

# Big brain matrix eigenvalue lightspeed fourier transform for the great good solar light

## A 4-bit dance in frequency space

Pushkar Mohile (180260027)<sup>1</sup> and Sankalp Gambhir (180260032)<sup>1</sup>

Indian Institute of Technology, Bombay

**Abstract.** Identification of sounds has immense applications in the embedded systems space, ranging from simple detection of sounds to complete voice transcription. Being able to do this on low power devices is an area of active interest. We present a new approach to this problem involving bypassing a complete Fourier Transform and approximating its results using a cross-correlation based approach pruning a tree of (preset) frequencies. Our method returns present frequencies with reasonable accuracy whilst maintaining the speed expected of such an embedded system.

## 1 Project Details

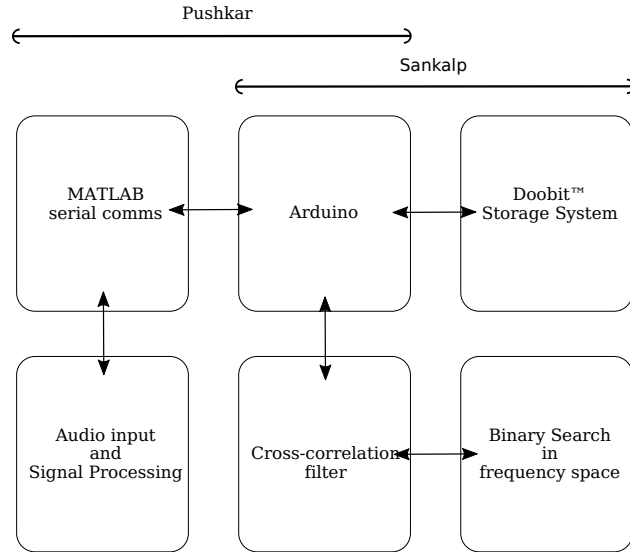


Fig. 1: Block diagram and work distribution.

## **2 Main components and Inventory**

The project mainly revolves around processing onboard the Arduino, so not much hardware is required:

- Arduino
- USB Cable for serial communication
- Mic – replaced by laptop with MATLAB here
- LEDs for displaying frequencies (optional extra)

## **3 Results**

not good

screenshots of code compiled and running on arduino

photo video uploads

## Appendix A Arduino Code

Here is the full code sent to the Arduino, verbatim.

```
1 // main.ino
2
3 #include <cmath>           // for sqrt
4 #include <vector>          // for vector...
5 #include <algorithm>       // the OTHER std::move
6 #include <cstdint>         // ALL the ints
7 #include <cassert>        // testing
8
9 #define SIZE 100
10
11 // custom typing -- forward declaration
12 struct doobit;
13
14 // functions and constants
15 int correlation(signal*, signal*);
16 int crosscorrelation(signal*, signal*, int = 0);
17 std::vector<float> checkcorr(signal*, std::vector<float>);
18
19 std::vector<float> freq = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8};
20 const float corr_threshold = 0.1;
21
22 bool recording = false;
23
24 typedef doobit signal;
25 signal f[SIZE];
26
27 void setup(){
28     Serial.begin(9600);
29 }
30
31 void loop(){
32
33     if(recording){
34         // recording data
35     }
36     else{
37         // calculate with the data
38
39         // f coming from data
40         //auto f_gen = [](int x){
41         //    return (7.0*(sin(0.3 * x) + 4*sin(0.5 * x) + sin
42         //    (0.8 * x + 0.6))/6.0);
43         //};
44
45         //for(int i = 0; i < SIZE; i++){
46         //    f[i] = signal(f_gen(2*i), f_gen(2*i+1));
47         //}
48
49         auto wpresent = checkcorr(f, freq);
```

```

49         for(auto w : wpresent){
50             printf("%f ", w);
51         }
52     }
53
54     printf("\n");
55 }
56
57     return;
58 }
59
60 // definitions
61
62 // bit mask storage
63 struct doobit{
64     uint_fast8_t data;
65
66     doobit(int8_t x = 0, int8_t y = 0){ // handles all our casts too
67         this->storelow(x);
68         this->storehigh(y);
69     }
70
71     void storelow(int8_t);
72     void storehigh(int8_t);
73
74     int8_t getlow();
75     int8_t gethigh();
76
77     int16_t operator*(doobit& b){
78         auto highprod = this->gethigh() * b.gethigh();
79         auto lowprod = this->getlow() * b.getlow();
80         return (highprod + lowprod);
81     }
82 };
83
84 void doobit::storelow(int8_t x){
85     this->data &= 0b11110000; // clear for storage
86
87     x += 7; // remove signed component
88     assert((!(x & 0b11110000)) && "doobit range violation");
89
90     this->data |= x;
91 }
92
93 void doobit::storehigh(int8_t x){
94     this->data &= 0b00001111; // clear for storage
95
96     x += 7; // remove signed component
97     assert((!(x & 0b11110000)) && "doobit range violation");
98
99     this->data |= (x << 4);
100 }

```

```

101
102 int8_t doobit::getlow(){
103     int8_t x = (data & 0b00001111); // bitmask
104     return (x-7); // reinsert sign
105 }
106
107 int8_t doobit::gethigh(){
108     int8_t x = ((data & 0b11110000) >> 4); // bitmask and shift
109     return (x-7); // reinsert sign
110 }
111
112
113 int correlation(signal* f, signal* g){
114     int sum = 0;
115
116     for(int i = 0; i < SIZE; i++){
117         sum += f[i] * g[i];
118     }
119     return sum;
120 }
121
122 int crosscorrelation(signal* f, signal* g, int m){
123     int sum = 0;
124
125     if(m >= 0){
126         for(int i = 0; i < SIZE - m; i++){
127             sum += f[i] * g[i+m];
128         }
129         for(int i = 0; i < m; i++){
130             sum += f[i+SIZE-m] * g[i];
131         }
132     }
133     else{
134         m = -m;
135         for(int i = 0; i < m; i++){
136             sum += f[i] * g[i+SIZE-m];
137         }
138         for(int i = m; i < SIZE; i++){
139             sum += f[i] * g[i-m];
140         }
141     }
142     return sum;
143 }
144
145 std::vector<float> checkcorr(signal* f, std::vector<float> wlist){
146
147     if(wlist.size() == 0) return wlist;
148
149     float maxcorr = -1;
150
151     auto g_gen = [wlist](int x){
152         float sum = 0;

```

```

153         for(auto w : wlist){
154             sum += sin(w*x);
155         }
156         return 7.0*sum/float(wlist.size());
157     };
158
159     auto g = new signal[SIZE];
160
161     for(int i = 0; i < SIZE; i++){
162         g[i] = signal(g_gen(2*i), g_gen(2*i + 1));
163     }
164
165     auto norm_coeff = sqrt((correlation(f, f) * correlation(g, g)));
166     norm_coeff = 1/norm_coeff;
167
168     for(int i = -SIZE+1; i < SIZE; i++){
169         auto corr = crosscorrelation(f, g, i);
170         maxcorr = maxcorr > corr ? maxcorr : corr;
171     }
172
173     // clean memory just in case it isn't deallocated
174     // before recursion else we run over quota
175     delete[] g;
176
177     if(maxcorr*norm_coeff < corr_threshold) return {};
178
179     if(wlist.size() == 1) return wlist;
180
181     auto wl = checkcorr(f, std::vector<float>(wlist.begin(), wlist.begin()
182         + (wlist.size()/2)));
183     auto wr = checkcorr(f, std::vector<float>(wlist.begin() + (wlist.
184         size()/2), wlist.end()));
185
186     std::move(wr.begin(), wr.end(), std::back_inserter(wl));
187
188     return wl;
189 }

```

## Appendix B MATLAB

The code used inside MATLAB to record and send audio to the Arduino.

```
1 function carr = getAudioAndSend2()
2 a = audiorecorder(10000,8,1);
3 recordblocking(a,1); %200 data points recorded
4 carr = getaudiodata(a,'int8')
5 Ard = serial("COM4","BaudRate",9600);
6 fopen(Ard);
7 for i = 1:200
8     fprintf(Ard,'%d\n',int2str(carr(i+1500)));
9 end
10 fscanf(Ard)
11 fclose(Ard);
12 end
```