Modelling Correlation Effect

Roberta Roberts, Glyn Humphreys, Dimitri Zervas and Fionn Murtagh

Abstract

We describe the requirements for the modelling of the correlation effect, the detailed algorithm, the program used, and sample output produced.

Contents

1. Introduction

It is important to note whether there is any difference in the analyses between detection of targets in correlated versus uncorrelated noise. In the human data, this is the strongest result. We definitely need to be able to capture this in order to reflect (simulate) human performance. If a difference between correlated and uncorrelated noise is not emerging, we need to think how further about how we can find this.

The second most obvious step at this stage would be to start having the correlated noise offset in time by different time steps, and to see if there remains improved detection for targets in correlated noise (this obviously depends on there being an efefct in the first place).

In conjuction with the modelling algorithm, below, there are text files containing the thresholds for the two experiments. Each file is labelled with the appropriate experiment.

The columns in file Experiment 1 represent thresholds in the positively correlated, negatively correlated and uncorrelated condition. Each row represents a value for each subject.The columns in file Experiment 2, represent thresholds in the positively correlated: lag 0, positively correlated: lag 200 ms, positively correlated: lag 400, and the uncorrelated condition. Each row represents a value for each subject.

2. Algorithm for Modelling Correlation Effect

Part 1: Basic Correlation Effect

1. For a Given Subject
    A. For the Uncorrelated Condition
        i. Use the feature height giving 79 % correct performance in the uncorrelated condition

a) Select 1 pattern for the left hand and 1 pattern for the right hand (at random)
b) Add the feature (at the 79% correct threshold intensity) to one of the hands (selected at random)
c) Add level N of internal noise
d) Take the difference between the left and right hands at each time step
e) Find the maximum value in this difference signal (possibly around the 9^{th} – 12^{th} time steps).
f) Make a left or right response depending on the sign of this difference
g) Repeat steps a) – f) 100 times and find percent correct
If the % correct > 79% then increase internal noise N
If the % correct < 79% then decrease internal noise N
h) Repeat step vii until % correct = 79%

B. For the Correlated Condition

i. Use the feature height giving 79 % correct performance in the **uncorrelated** condition

a) Select 1 pattern (at random) and apply it to both the left and right hands
b) Add the feature (at the 79% correct threshold intensity) to one of the hands (selected at random)
c) Add level N of internal noise
d) Take the difference between the left and right hands at each time step
e) Find the maximum value in this difference signal (possibly around the 9^{th} – 12^{th} time steps).
f) Make a left or right response depending on the sign of this difference
g) Repeat steps a) – f) 100 times and find percent correct

PREDICT:
that percent correct in the correlated condition should be > 79%.
Alternatively;
if internal noise was the primary determiner of performance; using level of internal noise found in step A and the actual threshold for **positively correlated** noise, the model should perform at 79% correct.

2. Repeat Step 1 for the correlated and uncorrelated noise conditions for all subjects in Experiments 1 and 2

Part 2: Phase Lag Effects

1. Using the procedure set out above vary the differencing procedure to use:
   a. A running average
   b. A running weighted average etc..

2. Do step 1 for each subject to determine their correlation advantage at each lag (200, 400 ms)

To better account for individual differences

1. In part 1, vary the samping window for the target
2. Add noise to the differencing process – to vary the size of the correlation benefit shown by individual subjects.

3. File coreffect.h

```
#ifndef __COREFFECT_H__
#define __COREFFECT_H__

// I N C L U D E S /////////////////////////////////////////////////////////////

#include <cmath>
#include <cstdlib>
#include <ctime>
#include <climits>
#include <cfloat>
#include <vector>
#include <iostream>
#include <fstream>

using namespace std;

// D E F I N E S ///////////////////////////////////////////////////////////////

// M A C R O S /////////////////////////////////////////////////////////////////

#define ranf() ((double) rand() / (double) RAND_MAX)
#define RandRange(a,b) ( (a) + (rand()%((b)-(a)+1)))

// C O N S T A N T S ///////////////////////////////////////////////////////////

const double DEFAULT_FEATURE_HEIGHT = 5.39;

const int PATTERN_PART_START = 9 - 1;
const int PATTERN_PART_END = 12 - 1;
const int PATTERN_PART_SIZE = PATTERN_PART_END -
PATTERN_PART_START + 1;

const double DEFAULT_NOISE_MEAN = 0;
const double DEFAULT_NOISE_STD_DEV = 1.0;

// T Y P E S ///////////////////////////////////////////////////////////////////

typedef vector<double> pattern_t;

typedef struct {
  double v[3];
} heights_t;

typedef enum { POSITIVE_CORRELATION, NEGATIVE_CORRELATION,
UNCORRELATED } correlation_t;
```

```
// S T R U C T U R E S ////////////////////////////////////////////////////
// E X T E R N A L S ////////////////////////////////////////////////////////
// P R O T O T Y P E S //////////////////////////////////////////////////////

double rand_gaussian(double s, double m);
void copy_pattern(pattern_t& pattern_out, const pattern_t& pattern_in);
void add_height(pattern_t& pattern, double height);
void add_noise(pattern_t& pattern, double sigma);
double calc_mean(const pattern_t& numbers);
double square(double num);
double standard_deviation(const pattern_t& numbers);
double pearson(const pattern_t& x, const pattern_t& y);

//double correlation_effect(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type);
double correlation_effect(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type, double s);
double correlation_effect2(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type, double s);
double correlation_effect3(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type, double s);
bool load_pattern(pattern_t& pattern, const char filename[]);
void print_pattern(const pattern_t& pattern);

#endif // __COREFFECT_H__
```

4. File coreffect.cpp

```
// I N C L U D E S ///////////////////////////////////////////////////////////

#include "coreffect.h"

// G L O B A L S  ///////////////////////////////////////////////////////////


// E X T E R N A L S ////////////////////////////////////////////////////////


// F U N C T I O N S ////////////////////////////////////////////////////////

double rand_gaussian(double s, double m)
{
  static int pass = 0;
  static double y2;
  double x1, x2, w, y1;

  if (pass)
  {
    y1 = y2;
  } else  {

    do {
      x1 = 2.0f * ranf () - 1.0f;
      x2 = 2.0f * ranf () - 1.0f;
      w = x1 * x1 + x2 * x2;
    } while (w >= 1.0f);

    w = sqrt (-2.0 * log (w) / w);
    y1 = x1 * w;
    y2 = x2 * w;
  }
  pass = !pass;

  return ( (y1 * s + m));
}

//////////////////////////////////////////////////////////////////////////////

void copy_pattern(pattern_t& pattern_out, const pattern_t& pattern_in) {

 if (!pattern_out.empty()) {
  pattern_out.clear();
 }

 pattern_out.resize(pattern_in.size());
```

```
  for (size_t i = 0; i < pattern_in.size(); i++) {
    pattern_out[i] = pattern_in[i];
  }

} // end copy_pattern

////////////////////////////////////////////////////////////////////////////

void copy_pattern(pattern_t& pattern_out, const pattern_t& pattern_in, int start, int
end) {

  if (!pattern_out.empty()) {
    pattern_out.clear();
  }

  pattern_out.resize(end - start + 1);

  for (size_t i = start; i <= end; i++) {
    pattern_out[i-start] = pattern_in[i];
  }

} // end copy_pattern

////////////////////////////////////////////////////////////////////////////

bool patterns_equal(const pattern_t& pattern1, const pattern_t& pattern2) {

  for (size_t i = 0; i < pattern1.size(); i++) {
    if (pattern1[i] != pattern2[i]) {
      return false;
    }
  }

  return true;
} // end copy_pattern

////////////////////////////////////////////////////////////////////////////

void add_height(pattern_t& pattern, double height) {
  for (size_t i = PATTERN_PART_START; i <= PATTERN_PART_END; i++) {
    pattern[i] += height;
  }
}

////////////////////////////////////////////////////////////////////////////

void add_noise(pattern_t& pattern, double sigma) {
  for (size_t i = 0; i < pattern.size(); i++) {
    pattern[i] += rand_gaussian(sigma, DEFAULT_NOISE_MEAN);
  }
```

```
}

////////////////////////////////////////////////////////////////////////////

double pattern_diff(const pattern_t& pattern1, const pattern_t& pattern2) {

  double max = DBL_MIN;

  for (size_t i = PATTERN_PART_START; i <= PATTERN_PART_END; i++) {

    double diff = pattern1[i] - pattern2[i];

    if (fabs(max) < fabs(diff)) {
      max = diff;
    }
  }

  return max;
}

////////////////////////////////////////////////////////////////////////////

bool pattern_diff2(const pattern_t& pattern1, const pattern_t& pattern2) {

  if (pattern1.size() != pattern2.size()) {
    return false;
  }

  pattern_t patterndiff;
  patterndiff.resize(pattern1.size());

  double max_diff_overall = DBL_MIN;

  for (size_t i = 0; i < pattern1.size(); i++) {

    patterndiff[i] = fabs(pattern1[i] - pattern2[i]);

    if (max_diff_overall < patterndiff[i]) {
      max_diff_overall = patterndiff[i];
    }
  }


  double max_part_diff = DBL_MIN;

  for (size_t i = PATTERN_PART_START; i <= PATTERN_PART_END; i++) {

    if (max_part_diff < patterndiff[i]) {
      max_part_diff = patterndiff[i];
    }
```

```
  }

  if (max_part_diff == max_diff_overall) {
    return true;
  }

  return false;
}

////////////////////////////////////////////////////////////////////////////

double calc_mean(const pattern_t& numbers) {

  double sum = 0;
  for (size_t i = 0; i < numbers.size(); i++) {
    sum += numbers[i];
  }

  return (sum / (double) numbers.size());
}

////////////////////////////////////////////////////////////////////////////

double square(double num) {
  return (num*num);
}

////////////////////////////////////////////////////////////////////////////

double standard_deviation(const pattern_t& numbers) {

  double mean = calc_mean(numbers);

  double sum = 0;
  for (size_t i = 0; i < numbers.size(); i++) {
    sum += square(numbers[i] - mean);
  }

  return sqrt(sum / (double) (numbers.size() - 1));
}

////////////////////////////////////////////////////////////////////////////

double pearson(const pattern_t& x, const pattern_t& y) {

  double n = (double)x.size();

  double sxy = 0;

  double sx = 0;
```

```
  double sx2 = 0;

  double sy = 0;
  double sy2 = 0;

  for (int i=0; i<n; i++) {

    sxy += (x[i] * y[i]);

    sx += x[i];
    sx2 += (x[i]*x[i]);

    sy += y[i];
    sy2 += (y[i]*y[i]);
  }


  double a = sxy - ((sx*sy) / n);

  double b = sx2 - ((sx*sx) / n);

  double c = sy2 - ((sy*sy) / n);

  double r = a / sqrt(b*c);

  return r;
}

////////////////////////////////////////////////////////////////////////////
/*
double correlation_effect(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type) {

  double k = 0;
  int min_diff = INT_MAX;
  int wanted_detections = 0;
  double wanted_sigma = 0;

  for (int j = 0; j < 10000; j++) {


    int count_detections = 0;

    double sigma = 0.0;

    for (int i = 0; i < 100; i++) {

      pattern_t left;
      pattern_t right;
```

```
if (correlation_type == POSITIVE_CORRELATION) {
  int left_pattern_index = RandRange(0, patterns.size()-1);
  copy_pattern(left, patterns[left_pattern_index]);
  copy_pattern(right, left);
}
else
if (correlation_type == NEGATIVE_CORRELATION) {
  int half_size = patterns.size() / 2;
  int left_pattern_index = RandRange(0, half_size - 1);
  copy_pattern(left, patterns[left_pattern_index]);
  copy_pattern(right, patterns[left_pattern_index + half_size]);
}
else
if (correlation_type == UNCORRELATED) {
  int left_pattern_index = RandRange(0, patterns.size() - 1);
  copy_pattern(left, patterns[left_pattern_index]);

  int right_pattern_index = RandRange(0, patterns.size() - 1);
  while (right_pattern_index == left_pattern_index) {
    right_pattern_index = RandRange(0, patterns.size() - 1);
  }
  copy_pattern(right, patterns[right_pattern_index]);
}

pattern_t both_patterns;
both_patterns.resize(2*left.size());

for (size_t l = 0; l < left.size(); l++) {
  both_patterns[l] = left[l];
}
for (size_t l = 0; l < right.size(); l++) {
  both_patterns[left.size() + l] = right[l];
}

double sigma0 = standard_deviation(both_patterns);

both_patterns.clear();

if (RandRange(0, 1) == 0) {
  add_height(left, feature_height);
}
else {
  add_height(right, feature_height);
}

sigma = k * sigma0;

add_noise(left, sigma);
add_noise(right, sigma);
```

```
      if (pattern_diff2(left, right)) {
        count_detections++;
      }

    } // end for i

    if (count_detections == 79) {
      wanted_sigma = sigma;
      wanted_detections = count_detections;
      break;
    }
    else {
      int diff = abs(79 - count_detections);
      if (min_diff > diff) {
      //if (count_detections == 79) {
        min_diff = diff;
        wanted_sigma = sigma;
        wanted_detections = count_detections;
        //break;
      }
    }

    k += 0.00025;

  } // end for j

  cout << "Detection Number: " << wanted_detections << "%" << endl;

  return wanted_sigma;
}
*/
////////////////////////////////////////////////////////////////////////////////

double correlation_effect(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type, double s) {

  int min_diff = INT_MAX;
  int wanted_detections = 0;
  double wanted_sigma = 0;

  double k = s;

  int proportion_correct_counter = 0;

  int count_steps = 0;

  pattern_t hand;

  while(true) {
```

```
proportion_correct_counter = 0;

double dt = feature_height;

for (int i = 0; i < 100; i++) {

 pattern_t left;
 pattern_t right;

 if (correlation_type == POSITIVE_CORRELATION) {
  int left_pattern_index = RandRange(0, patterns.size()-1);
  copy_pattern(left, patterns[left_pattern_index]);
  copy_pattern(right, left);
 }
 else
 if (correlation_type == NEGATIVE_CORRELATION) {
  int half_size = patterns.size() / 2;
  int left_pattern_index = RandRange(0, half_size - 1);
  copy_pattern(left, patterns[left_pattern_index]);
  copy_pattern(right, patterns[left_pattern_index + half_size]);
 }
 else
 if (correlation_type == UNCORRELATED) {
  int left_pattern_index = RandRange(0, patterns.size() - 1);
  copy_pattern(left, patterns[left_pattern_index]);

  int right_pattern_index = RandRange(0, patterns.size() - 1);
  while (right_pattern_index == left_pattern_index) {
   right_pattern_index = RandRange(0, patterns.size() - 1);
  }
  copy_pattern(right, patterns[right_pattern_index]);
 }


 bool ramp_added_at_left = false;
 if (RandRange(0,1) == 0) {
  add_height(left, feature_height);
  ramp_added_at_left = true;
 }
 else {
  add_height(right, feature_height);
 }

 add_noise(left, k);
 add_noise(right, k);

 pattern_t hand_with_ramp;
 if (ramp_added_at_left) {
  copy_pattern(hand_with_ramp, left);
 }
```

```
else {
  copy_pattern(hand_with_ramp, right);
}

pattern_t left_4;
copy_pattern(left_4, left, 0, 3);

pattern_t right_4;
copy_pattern(right_4, right, 0, 3);

double r = pearson(left_4, right_4);

if (r >= 0.5) {
  if (pattern_diff(left, right) > 0) {
    copy_pattern(hand, left);
  }
  else {
    copy_pattern(hand, right);
  }
}
else {

  dt = feature_height;

  if (RandRange(0,1) == 0) {
    copy_pattern(hand, right);
  }
  else {
    copy_pattern(hand, left);
  }

  for (size_t j = PATTERN_PART_START; j <= PATTERN_PART_END; j++) {

    if (right[j] > dt) {
      dt = right[j];
      copy_pattern(hand, right);
    }
    if (left[j] > dt) {
      dt = left[j];
      copy_pattern(hand, left);
    }

  }

}

if (patterns_equal(hand_with_ramp, hand)) {
  proportion_correct_counter++;
}
```

```
    } // end for i


//    int diff = abs(79 - proportion_correct_counter);
//    if (min_diff > diff) {
//      min_diff = diff;
//      wanted_sigma = k;
//      wanted_detections = proportion_correct_counter;
//    }

    if (++count_steps == 1000) {
      //wanted_detections = proportion_correct_counter;
      //wanted_sigma = k;
      break;
    }
    else
    if (proportion_correct_counter > 79) {
      k += 0.1;
    }
    else
    if (proportion_correct_counter < 79) {
      k -= 0.1;
    }
    else
    if (proportion_correct_counter == 79) {
      wanted_detections = proportion_correct_counter;
      wanted_sigma = k;
      break;
    }


  } // end while

//  wanted_detections = proportion_correct_counter;
//  wanted_sigma = k;

  cout << "   Detection Number: " << wanted_detections << "%" << endl;

  return wanted_sigma;
} // end correlation_effect

////////////////////////////////////////////////////////////////////////////////

double correlation_effect2(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type, double s) {

  int min_diff = INT_MAX;
  int wanted_detections = 0;
  double wanted_sigma = 0;
```

```
double k = s;

int proportion_correct_counter = 0;

int count_steps = 0;

pattern_t hand;

while(true) {

  proportion_correct_counter = 0;

  double dt = feature_height;

  for (int i = 0; i < 100; i++) {

    pattern_t left;
    pattern_t right;

    if (correlation_type == POSITIVE_CORRELATION) {
      int left_pattern_index = RandRange(0, patterns.size()-1);
      copy_pattern(left, patterns[left_pattern_index]);
      copy_pattern(right, left);
    }
    else
    if (correlation_type == NEGATIVE_CORRELATION) {
      int half_size = patterns.size() / 2;
      int left_pattern_index = RandRange(0, half_size - 1);
      copy_pattern(left, patterns[left_pattern_index]);
      copy_pattern(right, patterns[left_pattern_index + half_size]);
    }
    else
    if (correlation_type == UNCORRELATED) {
      int left_pattern_index = RandRange(0, patterns.size() - 1);
      copy_pattern(left, patterns[left_pattern_index]);

      int right_pattern_index = RandRange(0, patterns.size() - 1);
      while (right_pattern_index == left_pattern_index) {
        right_pattern_index = RandRange(0, patterns.size() - 1);
      }
      copy_pattern(right, patterns[right_pattern_index]);
    }


    bool ramp_added_at_left = false;
    if (RandRange(0,1) == 0) {
      add_height(left, feature_height);
      ramp_added_at_left = true;
    }
    else {
```

```
    add_height(right, feature_height);
  }

  add_noise(left, k);
  add_noise(right, k);

  pattern_t hand_with_ramp;
  if (ramp_added_at_left) {
    copy_pattern(hand_with_ramp, left);
  }
  else {
    copy_pattern(hand_with_ramp, right);
  }

  pattern_t left_4;
  copy_pattern(left_4, left, 0, 3);

  pattern_t right_4;
  copy_pattern(right_4, right, 0, 3);

  if (pattern_diff(left, right) > 0) {
    copy_pattern(hand, left);
  }
  else {
    copy_pattern(hand, right);
  }

  if (patterns_equal(hand_with_ramp, hand)) {
    proportion_correct_counter++;
  }

} // end for i


int diff = abs(79 - proportion_correct_counter);
if (min_diff > diff) {
  min_diff = diff;
  wanted_sigma = k;
  wanted_detections = proportion_correct_counter;
}

if (++count_steps == 1000) {
  //wanted_detections = proportion_correct_counter;
  //wanted_sigma = k;
  break;
}
else
if (proportion_correct_counter > 79) {
  k += 0.1;
}
```

```
    else
    if (proportion_correct_counter < 79) {
      k -= 0.1;
    }
    else
    if (proportion_correct_counter == 79) {
      wanted_detections = proportion_correct_counter;
      wanted_sigma = k;
      break;
    }


  } // end while

//  wanted_detections = proportion_correct_counter;
//  wanted_sigma = k;

  cout << "   Detection Number: " << wanted_detections << "%" << endl;

  return wanted_sigma;
}

////////////////////////////////////////////////////////////////////////////

double correlation_effect3(const vector<pattern_t>& patterns, double feature_height,
correlation_t correlation_type, double s) {

  int min_diff = INT_MAX;
  int wanted_detections = 0;
  double wanted_sigma = 0;

  double k = s;

  int proportion_correct_counter = 0;

  int count_steps = 0;

  pattern_t hand;

  while(true) {

    proportion_correct_counter = 0;

    double dt = feature_height;

    for (int i = 0; i < 100; i++) {

      pattern_t left;
      pattern_t right;
```

```
if (correlation_type == POSITIVE_CORRELATION) {
  int left_pattern_index = RandRange(0, patterns.size()-1);
  copy_pattern(left, patterns[left_pattern_index]);
  copy_pattern(right, left);
}
else
if (correlation_type == NEGATIVE_CORRELATION) {
  int half_size = patterns.size() / 2;
  int left_pattern_index = RandRange(0, half_size - 1);
  copy_pattern(left, patterns[left_pattern_index]);
  copy_pattern(right, patterns[left_pattern_index + half_size]);
}
else
if (correlation_type == UNCORRELATED) {
  int left_pattern_index = RandRange(0, patterns.size() - 1);
  copy_pattern(left, patterns[left_pattern_index]);

  int right_pattern_index = RandRange(0, patterns.size() - 1);
  while (right_pattern_index == left_pattern_index) {
    right_pattern_index = RandRange(0, patterns.size() - 1);
  }
  copy_pattern(right, patterns[right_pattern_index]);
}


bool ramp_added_at_left = false;
if (RandRange(0,1) == 0) {
  add_height(left, feature_height);
  ramp_added_at_left = true;
}
else {
  add_height(right, feature_height);
}

add_noise(left, k);
add_noise(right, k);

pattern_t hand_with_ramp;
if (ramp_added_at_left) {
  copy_pattern(hand_with_ramp, left);
}
else {
  copy_pattern(hand_with_ramp, right);
}

pattern_t left_4;
copy_pattern(left_4, left, 0, 3);

pattern_t right_4;
copy_pattern(right_4, right, 0, 3);
```

```
      dt = feature_height;

      if (RandRange(0,1) == 0) {
        copy_pattern(hand, right);
      }
      else {
        copy_pattern(hand, left);
      }

      for (size_t j = PATTERN_PART_START; j <= PATTERN_PART_END; j++) {

        if (right[j] > dt) {
          dt = right[j];
          copy_pattern(hand, right);
        }
        if (left[j] > dt) {
          dt = left[j];
          copy_pattern(hand, left);
        }

      }

      if (patterns_equal(hand_with_ramp, hand)) {
        proportion_correct_counter++;
      }

    } // end for i


//    int diff = abs(79 - proportion_correct_counter);
//    if (min_diff > diff) {
//      min_diff = diff;
//      wanted_sigma = k;
//      wanted_detections = proportion_correct_counter;
//    }

    if (++count_steps == 1000) {
      //wanted_detections = proportion_correct_counter;
      //wanted_sigma = k;
      break;
    }
    else
    if (proportion_correct_counter > 79) {
      k += 0.1;
    }
    else
    if (proportion_correct_counter < 79) {
      k -= 0.1;
```

```
    }
    else
    if (proportion_correct_counter == 79) {
      wanted_detections = proportion_correct_counter;
      wanted_sigma = k;
      break;
    }

  } // end while

//  wanted_detections = proportion_correct_counter;
//  wanted_sigma = k;

  cout << "   Detection Number: " << wanted_detections << "%" << endl;

  return wanted_sigma;
} // end correlation effect3

////////////////////////////////////////////////////////////////////////////////

bool load_pattern(pattern_t& pattern, const char filename[]) {

  ifstream infile;

  if (!pattern.empty()) {
    pattern.clear();
  }

  infile.open(filename);
  if (infile.fail()) {
    return false;
  }
  while(!infile.eof()) {
    double number;
    infile >> number;
    pattern.push_back(number);
  }
  infile.close();

  return true;
}

////////////////////////////////////////////////////////////////////////////////

void print_pattern(const pattern_t& pattern) {
  for (size_t i=0; i<pattern.size(); i++) {
    cout << pattern[i] << " ";
  }
  cout <<  endl;
}
```

5. Sample Output

```
-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 7.4
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 8.71
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 14.15
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 7.06
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------
```

```
Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 17.52
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 5.71
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 5.16
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 5.41
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 5.55
```

```
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern9.txt
Feature Height = 5.63
Sigma0 = 12.955

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern10.txt
Feature Height = 6.13
Sigma0 = 14.5227

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern10.txt
Feature Height = 7.4
Sigma0 = 14.5227

When we add noise with Sigma = 0 (k = 0), there are 0
Detections

No detections found without noise

-----------------------------------------------------

Left Pattern filename:  ../allpatts-expt1/pattern1.txt
Right Pattern filename: ../allpatts-expt1/pattern10.txt
Feature Height = 8.71
Sigma0 = 14.5227

When we add noise with Sigma = 0 (k = 0), there are 0
Detections
```

```
No detections found without noise

-------------------------------------------------------
```

Output Cut Here