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

Pushkar Mohile (180260027)[1] and Sankalp Gambhir (180260032)[1]

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

big brain block diagram
- many group member label

2 Main components and Invectory

LEDS and shiz

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.

```
// main.ino

#include <cmath>        // for sqrt
#include <vector>       // for vector...
#include <algorithm>    // the OTHER std::move
#include <cstdint>      // ALL the ints
#include <cassert>      // testing

#define SIZE 100

// custom typing -- forward declaration
struct doobit;

// functions and constants
int correlation(signal*, signal*);
int crosscorrelation(signal*, signal*, int = 0);
std::vector<float> checkcorr(signal*, std::vector<float>);

std::vector<float> freq = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8};
const float corr_threshold = 0.1;

bool recording = false;

typedef doobit signal;
signal f[SIZE];

void setup(){
    Serial.begin(9600);
}

void loop(){

    if(recording){
        // recording data
    }
    else{
        // calculate with the data

        // f coming from data
        //auto f_gen = [](int x){
        //            return (7.0*(sin(0.3 * x) + 4*sin(0.5 * x) + sin(0.8 * x + 0.6))/6.0);
        //            };

        //for(int i = 0; i < SIZE; i++){
        //    f[i] = signal(f_gen(2*i), f_gen(2*i+1));
        //}

        auto wpresent = checkcorr(f, freq);
```

```
        for(auto w : wpresent){
            printf("%f ", w);
        }

        printf("\n");
    }

    return;
}

// definitions

// bit mask storage
struct doobit{
  uint_fast8_t data;

  doobit(int8_t x = 0, int8_t y = 0){ // handles all our casts too
    this->storelow(x);
    this->storehigh(y);
  }

  void storelow(int8_t);
  void storehigh(int8_t);

  int8_t getlow();
  int8_t gethigh();

  int16_t operator*(doobit& b){
    auto highprod = this->gethigh() * b.gethigh();
    auto lowprod = this->getlow() * b.getlow();
    return (highprod + lowprod);
  }
};

void doobit::storelow(int8_t x){
  this->data &= 0b11110000;  // clear for storage

  x += 7; // remove signed component
  assert((!(x & 0b11110000)) && "doobit range violation");

  this->data |= x;
}

void doobit::storehigh(int8_t x){
  this->data &= 0b00001111;  // clear for storage

  x += 7; // remove signed component
  assert((!(x & 0b11110000)) && "doobit range violation");

  this->data |= (x << 4);
}
```

```
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;
```

```
          for(auto w : wlist){
            sum += sin(w*x);
          }
          return 7.0*sum/float(wlist.size());
        };

  auto g = new signal[SIZE];

  for(int i = 0; i < SIZE; i++){
    g[i] = signal(g_gen(2*i), g_gen(2*i + 1));
  }

  auto norm_coeff = sqrt((correlation(f, f) * correlation(g, g)));
  norm_coeff = 1/norm_coeff;

  for(int i = -SIZE+1; i < SIZE; i++){
    auto corr = crosscorrelation(f, g, i);
    maxcorr = maxcorr > corr ? maxcorr : corr;
  }

  // clean memory just in case it isn't deallocated
  // before recursion else we run over quota
  delete[] g;

  if(maxcorr*norm_coeff < corr_threshold) return {};

  if(wlist.size() == 1) return wlist;

  auto wl = checkcorr(f, std::vector<float>(wlist.begin(), wlist.begin
    () + (wlist.size()/2)));
  auto wr = checkcorr(f, std::vector<float>(wlist.begin() + (wlist.
    size()/2), wlist.end()));

  std::move(wr.begin(), wr.end(), std::back_inserter(wl));

  return wl;
}
```