1. //Власов Роман Евгеньевич
2. //группа 250541
3. // Определение макроса uspeech\_h
4. #define uspeech\_h
5. // Подключение заголовочного файла Arduino.h для доступа к функциям платформы Arduino
6. #include "Arduino.h"
7. // Подключение библиотеки для работы со строками
8. #include <string.h>
9. // Подключение математической библиотеки для использования математических функций
10. #include <math.h>
11. // Подключение стандартной библиотеки для общего функционала
12. #include <stdlib.h>
13. // Определение порога тишины
14. #define SILENCE 2000
15. // Определение константы F\_DETECTION для обнаружения фонемы /f/
16. #define F\_DETECTION 3
17. // Определение константы F\_CONSTANT (порог для /f/)
18. #define F\_CONSTANT 350
19. // Определение константы MAX\_PLOSIVETIME (максимальное время для определения взрывного звука)
20. #define MAX\_PLOSIVETIME 1000
21. // Определение константы PROCESS\_SKEWNESS\_TIME (период анализа данных)
22. #define PROCESS\_SKEWNESS\_TIME 15
23. /\*\*
24. \* Класс для распознавания звука
25. \*/
26. class signal {
27. public:
28. // Объявление массива для хранения аудиоданных (буфер из 32 выборок)
29. int arr[32];
30. // Объявление переменной для средней мощности сигнала
31. int avgPower;
32. // Объявление тестового коэффициента для отладки
33. int testCoeff;
34. // Объявление минимального значения громкости, после которого сигнал считается готовым к распознанию
35. int minVolume;
36. // Объявление порога для распознавания фонемы /f/ (настраиваемый)
37. int fconstant;
38. // Объявление порога для распознавания фонем, таких как /ee/ или /i/
39. int econstant;
40. // Объявление порога для распознавания фонем, таких как /a/, /o/, /r/ или /l/
41. int aconstant;
42. // Объявление порога для распознавания фонем, таких как /z/, /v/ или /w/
43. int vconstant;
44. // Объявление порога для распознавания фонем, таких как /sh/ или /ch/ (значения выше – фонема /s/)
45. int shconstant;
46. // Объявление флага, разрешающего распознавание фонемы /f/
47. bool f\_enabled;
48. // Объявление коэффициента усиления для регулировки чувствительности
49. int amplificationFactor;
50. // Объявление порога мощности микрофона (значения ниже игнорируются)
51. int micPowerThreshold;
52. // Объявление масштабного коэффициента для входного сигнала
53. int scale;
54. // Объявление переменной для хранения распознанной фонемы (символ)
55. char phoneme;
56. // Объявление конструктора класса, принимающего номер порта
57. signal(int port);
58. // Объявление переменной для хранения мощности микрофона
59. int micPower;
60. // Объявление метода для выборки звука
61. void sample();
62. // Объявление метода для определения максимальной мощности сигнала
63. unsigned int maxPower();
64. // Объявление метода для вычисления общей мощности сигнала
65. unsigned int power();
66. // Объявление метода для вычисления отношения сигнал/шум (SNR)
67. int snr(int power);
68. // Объявление метода для калибровки микрофона (определение фонового уровня)
69. void calibrate();
70. // Объявление метода для распознавания фонемы (возврат символа)
71. char getPhoneme();
72. // Объявление переменной для хранения значения калибровки (средний уровень фонового шума)
73. int calib;
74. private:
75. // Объявление переменной для хранения номера порта, к которому подключён микрофон
76. int pin;
77. // Объявление переменной для хранения времени (миллисекунды) – может использоваться для отладки или тайминга
78. int mil;
79. // Объявление переменной для хранения позиции в массиве с максимальной амплитудой
80. int maxPos;
81. // Объявление флага, указывающего на наличие тишины
82. bool silence;
83. // Объявление массива для хранения истории коэффициентов (используемого для фильтрации шума)
84. unsigned int overview[7];
85. // Объявление приватного метода для вычисления «сложности» сигнала (отношение суммы модулей производных к мощности)
86. unsigned int complexity(int power);
87. };
88. /\*\*
89. \* Класс для накопления статистических показателей
90. \*/
91. class statCollector {
92. public:
93. // Объявление переменных для количества выборок, среднего значения и моментов 2-го, 3-го и 4-го порядка
94. int n, mean, M2, M3, M4;
95. // Объявление конструктора класса
96. statCollector();
97. // Объявление метода для вычисления куртоза (остроты распределения)
98. int kurtosis();
99. // Объявление метода для вычисления асимметрии распределения
100. int skew();
101. // Объявление метода для получения среднего значения
102. int \_mean();
103. // Объявление метода для получения стандартного отклонения (возвращается M2)
104. int stdev();
105. // Объявление метода для накопления статистических данных (обновление моментов) с новым значением x
106. void collect(int x);
107. };
108. /\*\*
109. \* Класс для аккумуляции фонем (слогов)
110. \*/
111. class syllable {
112. public:
113. // Объявление аккумуляторов для счёта появлений фонем: f, e, o, s, h, v
114. int f, e, o, s, h, v;
115. // Объявление переменных для хранения максимальных позиций (или показателей) для каждой фонемы
116. int maxf, maxe, maxo, maxs, maxh, maxv;
117. // Объявление переменных для модальности каждой фонемы (индикатор наличия двух пиков в распределении)
118. int modalityf, modalitye, modalityo, modalitys, modalityh, modalityv;
119. // Объявление переменной для хранения длины произнесённого слога (количество фонем)
120. int length;
121. // Объявление счётчика взрывных звуков (плозивов)
122. int plosiveCount;
123. // Объявление конструктора класса syllable
124. syllable();
125. // Объявление метода для сброса аккумуляторов (вызывается при обнаружении тишины)
126. void reset();
127. // Объявление метода для классификации символа (фонемы) с обновлением аккумуляторов в зависимости от входного символа
128. void classify(char c);
129. // Объявление метода для возврата аккумуляторов в виде указателя на массив int
130. int\* tointptr();
131. // Объявление метода для отладочного вывода накопленных данных на Arduino
132. void debugPrint();
133. // Объявление метода для расчёта «расстояния» (схожести) между двумя слогами
134. void distance(syllable s);
135. private:
136. // Объявление временных аккумуляторов для промежуточного подсчёта фонем
137. char cf, ce, co, cs, ch, cv;
138. // Объявление переменных для хранения предыдущих значений временных аккумуляторов (для вычисления пиков и модальностей)
139. char prevf, preve, prevo, prevs, prevh, prevv;
140. // Объявление переменной для хранения текущего пика (фонемы) и флага ожидания пробела (для разделения слогов)
141. char currPeak, expectSp;
142. // Объявление переменной для хранения времени последнего обновления (определение интервала между слогами)
143. unsigned long lastTime;
144. };
145. #include "uspeech.h"
146. // Реализация метода распознавания фонемы getPhoneme класса signal
147. char signal::getPhoneme() {
148. // Выполнение выборки звуковых данных
149. sample();
150. // Вычисление суммарной мощности (энергии) сигнала
151. unsigned int pp = power();
152. // Сравнение вычисленной мощности с пороговым значением тишины
153. if (pp > SILENCE) {
154. // Вычисление "сложности" сигнала
155. int k = complexity(pp);
156. // Сдвиг значений в истории коэффициентов (низкочастотная фильтрация)
157. overview[6] = overview[5];
158. overview[5] = overview[4];
159. overview[4] = overview[3];
160. overview[3] = overview[2];
161. overview[2] = overview[1];
162. overview[1] = overview[0];
163. // Запись нового значения сложности в начало массива истории
164. overview[0] = k;
165. // Инициализация переменной для расчёта среднего коэффициента
166. int coeff = 0;
167. // Итерация по элементам массива для суммирования значений
168. for (uint8\_t f = 0; f < 6; f++) {
169. coeff += overview[f];
170. }
171. // Вычисление среднего коэффициента фильтрации
172. coeff /= 7;
173. // Расчёт мощности микрофона с экспоненциальным сглаживанием
174. micPower = 0.05 \* maxPower() + (1 - 0.05) \* micPower;
175. // Сохранение вычисленного коэффициента для отладки
176. testCoeff = coeff;
177. // Классификация фонемы на основе среднего коэффициента
178. if (coeff < econstant) {
179. // Выбор фонемы 'e'
180. phoneme = 'e';
181. } else if (coeff < aconstant) {
182. // Выбор фонемы 'o'
183. phoneme = 'o';
184. } else if (coeff < vconstant) {
185. // Выбор фонемы 'v'
186. phoneme = 'v';
187. } else if (coeff < shconstant) {
188. // Выбор фонемы 'h'
189. phoneme = 'h';
190. } else {
191. // Выбор фонемы 's'
192. phoneme = 's';
193. }
194. // Проверка разрешения распознавания фонемы /f/
195. if (f\_enabled) {
196. // Сравнение мощности микрофона с порогом для фонемы /f/
197. if (micPower > fconstant) {
198. // Возврат фонемы 'f'
199. return 'f';
200. }
201. }
202. // Возврат распознанной фонемы
203. return phoneme;
204. }
205. else {
206. // Обнуление мощности микрофона при отсутствии звука
207. micPower = 0;
208. // Обнуление тестового коэффициента
209. testCoeff = 0;
210. // Возврат пробела (отсутствие звука)
211. return ' ';
212. }
213. }
214. #include "uspeech.h"
215. // Реализация конструктора класса signal
216. signal::signal(int port) {
217. // Присвоение номера порта для микрофона
218. pin = port;
219. // Инициализация порога для фонемы /f/ значением константы F\_CONSTANT
220. fconstant = F\_CONSTANT;
221. // Установка порога для фонем 'e'
222. econstant = 2;
223. // Установка порога для фонем 'o'
224. aconstant = 4;
225. // Установка порога для фонем 'v'
226. vconstant = 6;
227. // Установка порога для фонем 'h'
228. shconstant = 10;
229. // Установка коэффициента усиления для вычисления сложности сигнала
230. amplificationFactor = 10;
231. // Задание порога, ниже которого мощность микрофона считается слишком низкой
232. micPowerThreshold = 50;
233. // Установка масштаба для входного сигнала
234. scale = 1;
235. }
236. // Реализация метода калибровки микрофона на основе усреднения фонового уровня шума
237. void signal::calibrate(){
238. // Инициализация переменной калибровки нулем
239. calib = 0;
240. // Инициализация переменной для суммирования измерений
241. uint32\_t samp = 0;
242. // Цикл для сбора 10 000 выборок фонового шума
243. for (uint16\_t ind = 0; ind < 10000; ind++){
244. // Считывание значения с аналового входа, масштабирование и накопление
245. samp += analogRead(pin) \* scale;
246. }
247. // Вычисление среднего значения фонового шума
248. calib = samp / 10000;
249. }
250. // Реализация метода выборки звукового сигнала с вычитанием калибровочного значения
251. void signal::sample(){
252. // Инициализация счётчика выборки
253. int i = 0;
254. // Цикл для сбора 32 значений с микрофона
255. while (i < 32){
256. // Считывание значения с аналога, масштабирование, вычитание калибровки и запись в массив
257. arr[i] = (analogRead(pin) \* scale - calib);
258. // Инкремент счётчика
259. i++;
260. }
261. }
262. // Реализация метода для вычисления общей мощности сигнала
263. unsigned int signal::power(){
264. // Инициализация переменной для накопления суммы модулей значений сигнала
265. unsigned int j = 0;
266. // Инициализация счётчика
267. uint8\_t i = 0;
268. // Цикл по всем элементам массива с выборками
269. while (i < 32){
270. // Прибавление абсолютного значения текущей выборки к сумме
271. j += abs(arr[i]);
272. // Инкремент счётчика
273. i++;
274. }
275. // Возврат суммарной мощности сигнала
276. return j;
277. }
278. // Реализация метода для вычисления «сложности» сигнала (отношение суммы модулей разностей соседних значений к мощности)
279. unsigned int signal::complexity(int power){
280. // Инициализация переменной для накопления суммы модулей разностей соседних выборок
281. unsigned int j = 0;
282. // Установка счётчика с началом со второго элемента массива
283. uint8\_t i = 1;
284. // Цикл по элементам массива, начиная со второго значения
285. while (i < 32){
286. // Прибавление модуля разности текущей и предыдущей выборки
287. j += abs(arr[i] - arr[i - 1]);
288. // Инкремент счётчика
289. i++;
290. }
291. // Вычисление и возврат значения «сложности» сигнала с учётом коэффициента усиления
292. return (j \* amplificationFactor) / power;
293. }
294. // Реализация метода для определения максимальной амплитуды сигнала
295. unsigned int signal::maxPower() {
296. // Инициализация счётчика для обхода массива
297. int i = 0;
298. // Инициализация переменной для хранения максимальной амплитуды
299. unsigned int max = 0;
300. // Цикл по всем 32 выборкам сигнала
301. while (i < 32){
302. // Сравнение текущего максимального значения с модулем текущей выборки
303. if (max < abs(arr[i])){
304. // Обновление максимального значения амплитуды
305. max = abs(arr[i]);
306. // Запись позиции, на которой зафиксирован максимум
307. maxPos = i;
308. }
309. // Инкремент счётчика
310. i++;
311. // Накопление значения для расчёта средней мощности
312. avgPower += arr[i];
313. }
314. // Вычисление средней мощности сигнала
315. avgPower /= 32;
316. // Возврат максимальной амплитуды
317. return max;
318. }
319. // Реализация метода для вычисления отношения сигнал/шум (SNR)
320. int signal::snr(int power){
321. // Инициализация счётчиков для обхода массива
322. uint8\_t i = 0, j = 0;
323. // Вычисление среднего значения сигнала
324. int mean = power / 32;
325. // Цикл по всем выборкам сигнала
326. while (i < 32){
327. // Накопление квадрата разности между выборкой и средним значением
328. j += sq(arr[i] - mean);
329. // Инкремент счётчика
330. i++;
331. }
332. // Вычисление и возврат соотношения (корень из среднего квадрата ошибки к мощности)
333. return sqrt(j / mean) / power;
334. }
335. #include "uspeech.h"
336. // Реализация конструктора класса syllable
337. syllable::syllable(){
338. // Инициализация аккумуляторов для каждой фонемы
339. f = 0;
340. e = 0;
341. o = 0;
342. s = 0;
343. h = 0;
344. v = 0;
345. // Инициализация длины слога и временных аккумуляторов фонем
346. length = 0;
347. cf = 0;
348. ce = 0;
349. co = 0;
350. cs = 0;
351. ch = 0;
352. cv = 0;
353. // Инициализация модальности каждой фонемы
354. modalityf = 0;
355. modalitye = 0;
356. modalityo = 0;
357. modalitys = 0;
358. modalityh = 0;
359. modalityv = 0;
360. // Установка флага ожидания пробела в исходное состояние
361. expectSp = 1;
362. // Инициализация счётчика взрывных звуков
363. plosiveCount = 0;
364. }
365. // Реализация метода для сброса накопленных значений слога
366. void syllable::reset(){
367. // Сброс аккумуляторов для фонем
368. f = 0;
369. e = 0;
370. o = 0;
371. s = 0;
372. h = 0;
373. v = 0;
374. // Сброс длины слога и временных аккумуляторов
375. length = 0;
376. cf = 0;
377. ce = 0;
378. co = 0;
379. cs = 0;
380. ch = 0;
381. cv = 0;
382. // Сброс модальности для каждой фонемы
383. modalityf = 0;
384. modalitye = 0;
385. modalityo = 0;
386. modalitys = 0;
387. modalityh = 0;
388. modalityv = 0;
389. // Сброс флага ожидания пробела в исходное состояние
390. expectSp = 1;
391. // Сброс счётчика взрывных звуков
392. plosiveCount = 0;
393. }
394. /\*\*\*
395. \* Метод для классификации входного символа (фонемы) с обновлением соответствующих счетчиков
396. \*/
397. void syllable::classify(char c){
398. // Увеличение длины слога (количество обработанных символов)
399. length++;
400. // Проверка флага ожидания пробела
401. if (expectSp == 0){
402. // Проверка, является ли текущий символ не пробелом
403. if (c != ' '){
404. // Сброс флага ожидания пробела (объединение символов в один слог)
405. expectSp = 1;
406. // Проверка, попадает ли интервал между символами в допустимый порог для взрывных звуков
407. if ((millis() - lastTime) < MAX\_PLOSIVETIME){
408. // Увеличение счётчика взрывных звуков
409. plosiveCount++;
410. }
411. }
412. }
413. // Обработка символа в конструкции switch
414. switch (c) {
415. // Обработка символа 'f'
416. case 'f':
417. // Увеличение основного счётчика для 'f'
418. f++;
419. // Увеличение временного счётчика для 'f'
420. cf++;
421. // Выход из оператора switch для 'f'
422. break;
423. // Обработка символа 'e'
424. case 'e':
425. // Увеличение счётчика для 'e'
426. e++;
427. // Увеличение временного счётчика для 'e'
428. ce++;
429. break;
430. // Обработка символа 'o'
431. case 'o':
432. // Увеличение счётчика для 'o'
433. o++;
434. // Увеличение временного счётчика для 'o'
435. co++;
436. break;
437. // Обработка символа 'v'
438. case 'v':
439. // Увеличение счётчика для 'v'
440. v++;
441. // Увеличение временного счётчика для 'v'
442. cv++;
443. break;
444. // Обработка символа 'h'
445. case 'h':
446. // Увеличение счётчика для 'h'
447. h++;
448. // Увеличение временного счётчика для 'h'
449. ch++;
450. break;
451. // Обработка символа 's'
452. case 's':
453. // Увеличение счётчика для 's'
454. s++;
455. // Увеличение временного счётчика для 's'
456. cs++;
457. break;
458. // Обработка символа пробела
459. case ' ':
460. // Проверка активности флага ожидания пробела
461. if (expectSp != 0){
462. // Сброс флага ожидания пробела для разделения слогов
463. expectSp = 0;
464. // Фиксация времени, используемого для расчёта интервала между слогами
465. lastTime = millis();
466. }
467. break;
468. // Обработка любых остальных символов
469. default:
470. break;
471. }
472. // Периодический анализ для определения пиков и модальности
473. if ((length & PROCESS\_SKEWNESS\_TIME) == 0){
474. // Анализ временного счётчика для 'f'
475. if ((cf > prevf) & (prevf < PROCESS\_SKEWNESS\_TIME)){
476. // Обновление предыдущего значения для 'f'
477. prevf = cf;
478. // Фиксация длины слога при достижении пика 'f'
479. maxf = length;
480. // Установка текущего пика и увеличение модальности для 'f'
481. if (currPeak != 'f'){
482. currPeak = 'f';
483. modalityf++;
484. }
485. }
486. // Анализ временного счётчика для 'e'
487. if ((ce > preve) & (preve < PROCESS\_SKEWNESS\_TIME)){
488. preve = ce;
489. maxe = length;
490. if (currPeak != 'e'){
491. currPeak = 'e';
492. modalitye++;
493. }
494. }
495. // Анализ временного счётчика для 'o'
496. if ((co > prevo) & (prevo < PROCESS\_SKEWNESS\_TIME)){
497. prevo = co;
498. maxo = length;
499. if (currPeak != 'o'){
500. currPeak = 'o';
501. modalityo++;
502. }
503. }
504. // Анализ временного счётчика для 's'
505. if ((cs > prevs) & (prevs < PROCESS\_SKEWNESS\_TIME)){
506. prevs = cs;
507. maxs = length;
508. if (currPeak != 's'){
509. currPeak = 's';
510. modalitys++;
511. }
512. }
513. // Анализ временного счётчика для 'h'
514. if ((ch > prevh) & (prevh < PROCESS\_SKEWNESS\_TIME)){
515. prevh = ch;
516. maxh = length;
517. if (currPeak != 'h'){
518. currPeak = 'h';
519. modalityh++;
520. }
521. }
522. // Анализ временного счётчика для 'v' с порогом 15
523. if ((cv > prevv) & (prevv < 15)){
524. prevv = cv;
525. maxv = length;
526. if (currPeak != 'v'){
527. currPeak = 'v';
528. modalityv++;
529. }
530. }
531. // Сброс временных счётчиков для нового интервала анализа
532. cf = 0;
533. ce = 0;
534. co = 0;
535. cs = 0;
536. ch = 0;
537. cv = 0;
538. }
539. }
540. // Реализация метода для возврата указателя на массив с накопленными данными
541. int\* syllable::tointptr(){
542. // Объявление статического массива для хранения накопленных значений
543. static int matrix[20];
544. // Запись значения аккумулятора 'f' в массив
545. matrix[0] = f;
546. // Запись значения аккумулятора 'e' в массив
547. matrix[1] = e;
548. // Запись значения аккумулятора 'o' в массив
549. matrix[2] = o;
550. // Запись значения аккумулятора 'v' в массив
551. matrix[3] = v;
552. // Запись значения аккумулятора 's' в массив
553. matrix[4] = s;
554. // Запись значения аккумулятора 'h' в массив
555. matrix[5] = h;
556. // Запись значения модальности для 'f' в массив
557. matrix[6] = modalityf;
558. // Запись значения модальности для 'e' в массив
559. matrix[7] = modalitye;
560. // Запись значения модальности для 'o' в массив
561. matrix[8] = modalityo;
562. // Запись значения модальности для 'v' в массив
563. matrix[9] = modalityv;
564. // Запись значения модальности для 's' в массив
565. matrix[10] = modalitys;
566. // Запись значения модальности для 'h' в массив
567. matrix[11] = modalityh;
568. // Запись значения пика для 'f' в массив
569. matrix[12] = maxf;
570. // Запись значения пика для 'e' в массив
571. matrix[13] = maxe;
572. // Запись значения пика для 'o' в массив
573. matrix[14] = maxo;
574. // Запись значения пика для 'v' в массив
575. matrix[15] = maxv;
576. // Запись значения пика для 's' в массив
577. matrix[16] = maxs;
578. // Запись значения пика для 'h' в массив
579. matrix[17] = maxh;
580. // Запись значения длины слога в массив
581. matrix[18] = length;
582. // Запись значения счётчика взрывных звуков в массив
583. matrix[19] = plosiveCount;
584. // Возврат указателя на заполненный массив
585. return matrix;
586. }
587. #include "uspeech.h"
588. // Реализация конструктора класса statCollector для накопления статистических данных
589. statCollector::statCollector(){
590. // Инициализация переменных для количества выборок, среднего значения и моментов нулевыми значениями
591. n = 0;
592. mean = 0;
593. M2 = 0;
594. M3 = 0;
595. M4 = 0;
596. }
597. // Реализация метода для возврата среднего значения
598. int statCollector::\_mean(){
599. // Возврат рассчитанного среднего значения
600. return mean;
601. }
602. // Реализация метода для получения стандартного отклонения (в данном случае дисперсии – M2)
603. int statCollector::stdev(){
604. // Возврат значения второго центрального момента
605. return M2;
606. }
607. // Реализация метода для вычисления куртоза (измерение «остроты» распределения)
608. int statCollector::kurtosis(){
609. // Вычисление куртоза по формуле (нормированный 4-й момент минус 3)
610. int kurtosis = (n \* M4) / (M2 \* M2) - 3;
611. // Возврат вычисленного значения куртоза
612. return kurtosis;
613. }
614. // Реализация метода для вычисления асимметрии распределения
615. int statCollector::skew(){
616. // Вычисление асимметрии по формуле (нормированный 3-й момент минус 3)
617. int kurtosis = (n \* M3) / (M2 \* M2 \* M2) - 3;
618. // Возврат полученного значения асимметрии
619. return kurtosis;
620. }
621. // Реализация метода для накопления статистических данных с новым значением x
622. void statCollector::collect(int x) {
623. // Сохранение предыдущего количества выборок
624. int n1 = n;
625. // Увеличение счётчика выборок
626. n = n + 1;
627. // Вычисление разницы между новым значением и текущим средним
628. int delta = x - mean;
629. // Вычисление поправочного коэффициента для среднего
630. int delta\_n = delta / n;
631. // Вычисление квадрата поправочного коэффициента
632. int delta\_n2 = delta\_n \* delta\_n;
633. // Вычисление вспомогательного терма для обновления статистических моментов
634. int term1 = delta \* delta\_n \* n1;
635. // Обновление среднего значения
636. mean = mean + delta\_n;
637. // Обновление четвёртого момента распределения
638. M4 = M4 + term1 \* delta\_n2 \* (n \* n - 3 \* n + 3) + 6 \* delta\_n2 \* M2 - 4 \* delta\_n \* M3;
639. // Обновление третьего момента (асимметрия)
640. M3 = M3 + term1 \* delta\_n \* (n - 2) - 3 \* delta\_n \* M2;
641. // Обновление второго момента (сумма квадратов отклонений)
642. M2 = M2 + term1;
643. }
644. #include <Wire.h>
645. // Подключение библиотеки для работы с интерфейсом I²C
646. #include <LiquidCrystal\_I2C.h>
647. // Подключение заголовочного файла uspeech.h (библиотека для распознавания речи)
648. #include <uspeech.h>
649. // Определение номера пина для зелёного светодиода
650. #define ledGreen 7
651. // Определение номера пина для оранжевого светодиода
652. #define ledOrange 6
653. // Определение номера пина для белого светодиода
654. #define ledWhite 5
655. // Определение макроса MIN3 для вычисления минимального значения из трёх аргументов
656. #define MIN3(a, b, c) ((a) < (b) ? ((a) < (c) ? (a) : (c)) : ((b) < (c) ? (b) : (c)))
657. // Создание объекта voice класса signal с подключением микрофона к аналоговому пину A0
658. signal voice(A0);
659. // Создание объекта lcd класса LiquidCrystal\_I2C для работы с LCD-дисплеем по I²C
660. // Указание адреса дисплея (0x27) и размера дисплея (16 столбцов, 2 строки)
661. LiquidCrystal\_I2C lcd(0x27, 16, 2);
662. // Определение константы для размера буфера, используемого для формирования слова из фонем
663. const int BUFFER\_MAX\_PHONEMES = 32;
664. // Объявление массива для накопления распознанных символов
665. char inputString[BUFFER\_MAX\_PHONEMES];
666. // Инициализация переменной индекса текущей позиции в буфере
667. byte index = 0;
668. // Определение количества элементов в словаре (паттернов) фонем
669. const int DICT\_MAX\_ELEMNTS = 3;
670. // Объявление словаря паттернов фонем для распознавания команд
671. char dict[DICT\_MAX\_ELEMNTS][BUFFER\_MAX\_PHONEMES] = {
672. "vvvoeeeeeeeofff", // Паттерн для команды green
673. "hhhhhvoovvvvf", // Паттерн для команды orange
674. "booooooffffffff" // Паттерн для команды white
675. };
676. // Определение порогового значения для расстояния Левенштейна (для распознавания слова)
677. int LOWEST\_COST\_MAX\_THREASHOLD = 20;
678. // Объявление глобальных переменных для контроля бездействия
679. // Переменная lastCommandTime хранит время последнего изменения команды
680. unsigned long lastCommandTime = 0;
681. // Флаг idleDisplayed указывает, что состояние бездействия уже выведено на дисплей
682. bool idleDisplayed = false;
683. // Функция для вычисления длины строки до пробела или конца строки
684. int strLength(char const\* s) {
685. // Инициализация счётчика
686. int i = 0;
687. // Цикл до достижения символа окончания строки или пробела
688. while (s[i] != '\0' && s[i] != ' ')
689. ++i;
690. // Возврат вычисленной длины строки
691. return i;
692. }
693. // Функция для расчёта расстояния Левенштейна между двумя строками
694. unsigned int levenshtein(char \*s1, char \*s2) {
695. // Объявление переменных для хранения длин строк
696. unsigned int s1len, s2len, x, y, lastdiag, olddiag;
697. // Вычисление длины первой строки
698. s1len = strlen(s1);
699. // Вычисление длины второй строки
700. s2len = strlen(s2);
701. // Объявление массива для хранения промежуточных результатов (колонка матрицы)
702. unsigned int column[s1len + 1];
704. // Инициализация первого столбца матрицы (расстояния)
705. for (y = 1; y <= s1len; y++)
706. column[y] = y;
708. // Основной цикл для расчёта расстояния Левенштейна
709. for (x = 1; x <= s2len; x++) {
710. // Обновление первого элемента столбца
711. column[0] = x;
712. // Внутренний цикл для перебора символов первой строки
713. for (y = 1, lastdiag = x - 1; y <= s1len; y++) {
714. // Сохранение текущего значения для использования в следующей итерации
715. olddiag = column[y];
716. // Вычисление минимального из вариантов: удаление, вставка или замена символа
717. column[y] = MIN3(
718. column[y] + 1,
719. column[y - 1] + 1,
720. lastdiag + (s1[y - 1] == s2[x - 1] ? 0 : 1)
721. );
722. // Обновление lastdiag для следующего шага
723. lastdiag = olddiag;
724. }
725. }
727. // Возврат конечного расстояния между строками
728. return column[s1len];
729. }
730. // Функция для поиска наиболее похожего паттерна в словаре, используя алгоритм Левенштейна
731. char\* guessWord(char\* target) {
732. // Проверка на прямое совпадение указателей (для статичных строк)
733. for (int i = 0; i < DICT\_MAX\_ELEMNTS; i++) {
734. if (dict[i] == target) {
735. return dict[i];
736. }
737. }
739. // Объявление массива для хранения стоимости расстояния для каждого паттерна
740. unsigned int cost[DICT\_MAX\_ELEMNTS];
741. // Вычисление расстояния Левенштейна для каждого паттерна словаря
742. for (int j = 0; j < DICT\_MAX\_ELEMNTS; j++) {
743. cost[j] = levenshtein(dict[j], target);
744. // Вывод стоимости для отладки через Serial
745. Serial.println("dict[j]=" + String(dict[j]) + " target=" + String(target) +
746. " cost=" + String(cost[j]));
747. }
749. // Инициализация переменных для поиска паттерна с наименьшей стоимостью
750. int lowestCostIndex = -1;
751. int lowestCost = LOWEST\_COST\_MAX\_THREASHOLD;
753. // Поиск паттерна с минимальным расстоянием, удовлетворяющим порогу
754. for (int j = 0; j < DICT\_MAX\_ELEMNTS; j++) {
755. if (cost[j] < lowestCost) {
756. lowestCost = cost[j];
757. lowestCostIndex = j;
758. }
759. }
761. // Если найден подходящий паттерн, возвращается он; иначе – пустая строка
762. if (lowestCostIndex > -1) {
763. return dict[lowestCostIndex];
764. } else {
765. return "";
766. }
767. }
768. // Функция для обработки распознанной команды (слова) с выводом на дисплей и управлением светодиодами
769. void parseCommand(char\* str) {
770. // Получение наиболее похожего паттерна для введённой строки
771. char \*gWord = guessWord(str);
772. // Вывод результата распознавания в Serial
773. Serial.println("guessed: " + String(gWord));
775. // Очистка дисплея LCD
776. lcd.clear();
777. // Установка курсора в начало первой строки
778. lcd.setCursor(0, 0);
779. // Вывод метки "Word:" на LCD
780. lcd.print("Word:");
781. // Перемещение курсора на вторую строку дисплея
782. lcd.setCursor(0, 1);
784. // Если паттерн не найден, дальнейшая обработка прекращается
785. if (gWord == "") {
786. return;
787. }
788. // Обработка команды "green"
789. else if (gWord == dict[0]) {
790. // Вывод текста "green" на дисплей
791. lcd.print("green");
792. // Включение зелёного светодиода
793. digitalWrite(ledGreen, HIGH);
794. // Выключение оранжевого светодиода
795. digitalWrite(ledOrange, LOW);
796. // Выключение белого светодиода
797. digitalWrite(ledWhite, LOW);
798. }
799. // Обработка команды "orange"
800. else if (gWord == dict[1]) {
801. // Вывод текста "orange" на дисплей
802. lcd.print("orange");
803. // Выключение зелёного светодиода
804. digitalWrite(ledGreen, LOW);
805. // Включение оранжевого светодиода
806. digitalWrite(ledOrange, HIGH);
807. // Выключение белого светодиода
808. digitalWrite(ledWhite, LOW);
809. }
810. // Обработка команды "white"
811. else if (gWord == dict[2]) {
812. // Вывод текста "white" на дисплей
813. lcd.print("white");
814. // Выключение зелёного светодиода
815. digitalWrite(ledGreen, LOW);
816. // Выключение оранжевого светодиода
817. digitalWrite(ledOrange, LOW);
818. // Включение белого светодиода
819. digitalWrite(ledWhite, HIGH);
820. }
821. // Обновление времени последнего распознавания команды и флага бездействия
822. lastCommandTime = millis();
823. idleDisplayed = false;
824. }
825. void setup() {
826. // Настройка параметров uSpeech (распознавание, пороговые значения) и калибровка микрофона
827. voice.f\_enabled = true;
828. voice.minVolume = 2000;
829. voice.fconstant = 500;
830. voice.econstant = 2;
831. voice.aconstant = 4;
832. voice.vconstant = 6;
833. voice.shconstant = 10;
834. // Запуск калибровки фонового уровня микрофона
835. voice.calibrate();
837. // Инициализация последовательного порта для отладки
838. Serial.begin(9600);
840. // Настройка пинов светодиодов в режиме OUTPUT
841. pinMode(ledGreen, OUTPUT);
842. pinMode(ledOrange, OUTPUT);
843. pinMode(ledWhite, OUTPUT);
845. // Инициализация дисплея LCD
846. lcd.init();
847. // Включение подсветки дисплея
848. lcd.backlight();
849. // Очистка дисплея
850. lcd.clear();
852. // Вывод приветственного сообщения на дисплей
853. lcd.setCursor(0, 0);
854. lcd.print("Voice Recog");
855. lcd.setCursor(0, 1);
856. lcd.print("Say a color...");
858. // Последовательное мигание светодиодов в течение 3 секунд при запуске
859. unsigned long startTime = millis();
860. while (millis() - startTime < 3000) {
861. // Включение зелёного светодиода
862. digitalWrite(ledGreen, HIGH);
863. delay(200);
864. // Выключение зелёного светодиода
865. digitalWrite(ledGreen, LOW);
866. delay(100);
868. // Включение оранжевого светодиода
869. digitalWrite(ledOrange, HIGH);
870. delay(200);
871. // Выключение оранжевого светодиода
872. digitalWrite(ledOrange, LOW);
873. delay(100);
875. // Включение белого светодиода
876. digitalWrite(ledWhite, HIGH);
877. delay(200);
878. // Выключение белого светодиода
879. digitalWrite(ledWhite, LOW);
880. delay(100);
881. }
883. // Очистка дисплея после мигания и вывод приглашения к произнесению команды
884. lcd.clear();
885. lcd.setCursor(0, 0);
886. lcd.print("Say a color...");
888. // Инициализация переменных для контроля бездействия
889. lastCommandTime = millis();
890. idleDisplayed = false;
891. }
892. void loop() {
893. // Получение новой выборки звука с микрофона
894. voice.sample();
895. // Распознавание фонемы из полученной выборки
896. char p = voice.getPhoneme();
898. // Если получен символ пробела или буфер заполнен, обработка накопленного слова
899. if (p == ' ' || index >= BUFFER\_MAX\_PHONEMES) {
900. // Если накопленная строка не пустая
901. if (strLength(inputString) > 0) {
902. // Вывод распознанного слова в Serial
903. Serial.println("received: " + String(inputString));
904. // Обработка команды, соответствующей накопленному слову
905. parseCommand(inputString);
906. // Очистка буфера для следующего слова
907. inputString[0] = 0;
908. // Сброс индекса буфера
909. index = 0;
910. }
911. } else {
912. // Добавление нового символа в буфер
913. inputString[index] = p;
914. // Увеличение индекса
915. index++;
916. // Обеспечение корректного завершения строки нулевым символом
917. inputString[index] = '\0';
918. }
920. // Если с момента последнего распознавания прошло более 5 секунд, вывод сообщения idle
921. if (millis() - lastCommandTime > 5000 && !idleDisplayed) {
922. // Выключение всех светодиодов
923. digitalWrite(ledGreen, LOW);
924. digitalWrite(ledOrange, LOW);
925. digitalWrite(ledWhite, LOW);
927. // Очистка дисплея LCD
928. lcd.clear();
929. lcd.setCursor(0, 0);
930. lcd.print("Say a color...");
932. // Установка флага, что режим бездействия уже выведен
933. idleDisplayed = true;
934. }
935. }