



Diff Direction

268 lines - 26 Removals

```

1  /* Edge Impulse Arduino examples
2  * Copyright (c) 2021 EdgeImpulse Inc.
3  *
4  * Permission is hereby granted, free of charge, to any
   person obtaining a copy
5  * of this software and associated documentation files
   (the "Software"), to deal
6  * in the Software without restriction, including witho
   ut limitation the rights
7  * to use, copy, modify, merge, publish, distribute, su
   blicense, and/or sell
8  * copies of the Software, and to permit persons to who
   m the Software is
9  * furnished to do so, subject to the following conditi
   ons:
10 *
11 * The above copyright notice and this permission notic
   e shall be included in
12 * all copies or substantial portions of the Software.
13 *
14 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY O
   F ANY KIND, EXPRESS OR
15 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
   OF MERCHANTABILITY,
16 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT
   T. IN NO EVENT SHALL THE
17 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAI
   M, DAMAGES OR OTHER
18 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
   OTHERWISE, ARISING FROM,
19 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
   OR OTHER DEALINGS IN THE
20 * SOFTWARE.
21 */
22
23 // If your target is limited in memory remove this macr
   o to save 10K RAM
24 #define EIDSP_QUANTIZE_FILTERBANK 0
25
26 /**
27 * Define the number of slices per model window. E.g. a
   model window of 1000 ms
28 * with slices per model window set to 4. Results in a
   slice size of 250 ms.
29 * For more info: https://docs.edgeimpulse.com/docs/con
   tinuous-audio-sampling
30

```

332 lines + 89 Additions

```

1  /* Edge Impulse Arduino examples
2  * Copyright (c) 2021 EdgeImpulse Inc.
3  *
4  * Permission is hereby granted, free of charge, to any
   person obtaining a copy
5  * of this software and associated documentation files
   (the "Software"), to deal
6  * in the Software without restriction, including witho
   ut limitation the rights
7  * to use, copy, modify, merge, publish, distribute, su
   blicense, and/or sell
8  * copies of the Software, and to permit persons to who
   m the Software is
9  * furnished to do so, subject to the following conditi
   ons:
10 *
11 * The above copyright notice and this permission notic
   e shall be included in
12 * all copies or substantial portions of the Software.
13 *
14 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY O
   F ANY KIND, EXPRESS OR
15 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
   OF MERCHANTABILITY,
16 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT
   T. IN NO EVENT SHALL THE
17 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAI
   M, DAMAGES OR OTHER
18 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
   OTHERWISE, ARISING FROM,
19 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
   OR OTHER DEALINGS IN THE
20 * SOFTWARE.
21 */
22 #include <LiquidCrystal_I2C.h>
23 #include <Wire.h>
24
25 LiquidCrystal_I2C lcd(0x27, 16, 2);
26
27 // If your target is limited in memory remove this macr
   o to save 10K RAM
28 #define EIDSP_QUANTIZE_FILTERBANK 0
29
30 /**
31 * Define the number of slices per model window. E.g. a
   model window of 1000 ms
32 * with slices per model window set to 4. Results in a
   slice size of 250 ms.
33 * For more info: https://docs.edgeimpulse.com/docs/con
   tinuous-audio-sampling
34

```

```

    */
31 //define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 3
32
33 /*
34 ** NOTE: If you run into TFLite arena allocation issu
e.
35 **
36 ** This may be due to may dynamic memory fragmentatio
n.
37 ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in
boards.local.txt (create
38 ** if it doesn't exist) and copy this file to
39 ** `<ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed
_core>/<core_version>`.
40 **
41 ** See
42 ** (https://support.arduino.cc/hc/en-us/articles/36001
2076960-Where-are-the-installed-cores-located-)
43 ** to find where Arduino installs cores on your machin
e.
44 **
45 ** If the problem persists then there's not enough mem
ory for this model and application.
46 */
47
48 /* Includes -----
----- */
49 #include <PDM.h>
50 #include <FYP_Direction_inferencing.h>
51
52 /** Audio buffers, pointers and selectors */
53 typedef struct {
54     signed short *buffers[2];
55     unsigned char buf_select;
56     unsigned char buf_ready;
57     unsigned int buf_count;
58     unsigned int n_samples;
59 } inference_t;
60
61 static inference_t inference;
62 static volatile bool record_ready = false;
63 // static signed short *sampleBuffer;
64 static signed short sampleBuffer[2048];
65 static bool debug_nn = false; // Set this to true to se
e e.g. features generated from the raw signal
66 static int print_results = -(EI_CLASSIFIER_SLICES_PER_M
ODEL_WINDOW);
67
68 /**
69 * @brief    Arduino setup function
70 */

```

```

    */
35 //define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 3
36
37 /*
38 ** NOTE: If you run into TFLite arena allocation issu
e.
39 **
40 ** This may be due to may dynamic memory fragmentatio
n.
41 ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in
boards.local.txt (create
42 ** if it doesn't exist) and copy this file to
43 ** `<ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed
_core>/<core_version>`.
44 **
45 ** See
46 ** (https://support.arduino.cc/hc/en-us/articles/36001
2076960-Where-are-the-installed-cores-located-)
47 ** to find where Arduino installs cores on your machin
e.
48 **
49 ** If the problem persists then there's not enough mem
ory for this model and application.
50 */
51
52 /* Includes -----
----- */
53 #include <FYP_Direction_inferencing.h>
54
55 #include "RPC.h"
56
57 /* Mic Setting */
58 int sound_analog = A6;
59
60 int Pass;
61 int outtoM4 = D5;
62
63 int setVar(int a) {
64     Pass = (int)a;
65     return a;
66 }
67
68
69 /** Audio buffers, pointers and selectors */
70 typedef struct {
71     signed short *buffers[2];
72     unsigned char buf_select;
73     unsigned char buf_ready;
74     unsigned int buf_count;
75     unsigned int n_samples;
76 } inference_t;
77
78 static inference_t inference;
79 static volatile bool record_ready = false;
80 //static signed short *sampleBuffer;
81 static signed short sampleBuffer[2048];
82 static bool debug_nn = false; // Set this to true to se
e e.g. features generated from the raw signal
83 static int print_results = -(EI_CLASSIFIER_SLICES_PER_M
ODEL_WINDOW);
84
85 /**
86 * @brief    Arduino setup function
87 */

```

```

71 void setup()
72 {
73     // put your setup code here, to run once:
74     Serial.begin(115200);
75
76     Serial.println("Edge Impulse Inferencing Demo");
77     // summary of inferencing settings (from model_meta
data.h)
78     ei_printf("Inferencing settings:\n");
79     ei_printf("\tInterval: ");
80     ei_printf_float((float)EI_CLASSIFIER_INTERVAL_MS);
81     ei_printf(" ms.\n");
82     ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_I
NPUT_FRAME_SIZE);
83     ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIE
R_RAW_SAMPLE_COUNT / 16);
84     ei_printf("\tNo. of classes: %d\n", sizeof(ei_class
ifier_inferencing_categories) /
85                                     sizeof(ei_c
lassifier_inferencing_categories[0]));
86
87     run_classifier_init();
88     if (microphone_inference_start(EI_CLASSIFIER_SLICE_
SIZE) == false) {
89         ei_printf("ERR: Failed to setup audio sampling
\r\n");
90         return;
91     }
92 }
93
94 /**
95  * @brief    Arduino main function. Runs the inferenc
ing loop.
96  */
97 void loop()
98 {
99     bool m = microphone_inference_record();
100     if (!m) {
101         ei_printf("ERR: Failed to record audio...\n");
102         return;
103     }
104
105     signal_t signal;
106     signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
107     signal.get_data = &microphone_audio_signal_get_dat
a;
108     ei_impulse_result_t result = {0};
109
110     EI_IMPULSE_ERROR r = run_classifier_continuous(&sig

```

```

88 void setup()
89 {
90     // put your setup code here, to run once:
91     LL_RCC_ForceCM4Boot();
92     RPC.begin();
93     Serial.begin(115200);
94     pinMode(outtoM4, OUTPUT);
95     digitalWrite(outtoM4, LOW);
96     RPC.bind("setVar", setVar); // do these have to be
the same?
97     Pass = 0;
98
99     Serial.println("Edge Impulse Inferencing Demo");
100     lcd.begin();
101     lcd.backlight();
102     // summary of inferencing settings (from model_meta
data.h)
103     ei_printf("Inferencing settings:\n");
104     ei_printf("\tInterval: ");
105     ei_printf_float((float)EI_CLASSIFIER_INTERVAL_MS);
106     ei_printf(" ms.\n");
107     ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_I
NPUT_FRAME_SIZE);
108     ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIE
R_RAW_SAMPLE_COUNT / 16);
109     ei_printf("\tNo. of classes: %d\n", sizeof(ei_class
ifier_inferencing_categories) /
110                                     sizeof(ei_c
lassifier_inferencing_categories[0]));
111
112     run_classifier_init();
113     if (microphone_inference_start(EI_CLASSIFIER_SLICE_
SIZE) == false) {
114         ei_printf("ERR: Failed to setup audio sampling
\r\n");
115         return;
116     }
117 }
118
119 /**
120  * @brief    Arduino main function. Runs the inferenc
ing loop.
121  */
122 void loop()
123 {
124     if (Pass == 1){
125         lcd.setCursor(0, 0);
126         lcd.print("Stage: 2 ");
127         lcd.setCursor(0, 1);
128         lcd.print("Wait for Command");
129         digitalWrite(outtoM4, HIGH);
130     }
131     bool m = microphone_inference_record();
132     if (!m) {
133         ei_printf("ERR: Failed to record audio...\n");
134         return;
135     }
136
137     signal_t signal;
138     signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
139     signal.get_data = &microphone_audio_signal_get_dat
a;
140     ei_impulse_result_t result = {0};
141
142     EI_IMPULSE_ERROR r = run_classifier_continuous(&sig

```

```

    nal, &result, debug_nn);
111     if (r != EI_IMPULSE_OK) {
112         ei_printf("ERR: Failed to run classifier (%d
\n", r);
113         return;
114     }
115
116     if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MO
DEL_WINDOW)) {
117         // print the predictions
118         ei_printf("Predictions ");
119         ei_printf("(DSP: %d ms., Classification: %d m
s., Anomaly: %d ms.)",
120             result.timing.dsp, result.timing.classifica
tion, result.timing.anomaly);
121         ei_printf(": \n");
122         for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_CO
UNT; ix++) {
123             ei_printf("    %s: ", result.classification
[ix].label);
124             ei_printf_float(result.classification[ix].v
alue);
125             ei_printf("\n");
126
127     }
128
129     #if EI_CLASSIFIER_HAS_ANOMALY == 1
130         ei_printf("    anomaly score: ");

```

```

    nal, &result, debug_nn);
142     if (r != EI_IMPULSE_OK) {
143         ei_printf("ERR: Failed to run classifier (%d
\n", r);
144         return;
145     }
146
147     if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MO
DEL_WINDOW)) {
148         // print the predictions
149         ei_printf("Predictions ");
150         ei_printf("(DSP: %d ms., Classification: %d m
s., Anomaly: %d ms.)",
151             result.timing.dsp, result.timing.classifica
tion, result.timing.anomaly);
152         ei_printf(": \n");
153         for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_CO
UNT; ix++) {
154             ei_printf("    %s: ", result.classification
[ix].label);
155             ei_printf_float(result.classification[ix].v
alue);
156             ei_printf("\n");
157         }
158
159         for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_CO
UNT; ix++) {
160             if ( (ix != 2) && (result.classification[i
x].value >= 0.5)) {
161                 Pass=0;
162                 digitalWrite(outtoM4, LOW);
163                 digitalWrite(outtoM4, 0);
164                 switch (ix) {
165                     case 0:
166                         lcd.setCursor(0, 1);
167                         lcd.print("Down      ");
168                         break;
169
170                     case 1:
171                         lcd.setCursor(0, 1);
172                         lcd.print("Left      ");
173                         break;
174
175                     case 3:
176                         lcd.setCursor(0, 1);
177                         lcd.print("Right     ");
178                         break;
179
180                     case 4:
181                         lcd.setCursor(0, 1);
182                         lcd.print("Up        ");
183                         break;
184
185                     default:
186                         break;
187                 }
188                 Serial.println("Transforming to M4...");
189                 delay(100);
190             }
191         }
192
193     #if EI_CLASSIFIER_HAS_ANOMALY == 1
194         ei_printf("    anomaly score: ");
195

```

```

        ei_printf_float(result.anomaly);
131     ei_printf("\n");
132 #endif
133
134     print_results = 0;
135 }

```

```

136 }
137
138 /**
139  * @brief      PDM buffer full callback
140  *             Copy audio data to app buffers
141  */
142 static void pdm_data_ready_inference_callback(void)
143 {
144     int bytesAvailable = PDM.available();
145
146     // read into the sample buffer
147     int bytesRead = PDM.read((char *)&sampleBuffer[0],
148                             bytesAvailable);

```

```

148
149     if ((inference.buf_ready == 0) && (record_ready ==
150         true)) {
151         for(int i = 0; i < bytesRead>>1; i++) {
152             inference.buffers[inference.buf_select][inf
153                 erence.buf_count++] = sampleBuffer[i];
154
155             if (inference.buf_count >= inference.n_samp
156                 les) {
157                 inference.buf_select ^= 1;
158                 inference.buf_count = 0;
159                 inference.buf_ready = 1;
160                 break;
161             }
162         }
163     }
164 }
165
166 /**
167  * @brief      Init inferencing struct and setup/start
168  *             PDM
169  * @param[in]  n_samples  The n samples
170  * @return     { description_of_the_return_value }
171  */
172 static bool microphone_inference_start(uint32_t n_samp
173     les)
174 {
175     inference.buffers[0] = (signed short *)malloc(n_sam

```

```

        ei_printf_float(result.anomaly);
196     ei_printf("\n");
197 #endif
198
199     print_results = 0;
200 }
201 }
202 else{
203
204     digitalWrite(outtoM4, LOW);
205     lcd.setCursor(0, 0);
206     lcd.print("Stage: 1      ");
207     while (RPC.available()) {
208         Serial.write(RPC.read()); // check if the M4 has
209         sent an RPC println
210     }
211 }
212 }
213
214 /**
215  * @brief      PDM buffer full callback
216  *             Copy audio data to app buffers
217  */
218 static void pdm_data_ready_inference_callback(void)
219 {
220     for (int i=0; i<2048; i++){
221
222         sampleBuffer[i] = analogRead(sound_analog);
223
224     }
225
226     size_t bytesRead;
227     bytesRead = 2048;
228
229     if ((inference.buf_ready == 0) && (record_ready ==
230         true)) {
231         for(int i = 0; i < bytesRead; i++) {
232             inference.buffers[inference.buf_select][inf
233                 erence.buf_count++] = sampleBuffer[i];
234
235             if (inference.buf_count >= inference.n_samp
236                 les) {
237                 inference.buf_select ^= 1;
238                 inference.buf_count = 0;
239                 inference.buf_ready = 1;
240                 break;
241             }
242         }
243     }
244 }
245
246 /**
247  * @brief      Init inferencing struct and setup/start
248  *             PDM
249  * @param[in]  n_samples  The n samples
250  * @return     { description_of_the_return_value }
251  */
252 static bool microphone_inference_start(uint32_t n_samp
253     les)
254 {
255     inference.buffers[0] = (signed short *)malloc(n_sam

```

```
    ples * sizeof(signed short));
173
174     if (inference.buffers[0] == NULL) {
175         return false;
176     }
177
178     inference.buffers[1] = (signed short *)malloc(n_sam
ples * sizeof(signed short));
179
```

```
    ples * sizeof(signed short));
253
254     if (inference.buffers[0] == NULL) {
255         return false;
256     }
257
258     inference.buffers[1] = (signed short *)malloc(n_sam
ples * sizeof(signed short));
259
```



