прибавление абсолютного значения текущей выборки к сумме
266. j += abs(arr[i]);
267. // Инкремент счётчика
268. i++;
269. }
270. // Возврат суммарной мощности сигнала
271. return j;
272. }
273. // Реализация метода для вычисления «сложности» сигнала (отношение суммы модулей разностей соседних значений к мощности)
274. unsigned int signal::complexity(int power){
275. // Инициализация переменной для накопления суммы модулей разностей соседних выборок
276. unsigned int j = 0;
277. // Установка счётчика с началом со второго элемента массива
278. uint8\_t i = 1;
279. // Цикл по элементам массива, начиная со второго значения
280. while (i < 32){
281. // Прибавление модуля разности текущей и предыдущей выборки
282. j += abs(arr[i] - arr[i - 1]);
283. // Инкремент счётчика
284. i++;
285. }
286. // Вычисление и возврат значения «сложности» сигнала с учётом коэффициента усиления
287. return (j \* amplificationFactor) / power;
288. }
289. // Реализация метода для определения максимальной амплитуды сигнала
290. unsigned int signal::maxPower() {
291. // Инициализация счётчика для обхода массива
292. int i = 0;
293. // Инициализация переменной для хранения максимальной амплитуды
294. unsigned int max = 0;
295. // Цикл по всем 32 выборкам сигнала
296. while (i < 32){
297. // Сравнение текущего максимального значения с модулем текущей выборки
298. if (max < abs(arr[i])){
299. // Обновление максимального значения амплитуды
300. max = abs(arr[i]);
301. // Запись позиции, на которой зафиксирован максимум
302. maxPos = i;
303. }
304. // Инкремент счётчика
305. i++;
306. // Накопление значения для расчёта средней мощности
307. avgPower += arr[i];
308. }
309. // Вычисление средней мощности сигнала
310. avgPower /= 32;
311. // Возврат максимальной амплитуды
312. return max;
313. }
314. // Реализация метода для вычисления отношения сигнал/шум (SNR)
315. int signal::snr(int power){
316. // Инициализация счётчиков для обхода массива
317. uint8\_t i = 0, j = 0;
318. // Вычисление среднего значения сигнала
319. int mean = power / 32;
320. // Цикл по всем выборкам сигнала
321. while (i < 32){
322. // Накопление квадрата разности между выборкой и средним значением
323. j += sq(arr[i] - mean);
324. // Инкремент счётчика
325. i++;
326. }
327. // Вычисление и возврат соотношения (корень из квадрата ошибки к мощности)
328. return sqrt(j / mean) / power;
329. }
330. #include "uspeech.h"
331. // Реализация конструктора класса syllable
332. syllable::syllable(){
333. // Инициализация аккумуляторов для каждой фонемы
334. f = 0;
335. e = 0;
336. o = 0;
337. s = 0;
338. h = 0;
339. v = 0;
340. // Инициализация длины слога и временных аккумуляторов фонем
341. length = 0;
342. cf = 0;
343. ce = 0;
344. co = 0;
345. cs = 0;
346. ch = 0;
347. cv = 0;
348. // Инициализация модальности каждой фонемы
349. modalityf = 0;
350. modalitye = 0;
351. modalityo = 0;
352. modalitys = 0;
353. modalityh = 0;
354. modalityv = 0;
355. // Установка флага ожидания пробела в исходное состояние
356. expectSp = 1;
357. // Инициализация счётчика взрывных звуков
358. plosiveCount = 0;
359. }
360. // Реализация метода для сброса накопленных значений слога
361. void syllable::reset(){
362. // Сброс аккумуляторов для фонем
363. f = 0;
364. e = 0;
365. o = 0;
366. s = 0;
367. h = 0;
368. v = 0;
369. // Сброс длины слога и временных аккумуляторов
370. length = 0;
371. cf = 0;
372. ce = 0;
373. co = 0;
374. cs = 0;
375. ch = 0;
376. cv = 0;
377. // Сброс модальности для каждой фонемы
378. modalityf = 0;
379. modalitye = 0;
380. modalityo = 0;
381. modalitys = 0;
382. modalityh = 0;
383. modalityv = 0;
384. // Сброс флага ожидания пробела в исходное состояние
385. expectSp = 1;
386. // Сброс счётчика взрывных звуков
387. plosiveCount = 0;
388. }
389. //Метод для классификации (фонемы) с обновлением счетчиков
390. void syllable::classify(char c){
391. // Увеличение длины слога (количество обработанных символов)
392. length++;
393. // Проверка флага ожидания пробела
394. if (expectSp == 0){
395. // Проверка, является ли текущий символ не пробелом
396. if (c != ' '){
397. // Сброс флага ожидания пробела (объединение символов в один слог)
398. expectSp = 1;
399. // Проверка, попадает ли интервал между символами в допустимый порог для взрывных звуков
400. if ((millis() - lastTime) < MAX\_PLOSIVETIME){
401. // Увеличение счётчика взрывных звуков
402. plosiveCount++;
403. }
404. }
405. }
406. // Обработка символа в конструкции switch
407. switch (c) {
408. // Обработка символа 'f'
409. case 'f':
410. // Увеличение основного счётчика для 'f'
411. f++;
412. // Увеличение временного счётчика для 'f'
413. cf++;
414. // Выход из оператора switch для 'f'
415. break;
416. // Обработка символа 'e'
417. case 'e':
418. // Увеличение счётчика для 'e'
419. e++;
420. // Увеличение временного счётчика для 'e'
421. ce++;
422. break;
423. // Обработка символа 'o'
424. case 'o':
425. // Увеличение счётчика для 'o'
426. o++;
427. // Увеличение временного счётчика для 'o'
428. co++;
429. break;
430. // Обработка символа 'v'
431. case 'v':
432. // Увеличение счётчика для 'v'
433. v++;
434. // Увеличение временного счётчика для 'v'
435. cv++;
436. break;
437. // Обработка символа 'h'
438. case 'h':
439. // Увеличение счётчика для 'h'
440. h++;
441. // Увеличение временного счётчика для 'h'
442. ch++;
443. break;
444. // Обработка символа 's'
445. case 's':
446. // Увеличение счётчика для 's'
447. s++;
448. // Увеличение временного счётчика для 's'
449. cs++;
450. break;
451. // Обработка символа пробела
452. case ' ':
453. // Проверка активности флага ожидания пробела
454. if (expectSp != 0){
455. // Сброс флага ожидания пробела для разделения слогов
456. expectSp = 0;
457. // Фиксация времени, используемого для расчёта интервала между слогами
458. lastTime = millis();
459. }
460. break;
461. // Обработка любых остальных символов
462. default:
463. break;
464. }
465. // Периодический анализ для определения пиков и модальности
466. if ((length & PROCESS\_SKEWNESS\_TIME) == 0){
467. // Анализ временного счётчика для 'f'
468. if ((cf > prevf) & (prevf < PROCESS\_SKEWNESS\_TIME)){
469. // Обновление предыдущего значения для 'f'
470. prevf = cf;
471. // Фиксация длины слога при достижении пика 'f'
472. maxf = length;
473. // Установка текущего пика и увеличение модальности для 'f'
474. if (currPeak != 'f'){
475. currPeak = 'f';
476. modalityf++;
477. }
478. }
479. // Анализ временного счётчика для 'e'
480. if ((ce > preve) & (preve < PROCESS\_SKEWNESS\_TIME)){
481. preve = ce;
482. maxe = length;
483. if (currPeak != 'e'){
484. currPeak = 'e';
485. modalitye++;
486. }
487. }
488. // Анализ временного счётчика для 'o'
489. if ((co > prevo) & (prevo < PROCESS\_SKEWNESS\_TIME)){
490. prevo = co;
491. maxo = length;
492. if (currPeak != 'o'){
493. currPeak = 'o';
494. modalityo++;
495. }
496. }
497. // Анализ временного счётчика для 's'
498. if ((cs > prevs) & (prevs < PROCESS\_SKEWNESS\_TIME)){
499. prevs = cs;
500. maxs = length;
501. if (currPeak != 's'){
502. currPeak = 's';
503. modalitys++;
504. }
505. }
506. // Анализ временного счётчика для 'h'
507. if ((ch > prevh) & (prevh < PROCESS\_SKEWNESS\_TIME)){
508. prevh = ch;
509. maxh = length;
510. if (currPeak != 'h'){
511. currPeak = 'h';
512. modalityh++;
513. }
514. }
515. // Анализ временного счётчика для 'v' с порогом 15
516. if ((cv > prevv) & (prevv < 15)){
517. prevv = cv;
518. maxv = length;
519. if (currPeak != 'v'){
520. currPeak = 'v';
521. modalityv++;
522. }
523. }
524. // Сброс временных счётчиков для нового интервала анализа
525. cf = 0;
526. ce = 0;
527. co = 0;
528. cs = 0;
529. ch = 0;
530. cv = 0;
531. }
532. }
533. // Реализация метода для возврата указателя на массив с накопленными данными
534. int\* syllable::tointptr(){
535. // Объявление статического массива для хранения накопленных значений
536. static int matrix[20];
537. // Запись значения аккумулятора 'f' в массив
538. matrix[0] = f;
539. // Запись значения аккумулятора 'e' в массив
540. matrix[1] = e;
541. // Запись значения аккумулятора 'o' в массив
542. matrix[2] = o;
543. // Запись значения аккумулятора 'v' в массив
544. matrix[3] = v;
545. // Запись значения аккумулятора 's' в массив
546. matrix[4] = s;
547. // Запись значения аккумулятора 'h' в массив
548. matrix[5] = h;
549. // Запись значения модальности для 'f' в массив
550. matrix[6] = modalityf;
551. // Запись значения модальности для 'e' в массив
552. matrix[7] = modalitye;
553. // Запись значения модальности для 'o' в массив
554. matrix[8] = modalityo;
555. // Запись значения модальности для 'v' в массив
556. matrix[9] = modalityv;
557. // Запись значения модальности для 's' в массив
558. matrix[10] = modalitys;
559. // Запись значения модальности для 'h' в массив
560. matrix[11] = modalityh;
561. // Запись значения пика для 'f' в массив
562. matrix[12] = maxf;
563. // Запись значения пика для 'e' в массив
564. matrix[13] = maxe;
565. // Запись значения пика для 'o' в массив
566. matrix[14] = maxo;
567. // Запись значения пика для 'v' в массив
568. matrix[15] = maxv;
569. // Запись значения пика для 's' в массив
570. matrix[16] = maxs;
571. // Запись значения пика для 'h' в массив
572. matrix[17] = maxh;
573. // Запись значения длины слога в массив
574. matrix[18] = length;
575. // Запись значения счётчика взрывных звуков в массив
576. matrix[19] = plosiveCount;
577. // Возврат указателя на заполненный массив
578. return matrix;
579. }
580. #include "uspeech.h"
581. //Реализация класса statCollector для накопления статистических данных
582. statCollector::statCollector(){
583. // Инициализация переменных для количества выборок, среднего значения и моментов нулевыми значениями
584. n = 0;
585. mean = 0;
586. M2 = 0;
587. M3 = 0;
588. M4 = 0;
589. }
590. // Реализация метода для возврата среднего значения
591. int statCollector::\_mean(){
592. // Возврат рассчитанного среднего значения
593. return mean;
594. }
595. // Реализация метода для получения стандартного отклонения
596. int statCollector::stdev(){
597. // Возврат значения второго центрального момента
598. return M2;
599. }
600. //Метод для вычисления куртоза - измерение остроты распределения
601. int statCollector::kurtosis(){
602. // Вычисление куртоза по формуле (нормированный 4-й момент минус 3)
603. int kurtosis = (n \* M4) / (M2 \* M2) - 3;
604. // Возврат вычисленного значения куртоза
605. return kurtosis;
606. }
607. // Реализация метода для вычисления асимметрии распределения
608. int statCollector::skew(){
609. // Вычисление асимметрии по формуле (нормированный 3-й момент минус 3)
610. int kurtosis = (n \* M3) / (M2 \* M2 \* M2) - 3;
611. // Возврат полученного значения асимметрии
612. return kurtosis;
613. }
614. // Реализация метода для накопления статистических данных с новым значением x
615. void statCollector::collect(int x) {
616. // Сохранение предыдущего количества выборок
617. int n1 = n;
618. // Увеличение счётчика выборок
619. n = n + 1;
620. // Вычисление разницы между новым значением и текущим средним
621. int delta = x - mean;
622. // Вычисление поправочного коэффициента для среднего
623. int delta\_n = delta / n;
624. // Вычисление квадрата поправочного коэффициента
625. int delta\_n2 = delta\_n \* delta\_n;
626. // Вычисление вспомогательного терма для обновления статистических моментов
627. int term1 = delta \* delta\_n \* n1;
628. // Обновление среднего значения
629. mean = mean + delta\_n;
630. // Обновление четвёртого момента распределения
631. M4 = M4 + term1 \* delta\_n2 \* (n \* n - 3 \* n + 3) + 6 \* delta\_n2 \* M2 - 4 \* delta\_n \* M3;
632. // Обновление третьего момента (асимметрия)
633. M3 = M3 + term1 \* delta\_n \* (n - 2) - 3 \* delta\_n \* M2;
634. // Обновление второго момента (сумма квадратов отклонений)
635. M2 = M2 + term1;
636. }
637. #include <Wire.h>
638. // Подключение библиотеки для работы с интерфейсом I²C
639. #include <LiquidCrystal\_I2C.h>
640. // Подключение библиотеки для управления LCD-дисплеем по шине I²C
641. #include <uspeech.h>
642. // Подключение библиотеки uspeech для распознавания речи
643. #include <string.h>
644. // Подключение стандартной библиотеки C для работы со строками (strcmp, strlen)
645. #define ledGreen 7
646. // Определение номера пина 7 для зелёного светодиода
647. #define ledOrange 6
648. // Определение номера пина 6 для оранжевого светодиода
649. #define ledWhite 5
650. // Определение номера пина 5 для белого светодиода
651. #define buzzerPin 2
652. // Определение номера пина 2 для подключения пьезодинамика (буззера)
653. #define enablePin A1
654. // Определение номера аналогового пина A1 для переключателя (замыкание на GND = включено)
655. #define MIN3(a,b,c) ((a)<(b)?((a)<(c)?(a):(c)):((b)<(c)?(b):(c)))
656. // Макрос для вычисления минимального значения из трёх аргументов
657. signal voice(A0);
658. // Создание объекта voice класса signal, подключённого к аналоговому пину A0 для сбора аудиосэмплов
659. LiquidCrystal\_I2C lcd(0x27, 16, 2);
660. // Создание объекта lcd для работы с I²C LCD-дисплеем по адресу 0x27 (16×2 символа)
661. const int BUFFER\_MAX\_PHONEMES = 32;
662. // Константа: максимальный размер буфера для накопления фонем
663. char inputString[BUFFER\_MAX\_PHONEMES];
664. // Массив для накопления входных фонем в виде строки
665. byte index = 0;
666. // Текущий индекс в буфере inputString
667. const int DICT\_MAX\_ELEMNTS = 3;
668. // Константа: количество элементов (паттернов) в словаре
669. char dict[DICT\_MAX\_ELEMNTS][BUFFER\_MAX\_PHONEMES] = {
670. "vvvoeeeeeeeofff", // Паттерн для команды "green"
671. "hhhhhvoovvvvf", // Паттерн для команды "orange"
672. "booooooffffffff" // Паттерн для команды "white"
673. };
674. // Двумерный массив строк с паттернами фонем для распознавания трёх команд
675. int LOWEST\_COST\_MAX\_THREASHHOLD = 20;
676. // Пороговое значение для допустимой стоимости Левенштейна
677. unsigned long lastCommandTime = 0;
678. // Время (в миллисекундах) последнего распознанного и обработанного слова
679. bool idleDisplayed = false;
680. // Флаг, указывающий, выводилось ли сообщение "ожидание" на дисплей
681. bool voiceEnabled = false;
682. // Флаг, указывающий, включено ли распознавание речи (состояние переключателя)
683. //Параметр ms задаёт длительность звукового сигнала в миллисекундах (по умолчанию 100 мс).
684. void beep(int ms = 100) {
685. digitalWrite(buzzerPin, LOW);
686. delay(ms);
687. digitalWrite(buzzerPin, HIGH);
688. }
689. //Вычисляет длину строки до первого пробела или до символа конца строки
690. int strLength(const char \*s) {
691. int i = 0;
692. while (s[i] && s[i] != ' ')
693. ++i;
694. return i;
695. }
696. //Вычисляет расстояние Левенштейна между строками s1 и s2.
697. unsigned int levenshtein(char \*s1, char \*s2) {
698. unsigned int s1len = strlen(s1), s2len = strlen(s2);
699. unsigned int column[s1len + 1];
700. // Инициализация первого столбца матрицы расстояний
701. for (unsigned int y = 1; y <= s1len; y++)
702. column[y] = y;
703. // Основной цикл по символам второй строки
704. for (unsigned int x = 1; x <= s2len; x++) {
705. column[0] = x;
706. unsigned int lastdiag = x - 1;
707. for (unsigned int y = 1; y <= s1len; y++) {
708. unsigned int olddiag = column[y];
709. // Вычисляем стоимость удаления, вставки и замены
710. column[y] = MIN3(
711. column[y] + 1,
712. column[y - 1] + 1,
713. lastdiag + (s1[y - 1] == s2[x - 1] ? 0 : 1)
714. );
715. lastdiag = olddiag;
716. }
717. }
718. return column[s1len];
719. }
720. /\*\*
721. \* Функция guessWord
722. \* По входной строке target вычисляет стоимость Левенштейна до каждого паттерна
723. \* и возвращает указатель на наиболее близкий или пустую строку, если ни один
724. \* паттерн не уложился в порог.
725. \*/
726. char\* guessWord(char\* target) {
727. static unsigned int cost[DICT\_MAX\_ELEMNTS];
728. // Считаем стоимости для каждого паттерна
729. for (int j = 0; j < DICT\_MAX\_ELEMNTS; j++)
730. cost[j] = levenshtein(dict[j], target);
731. // Ищем лучший (минимальный) cost
732. int best = -1, bestCost = LOWEST\_COST\_MAX\_THREASHHOLD;
733. for (int j = 0; j < DICT\_MAX\_ELEMNTS; j++) {
734. if (cost[j] < bestCost) {
735. bestCost = cost[j];
736. best = j;
737. }
738. }
739. // Возвращаем либо указатель на строку словаря, либо пустую строку
740. return (best >= 0 ? dict[best] : (char\*)"");
741. }
742. /\*\*
743. \* Функция parseCommand
744. \* Обрабатывает распознанное слово: включает нужный светодиод,
745. \* выводит текст на дисплей и сбрасывает флаг ожидания.
746. \*/
747. void parseCommand(char\* str) {
748. char \*g = guessWord(str);
749. if (!g[0]) return; // Если слово не распознано — выходим
750. // Очистка дисплея и вывод метки
751. lcd.clear();
752. lcd.setCursor(0,0);
753. lcd.print("Word:");
754. lcd.setCursor(0,1);
755. // Сначала выключаем все светодиоды
756. digitalWrite(ledGreen, LOW);
757. digitalWrite(ledOrange, LOW);
758. digitalWrite(ledWhite, LOW);
759. // Сравнение распознанного слова с паттернами через strcmp
760. if (strcmp(g, dict[0]) == 0) {
761. lcd.print("green");
762. digitalWrite(ledGreen, HIGH);
763. }
764. else if (strcmp(g, dict[1]) == 0) {
765. lcd.print("orange");
766. digitalWrite(ledOrange, HIGH);
767. }
768. else if (strcmp(g, dict[2]) == 0) {
769. lcd.print("white");
770. digitalWrite(ledWhite, HIGH);
771. }
772. // Обновляем время последней команды и сбрасываем флаг idle
773. lastCommandTime = millis();
774. idleDisplayed = false;
775. }
776. void setup() {
777. Serial.begin(9600);
778. // Инициализация дисплея
779. lcd.init();
780. lcd.backlight();
781. lcd.clear();
782. // Настройка пинов на выход
783. pinMode(ledGreen, OUTPUT);
784. pinMode(ledOrange, OUTPUT);
785. pinMode(ledWhite, OUTPUT);
786. pinMode(buzzerPin, OUTPUT);
787. pinMode(enablePin, INPUT\_PULLUP);
788. digitalWrite(buzzerPin, HIGH); // Выключаем буззер по умолчанию
789. // Стартовая фаза с индикатором загрузки и калибровкой
790. lcd.print("Loading...");
791. unsigned long t0 = millis();
792. while (millis() - t0 < 5000) {
793. unsigned long dt = millis() - t0;
794. // Пищим первые 1 секунду
795. if (dt < 1000) digitalWrite(buzzerPin, LOW);
796. else digitalWrite(buzzerPin, HIGH);
797. // После первой секунды выводим сообщение о калибровке
798. if (dt >= 1000 && dt < 1250) {
799. lcd.clear();
800. lcd.print("Calibrating...");
801. static bool calibStarted = false;
802. if (!calibStarted) {
803. calibStarted = true;
804. // Настройка параметров распознавания перед калибровкой
805. voice.f\_enabled = true;
806. voice.minVolume = 1500;
807. voice.fconstant = 500;
808. voice.econstant = 2;
809. voice.aconstant = 4;
810. voice.vconstant = 6;
811. voice.shconstant = 10;
812. voice.calibrate();
813. }
814. }
815. // Мигаем всеми светодиодами в такт
816. bool st = ((millis() / 250) & 1);
817. digitalWrite(ledGreen, st);
818. digitalWrite(ledOrange, st);
819. digitalWrite(ledWhite, st);
820. delay(50);
821. }
822. // После загрузки — выключаем все индикаторы
823. digitalWrite(buzzerPin, HIGH);
824. digitalWrite(ledGreen, LOW);
825. digitalWrite(ledOrange, LOW);
826. digitalWrite(ledWhite, LOW);
827. // Читаем состояние переключателя и выводим финальное сообщение
828. voiceEnabled = (digitalRead(enablePin) == LOW);
829. lcd.clear();
830. if (voiceEnabled) lcd.print("Say a color...");
831. else lcd.print("Voice OFF");
832. lastCommandTime = millis();
833. idleDisplayed = false;
834. }
835. void loop() {
836. // Отслеживаем смену положения переключателя
837. bool en = (digitalRead(enablePin) == LOW);
838. if (en != voiceEnabled) {
839. voiceEnabled = en;
840. // Сбрасываем буфер ввода
841. index = 0;
842. inputString[0] = '\0';
843. // Обновляем экран и при необходимости калибруем микрофон
844. lcd.clear();
845. if (voiceEnabled) {
846. lcd.print("Calibrating...");
847. voice.calibrate();
848. lcd.clear(); lcd.print("Say a color...");
849. } else {
850. lcd.print("Voice OFF");
851. // Выключаем все светодиоды
852. digitalWrite(ledGreen, LOW);
853. digitalWrite(ledOrange, LOW);
854. digitalWrite(ledWhite, LOW);
855. }
856. lastCommandTime = millis();
857. idleDisplayed = false;
858. }
859. // Если голосовое управление включено — собираем и обрабатываем фонемы
860. if (voiceEnabled) {
861. voice.sample();
862. char p = voice.getPhoneme();
863. // Если пришёл пробел или буфер полон — обрабатываем накопленную строку
864. if (p == ' ' || index >= BUFFER\_MAX\_PHONEMES) {
865. if (strLength(inputString) > 0) {
866. parseCommand(inputString);
867. index = 0;
868. inputString[0] = '\0';
869. }
870. } else {
871. // Добавляем фонему в буфер
872. inputString[index++] = p;
873. inputString[index] = '\0';
874. }
875. // Если нет команд больше 5 секунд — показываем приглашение к вводу
876. if (millis() - lastCommandTime > 5000 && !idleDisplayed) {
877. digitalWrite(ledGreen, LOW);
878. digitalWrite(ledOrange, LOW);
879. digitalWrite(ledWhite, LOW);
880. lcd.clear();
881. lcd.print("Say a color...");
882. idleDisplayed = true;
883. }
884. }
885. }