DIPOLE LATTICE

ATUL SINGH ARORA

Upscaling a nano-structure

Dr. Ravi Mehrotra
Indian Institute of Science Education and Research, Mohali

May-July, 2013

Atul Singh Arora: *Dipole Lattice,* Upscaling a nano-structure,

Every honest researcher I know admits he's just a professional amateur. He's doing whatever he's doing for the first time. That makes him an amateur. He has sense enough to know that he's going to have a lot of trouble, so that makes him a professional.

— Charles F. Kettering (1876-1958) (Holder of 186 patents)

ACKNOWLEDGEMENTS

I thank Dr. Ravi Mehrotra for, well conceiving the experiment and guiding me through the process of its realization.

CONTENTS

1 This section can be skipped, without any loss of continuity.

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

1 PROLOGUE

Atoms and molecules are far too small to be observable as individual entities, with our eyes alone. Scientists have come a long way at understanding *their* world. It has been attempted to recreate a specific micro-structure, at a scale where we can directly observe it.

The configuration we've studied here, is that of a Magnetic Dipole Lattice, viz. Magnetic Dipoles that can only rotate about their axis, placed on a grid. Their physics by itself is rather interesting and can be simulated to observe the dynamics. The experiment is expected to show the same dynamics, that of the microscopic world, only directly observable.

1.1 PRIOR ART

TODO: Complete this part after understanding the physics and simulations on the system.

1.2 EXPERIMENTAL SETUP

The upscale version consists of Physical Magnetic Dipoles, that rest on near zero friction spots on a grid. A camera sits on top, with all the dipoles in its field of view. The Lattice Analyser takes the input from the camera and simulates the given temperature through a hardware unit and the coils attached to each dipole. It is that simple.

For implementation details, you may read the following sections.

1.2.1 *The Dipole*

According to the current design (as of May 19, 2013), the Magnetic Dipole is built off of two small cylindrical rare earth magnets, attached to a needle, with their flat face's surface normal perpendicular to the axis of the needle. The needle rests in an assembly with a glass slide at the bottom. This keeps it upright and nearly free of friction. Each dipole further has a circular disc on top, with its centre passing through that of the dipole. The disc has a pattern printed, designed to find its angular position using a camera. Further, the dipole assembly also has two coils along an axis perpendicular to the needle.

1.2.2 *Lattice Analyser*

This is the application that

1. records the dynamics of the system
2. calculates the required field strength of each electromagnet

using a webcam and computer vision techniques. The results of the latter part depend on the temperature that is to be simulated; temperature is not maintained by providing heat, but instead by providing a certain distribution of speeds to the dipoles.

1.2.3 *Temperature*

This is the hardware unit, (will be built around an ATmega 16) that provides the coils with the current as calculated by the Lattice Analyser (using a USB interface).

2 WATCH IT GROW

2.1 SENTIMENTAL INTRODUCTION[1]

Science often seems like a blackbox that relates observables. Even more often, it is rather convenient to lose touch of observables altogether, and wander in the blackbox. Performing an experiment, gets one closer to nature, to the roots of the subject.

2.2 THE JOURNEY

2.2.1 *Look it has begun*

This experiment wasn't started from scratch. My guide, Dr. Ravi Mehrotra, had already worked with a team and created the Dipoles as described earlier. The team had also worked on the image detection algorithms, but their work wasn't usable.

There were three tasks at hand, of which one had been significantly simplified by the prior work.

1. The Dipole

   This had one apparent problem; the dipoles had to be made virtually frictionless (which is not to say they had excessive friction, infact they would oscillate atleast about 8 times before stopping aligned with earth's magnetic field)

2. The Image Analysis

   This part I had to start from the beginning with two basic objectives, as stated earlier; measuring the angle of the dipoles and evaluating the current to be pumped based on the temperature selected.
   What was known soon, was that C++ will be used for programming and linux would be the operating system, to facilitate USB interface with the AVR (next step)

3. The Current Control Hardware

   This is simply for providing a current pulse proportional to the intensity calculated by the lattice analyser. Some schematics for this were available, but were found to be inaccurate and incomplete.

1 this section can be skipped, without any loss of continuity.

2.2.2 *Time Line*

Listed below is the event log, which has the progress as and when it was made.

Time Line

```
* May 31, Friday: Tried all sorts of methods for air suspension
    which failed to work satisfactorily. Despite suspension, we
    couldn't achieve a state that had low enough torque (either
    perturbation or friction, atleast one was visible)
* May 30, Thursday: The vacuum cleaner setup had to be used. The
    box was drilled accordingly and the test failed partially
    anyway. Which is to say that if it is light enough, the
    suspension does take place, however the vertical oscillations
     can not be removed.
* May 29, Wednesday: The pump was built but it failed the test.
    The air pressure generated was negligible. i7 was configured
    alongside and the Lattice Analyser was built on it (had to
    make the multi threading optional using macros) using OpenCV
    with OpenTBB. Tests were run and it was found to be fast
    enough for an 8 x 8 dipole matrix.
* May 28, Tuesday: Multi-threading retried, wasn't quite
    functional last time. Looked up various techniques for multi-
    threading and froze an algorithm. The pump couldn't be built
    for parts weren't available.
* May 27, Monday: Multithreading attempted and succeeded,
    although not a very good release. Making progress in
    installing IPP.

* May 21 - 26, Tuesday to Sunday: Not well; Succeeded in
    compiling the code in windows. It's slower. Looked up
    multithreading using C++ 11

* May 20, Monday: Time Lag measured and found to be roughly 3 to
    4 frames behind. Not quite acceptable. Attempting to install
    OpenCV with IPP

** May 18 and 19, Saturday and Sunday: Completing the
    documentation for the same. Thought of a way of testing the
    time lag.

* May 17, Friday: The algorithm was successfully completed to
    measure 360 degrees. PLUS, completed the frame recording,
    identification of each dipole as unique and dumping the data
    out in file AND its testing with uniform motion which it
    passed with flying colours (which is to say in the visible
    range!, because proper standard deviation tests haven't quite
     been done yet) The vision part of the anlyser is almost done
    .
```

```
* May 16, Thursday: Working on dipole detection. The algorithm
    has started to work partially. It still does a mod 180
    detection.

* May 15, Wednesday: The magnetic lifting worked, but friction
    reduction failed. Rather interestingly the dipole would align
     to the suspension magnet's field. Plus, today the spot
    recognition algorithm was finalized and it seemd to be
    perfect.

* May 14, Tuesday: Trying to get the webcam to work, eventually
    acceded to installing everything on a desktop machine. Worked
     on reducing the friction further

* May 13, Monday: Completed the proof of concept version of the
    latticeAnlyser. Tomorrow we plan to print the coloured ovals
    and test

** May 11 and 12, Saturday and Sunday: Read the opencv tutorials
    when the algorithms started appearing and fitting the bill!

* May 10, Friday: Continued with the setup, finetuning,
    installing other applications, making a documentation
    alongside for better support next time, added a shared folder
     between windows and linux

* May 9, Thursday: Managed to get a few things up and running,
    still setting up ubuntu to run with hardware acceleration,
    failed at trying to get the webcam to work, installed the
    build tools, opencv etc.

* May 8, Wednesday: Met with Dr. X (forgot the name of the person
     at NPL I'm working with) and concluded OpenCV and linux are
    what I'll use. Initiated the downloading of required
    applications, including virtual box and an ubuntu image
```

2.2.3 *Construction of the Dipole*

To remove the friction, there were various ideas, including use of a super conductor. However, eventually three methods were considered and experimentally tested.

1. Ferro-Fluid:

   As it turns out, there are substances that have a ferro magnetic properties but in the liquid form. Consequently, a strong enough magnet would glide if coated with this substance.

   Experimentally, it was found that the friction was higher than the 'needle on glass' setup.

2. Magnetic Levitation:

A magnet can easily suspend another magnet, granted it doesn't flip. This idea was used and a magnetic cylinder was placed coaxial to the needle, using a cylindrical eraser and glue. Beneath the glass slide, an identical magnet was placed with the face that repels upwards.

Experimentally, again it was found that the motion was more damped than the 'needle on glass' setup. The reason for this case was obvious after a little analysis and closer observation. The dipole would align to the field of the magnet, viz. the magnetic field was interfering with the dipole.

3. Air Levitation:

   To test this, the very first requirement was a source of stream of air. For this, we started small. We arranged for a small USB fan from a colleague. The next task was to channel the flow of air. This was accomplished by attaching the front part of a Pepsi Bottle such that the larger diameter was closer to the fan and the mouth of the bottle had the chord stuck to it (could still be moved if required), as diagrammatically given in Figure 1. The final setup has been given in Figure 4.

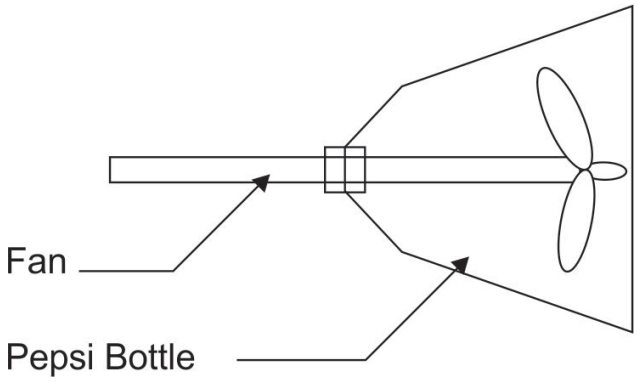

Figure 1: Fan Setup

This failed miserably for the air pressure would fall the moment the assembly covered the fan. Introducing slits to allow air to flow in created no appreciable difference. This idea had to be dropped in favour of the vacuum cleaner setup as shown in **??**

The vacuum cleaner was used as a blower and connected to a box using a pipe. On one of the faces of the box, four holes were drilled (which were later enlarged). These were covered to increase the pressure when required. On the open hole, one dipole was placed (which had to be re built with a minor modification, refer to **??**) with a disc at the bottom. This is when an apparently bizarre observation was made. At a given pressure, it was found that the dipole could remain suspended in

Figure 2: Final Fan Setup

Figure 3: Vacuum Cleaner Setup

air beyond a certain height (and obviously there was an upper bound for the same). However, for heights lower than that, the dipole would fall. The explanation which seemed to resolve this was that the air could spread while rising sufficiently only beyond that height to apply pressure at a large enough surface area, thus create enough force. Albeit the experiment was not performed under precisely the same conditions, when it was repeated with a larger disc, the same problem was encountered, suggesting that there may be more to the explanation that deduced so far. Other geometries at the base (other than the disc) were also tried, such as a cone and a thermecol sphere, neither of which worked at low pressures which were enough to suspend the disc based setups. Further, at high pressures, disturbances in the form of torque in the dipole's axis of rotation begin to appear, which are fatal for the experiment.

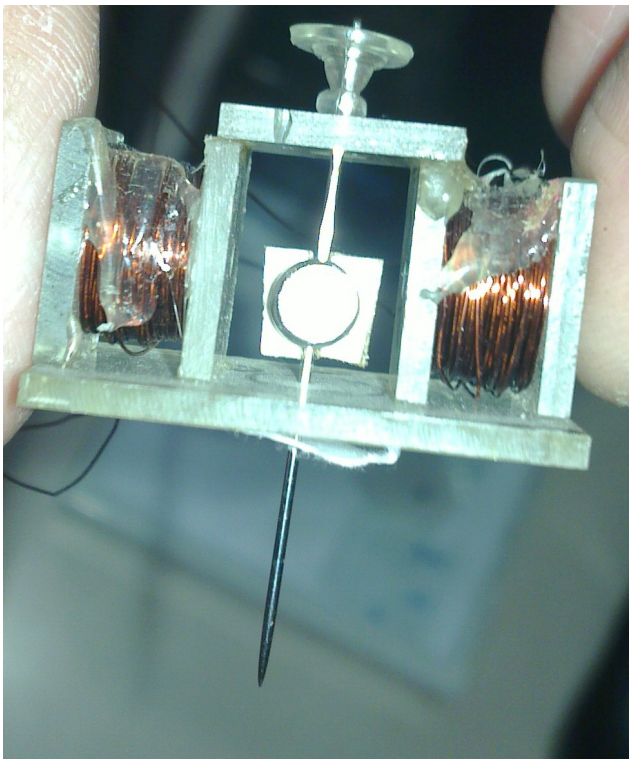

Figure 4: The Modified Dipole

Other problems with air-suspension included damping in the vertical direction. Once the dipole reaches the vertical equilibrium point, it overshoots just as an oscillator. Since the friction at the plates holding the needle vertical is negligible, the oscillations don't get damped quick enough. This was experimentally observed also. The energy ratio between the rotational part and the translation part, in the earth's field, was calculated and found to be approximately one for the given setup.

The most stable we could achieve with this setup was using a cylinderical projection from which the high pressure air escapes and a disc of comparable size attached to the dipole. This did get suspended satisfactorily, unlike the other methods where the suspension could not be maintained at the desired height, however the needle became rather wobbly and unstable resulting in increased friction.

Online research indicated that air-bearings do function well enough and so do the boards of air hockey. These will be explored in the coming weeks to improve upon the methods to achieve the desired results.

2.2.4 *Construction of the Lattice Analyser*

The lattice analyser has come a long way. Image detection trials were initiated with Figure 5.

Figure 5: Sample Image

The idea was that once the ellipses have been detected, and they are different in colour, one can evaluate from their centroids, the position and the angle of the dipole. It must be stated that earlier it was attempted to use the greyscale image as was provided. However soon the shadow interference led to using coloured patterns instead. These patterns were not printed but displayed on a screen and the camera aimed appropriately.

So first, the algorithm for detection of relevant part of the image had to be frozen. There were two candidates for this

1. Hough Transform Method

   Either one could use the already available in OpenCV, line detection or circle detection, both would've required changing the pattern on the dipole

   Or one could use an ellipse modification for the same, which would require programming the algorithm.

2. Contour Detection and Ellipse Fitting

   This method detects contours in a given image, and the OpenCV example also shows ellipse fitting for the same. This seemed promising too, but it seemed more expensive (computationally) than looking for predetermined shapes.

This work had been done within the first few days.

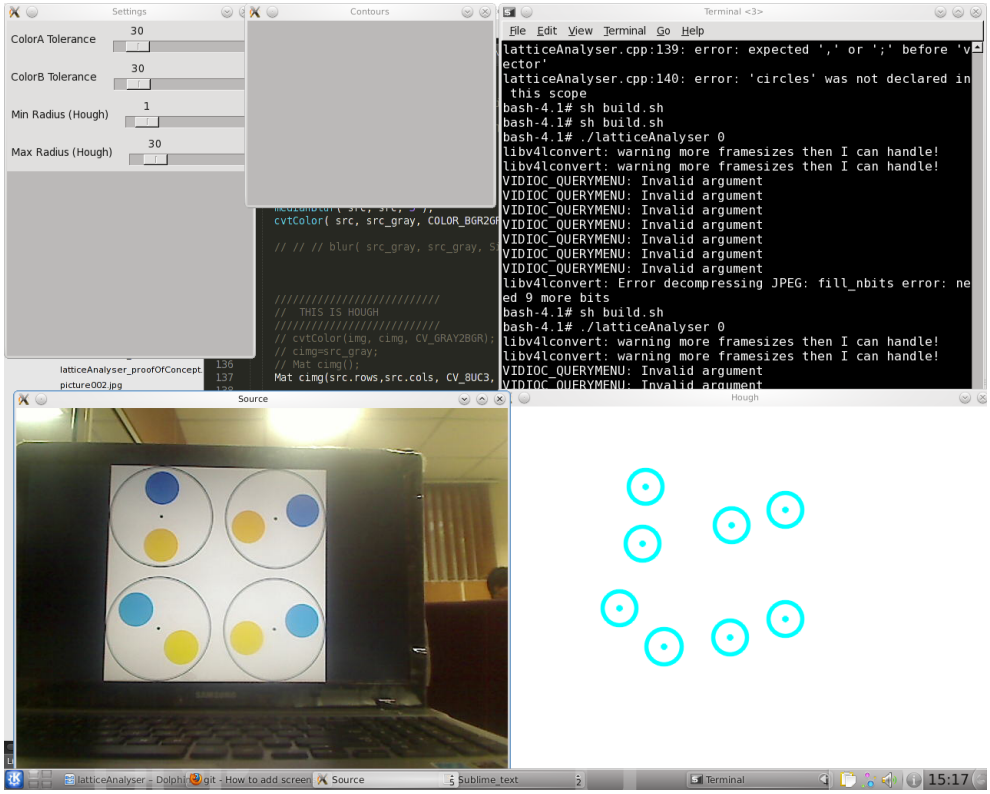

Figure 6: Hough Transform

Next, a colour filter was to setup to improve the accuracy. When the algorithms were implemented, it was found that the Hough Transform method often misses detection of circles, refer to Figure 6 (this is ofcourse after attaching a video stream instead of images to the code) as compared to contour detection Figure 7.

After the detection, according the plan, two colours were to be used for the ellipses. However, running the hough transform twice would've dropped the detection speed to half, which wasn't worth it. It was then decided that the shapes should be made different instead of relying on two colours for the same information. After looking at various combinations, Figure 8 was finalized, with an ellipse at the centre, and a circle along the minor axis for breaking the symmetry. This method did infact work as shown in Figure 9.

The next challenge was to realize that a dipole detection can be missed and therefore mess up the counting, if that is the only way of uniquely identifying them. Unique identification is obviously required, as the external hardware must fire the coils of the right dipole. Thus a reference frame was used to uniquely identify the dipoles initially. This is expected to happen when they are stationary to get a good reading. In each frame, whenever a dipole is detected, it is associated with the dipole in the reference frame, by matching its location.

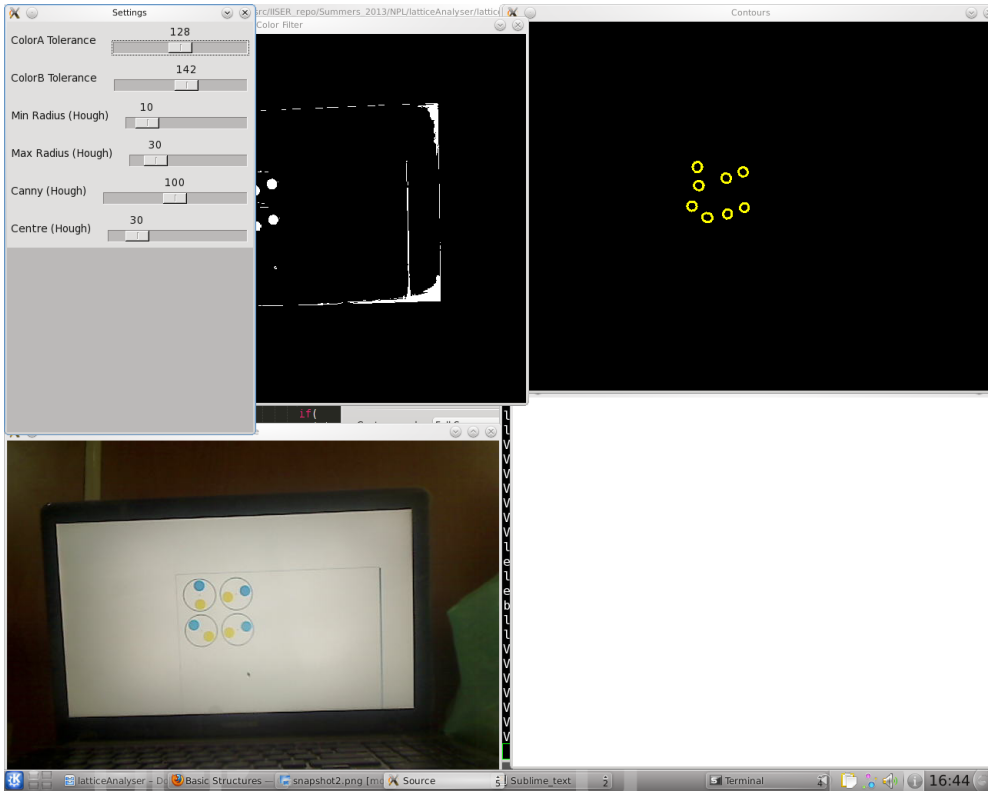

Figure 7: Contour Detection

If a dipole is not detected in a given frame, the software knows it was unable to record it and doesn't mess up neither the numbering nor the observations.

After implementation of the last part, an animation sequence was created in Power Point, with the dipoles rotating with a constant speed and the camera was aimed at the screen. A still from the same is given in Figure 11. Figure 10, shows the angular position versus time plot, for the first dipole and yes, it is linear, just as expected. Standard deviation tests are still to be done.

Further tests, relating to it's timing were done and it was found that the processing itself was taking longer than the time taken by

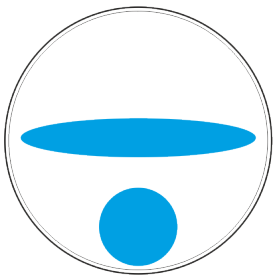

Figure 8: Final Pattern

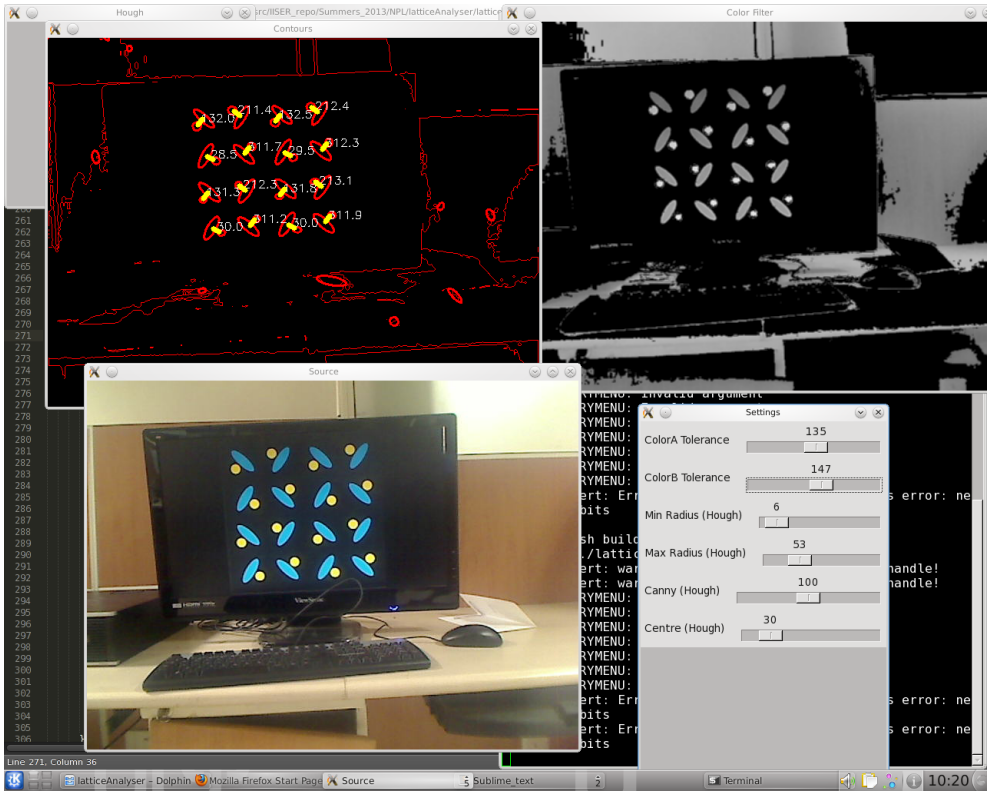

Figure 9: Multi Shape, Single Colour

one frame in a 30 FPS video stream. This could be improved with the following

1. Intel Performance Primitives (IPP)

   These are libraries that provide optimized algorithms for performing some of the basic tasks in OpenCV to speed up the overall computation

2. Multi Threading

   Rendering the frames in a different thread could perhaps increase the render speed or atleast ensure it doesn't affect the rate of processing of the image

3. Camera Initial Delay

   Despite a 30 FPS smooth video, there seemed to be an initial delay which persisted in the video preview of the camera. This can be corrected for by polling the camera quickly instead of processing the image each time.

4. OpenTBB

   To enable multi-threading within OpenCV to speed up the algorithms

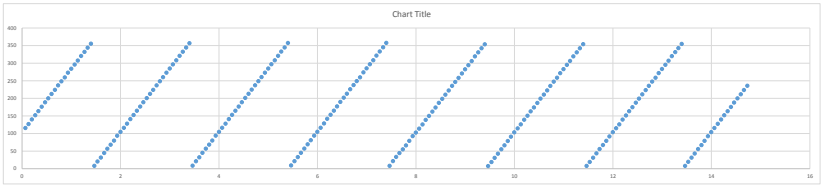

Figure 10: First Observation

5. Hardware

   The results obtained earlier were for a Pentium 4 HT system. A faster multi-core processor could produce potentially, better benchmarks.

Painstakingly, IPP was downloaded, built and integrated with OpenCV and re-built. There were various issues, starting from an incompatible version of IPP, to faulty documentation in OpenCV. The results did not improve however despite this. Perhaps the algorithms used do not rely on the methods optimized by IPP

Multi-threading was implemented using C++ 11. Various synchronization methods were looked up, including mutexes, uniqe locks; eventually atoms were used and tested with Visual Studio Express 2012 on windows (implicitly implying the application was first made to run on windows), as the linux machine had an older kernel and upgrading GCC wasn't recommended.

The camera's initial delay is caused by the initial stream. To rectify this, OpenCV has two methods, one for grabbing a frame, and the other for decoding it. If in a separate loop, the camera is polled regularly for frames, the delay is minimized. The frame can be decoded as soon as processing of the previous frame is over. This, as is suggestive, also requires multi-threading

OpenTBB is a library that is required to enable multi-threading support in OpenCV. This too had to be fiddled with for a while, before being built successfully.

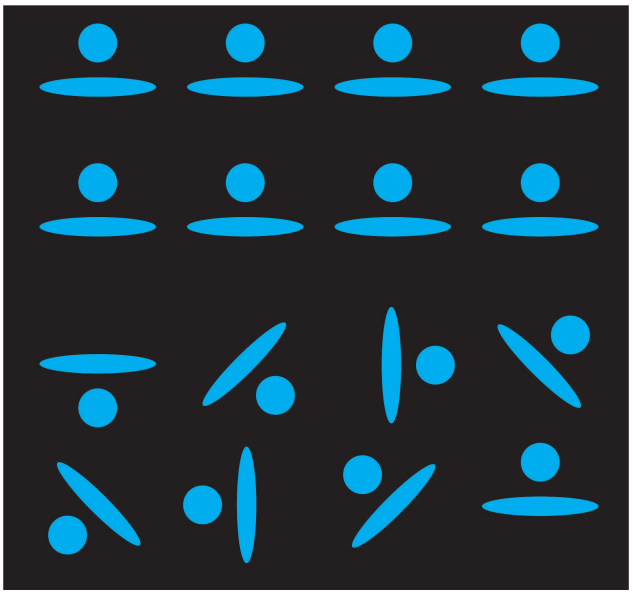

Figure 11: Final Test Pattern

The hardware was changed to an i7 machine which has more then enough cores and computation power.

These modifications were all combined and the code compiled using GCC (except certain parts of multi-threading, due to a compiler issue) and it was found to be able to process in about 25 milliseconds for an 8×8 matrix of dipoles. For a 16×16 matrix, the resolution of the camera (Logitech Pro 9000, 640×480 at 30 FPS) was found to be insufficient. Higher resolutions in the same camera did not have a 30 FPS temporal resolution. The alternative seems to be a camera for the Rasberry Pi which can be used over ethernet.

Further, the CLI of the program was modified to add support for inputting various commands, which would be required for testing the hardware (which is scheduled to be built this week).

Following is the source code of the same, which has been made available online.

latticeAnalyser.cpp

```
/*
  filename: latticeAnalyser.cpp
  description: This is the main application which analyses the
      lattice and
                  1. controls 'temperature'
                  2. records dipole position (thus angular
                      velocity) as a function of time
  baby steps (TM):
    1. Proof of concept stage
        a. Find a suitable algorithm
        b. Make the required modifications
```

```
            i. Add a colour filter
              Implement two colours [done]
              Add two sliders for adjusting tolerance [done, but
                  need to refresh something!]
              Add GUI for selecting the colours [done, but not a
                  gui so to speak]
              save settings [not doing it]
    2. Look at it grow!
        a. Enable screen cropping [done]
        b. Write an algorithm for ellipse to dipole conversion [
            completed]
        c. Save data for each frame using a circular array of
            sorts [done]
        d. Output the data perhaps in a text file  [done]
    3. Testing
        a. Test OpenCV's computation time [done]
                    b. Compiling it on Windows [done and as it
                        turns out, useless]
    4. Optimizing
        a. Multi-threading frame fethcing [done]
        b. Multi-threading display update [done, but failed to
            improve, infact made it worse]
        c. Multi-thread using mutex [done, no real difference but
             not slow either][it is actually very slow]
                    d. Atomic Sync for frame fetching [done]
        f. Atomic Sync for display update [done, and apparently
            faster! Yey]
            [There's something wrong though, since the frame
                jitters like a bad codec.]
            [resolved]
        g. Double Buffer for display [skipped]
    5. Hardware Interface
        a. Modify the CLI to include menus
*/

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <thread>
#include <mutex>
#include <chrono>
#include <string>
// #include <atomic>

// #include <string.h>
// #include <array>



//Configuration
```

```
// #define ATOMIC
// #define MULTI_THREAD_DISPLAY
// #define ATOMIC_DISPLAY
// #define MULTI_THREAD_CAMERA_UPDATE

const float version=0.6;


#ifdef ATOMIC_DISPLAY
  #ifndef MULTI_THREAD_DISPLAY
    #define MULTI_THREAD_DISPLAY
  #endif
#endif

#if (defined(ATOMIC) || defined(ATOMIC_DISPLAY) || defined(
    MULTI_THREAD_DISPLAY) || defined(MULTI_THREAD_CAMERA_UPDATE))
  #include<atomic>
#endif

#ifdef ATOMIC
  atomic<bool> threadsEnabled=false;
#else
  bool threadsEnabled=false;
#endif
using namespace cv;
using namespace std;


#ifdef MULTI_THREAD_DISPLAY
  #ifndef ATOMIC_DISPLAY
    mutex drawnow;
  #endif
#endif


#ifdef ATOMIC_DISPLAY
    atomic<bool> updateDisplayRequested=true;
    atomic<bool> updateDisplayCompleted=false;
#endif

mutex processingImage;
mutex grabbingFrame;

//For computation time
double tCstart,tCdelta,tCend; //time for computation
vector <double> computationTime;

vector <Mat>  buf;

Mat grabbedFrame;
#ifdef MULTI_THREAD_CAMERA_UPDATE
  atomic<bool> frameGrabbed=false,frameRequested=false;
```

```
#else
  bool frameGrabbed=false,frameRequested=false;
#endif

Mat srcPreCrop; Mat src; Mat src_gray; Mat srcColorFilter; Mat
    src_process; Mat srcColorA; Mat srcColorB;Mat drawing;

// int lastBuf=1;
//for the cropping
int cropped = 0;
Point origin;
Rect selection;
bool selectRegion;

// Mat cimg;
Mat_<Vec3b> srcTemp = src;
int thresh = 100;
int max_thresh = 255;
int canny=100;
int centre=30;
int minMinorAxis=1, maxMajorAxis=30;
int mode=0;
float theta=3.14159;

//mode
//0 is screen select
//1 is colour select

RNG rng(12345);

/////////////DIPOLE DETECTION

class dipole
{
public:
  float angle,order; //angle is the angle, order gives a rough
      size of the dipole detected
  int x,y,id;  //centre, id tells where its mapped
  int e1,e2; //index number of ellipse
  static int count[2]; //double buffer
  static int current;
};

int dipole::count[2] = {0};
int dipole::current=0;

//Not using a dynamic array, doesn't matter for now
//TODO: Use a dynamic aray
// array<dipole,500> dipoles;
// array<dipole,500> lastDipoles;
#define DmaxDipoleCount 1000
dipole dipoles[2][DmaxDipoleCount];
```

```
// Colour read
// Point origin;



///////////////DIPOLE INFORMATION STORAGE IN RAM
bool dipoleRec=false;

class dipoleSkel
{
public:
    float angle;
    int x,y;
    float instAngularVelocity;
    bool detected;  //stores whether the dipole was detected at
        all
};

//This class is for storing dipole data of a given frame
class dipoleFrame
{
public:
  double time;   //time elapsed since the seed frame
  float order;  //gives the rough size of the dipoles
  vector<dipoleSkel> data;
};

// #define DframeBufferLen  5000
dipoleFrame seedDipole;
//NOTE: You have to fix the numbering problem right here.
vector <dipoleFrame> dipoleData;

#define DframeBufferLen 5000
////THIS IS FOR STORING IN FILE
FILE * pFile;
char fileName[50];


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

// This is a colour filter for improving accuracy
  // 20, 28, 41 [dark]
  // TODO: Allow the user to select the colour
  Scalar colorB=Scalar(245,245,10);
  Scalar colorA=Scalar(10,245,245);
  int colorATol=30;
  int colorBTol=30;
//
  const char* source_window = "Source";
  const char* filter_window = "Color Filter";
  const char* settings_window="Settings";
```

```
//////////TIMING
  long t,tLast;
  double deltaT;
static void onMouse( int event, int x, int y, int, void* )
{
  if(mode==0)
  {
    if( selectRegion )
    {
        selection.x = MIN(x, origin.x);
        selection.y = MIN(y, origin.y);
        selection.width = std::abs(x - origin.x);
        selection.height = std::abs(y - origin.y);

        selection &= Rect(0, 0, src.cols, src.rows);
    }

    switch( event )
    {
    case CV_EVENT_LBUTTONDOWN:
        cropped=0;
        origin = Point(x,y);
        selection = Rect(x,y,0,0);
        selectRegion = true;
        break;
    case CV_EVENT_LBUTTONUP:
        cropped=0;
        selectRegion = false;
        if( selection.width > 0 && selection.height > 0 )
            cropped = -1;
        break;
    }
  }
  else if(mode==1)
  {
    switch( event )
    {
    case CV_EVENT_LBUTTONUP:
        cout<<x<<","<<y<<endl;
        colorA=Scalar(src.at<Vec3b>(x,y)[0],src.at<Vec3b>(x,y)
            [1],src.at<Vec3b>(x,y)[2]);
        cout<<"Color A's been changed to "<<endl<<colorA.val[0]<<
            endl<<colorA.val[1]<<endl<<colorA.val[2]<<endl;
        break;
    case CV_EVENT_RBUTTONUP:
        cout<<x<<","<<y<<endl;
        colorB=Scalar(src.at<Vec3b>(x,y)[0],src.at<Vec3b>(x,y)
            [1],src.at<Vec3b>(x,y)[2]);
        cout<<"Color B's been changed to "<<endl<<colorB.val[0]<<
            endl<<colorB.val[1]<<endl<<colorB.val[2]<<endl;
        break;
```

```
    }
  }
}

#ifdef __GNUC__
        void tGrabFrame(VideoCapture&& capture)
#else
        void tGrabFrame(VideoCapture& capture)
#endif
{
  while(threadsEnabled)
  {

        capture>>grabbedFrame;
        if(frameRequested)
        {
                grabbedFrame.copyTo(srcPreCrop);
                frameGrabbed=true;
        }

        //buf.push_back(frameGrabbed);

    // if(processingImage.try_lock())
    // {
    //   buf.copyTo(srcPreCrop);
    //   processingImage.unlock();
    // }
  }

}

void updateDisplay()
{
  if(!drawing.empty())
    imshow("Contours", drawing );
  // if(!cimg.empty())
    // imshow("Hough", cimg);
  if(!src_gray.empty())
    imshow( filter_window, src_gray);
  if(!srcPreCrop.empty())
    imshow( source_window, srcPreCrop );

}

#ifdef MULTI_THREAD_DISPLAY
  #ifndef ATOMIC_DISPLAY
    void tUpdateDisplay()
    {
      while(threadsEnabled)
      {
```

```
        drawnow.lock(); //this is to ensure it updates only once
            the processing has been done and not repetatively the
             same frame
        drawnow.unlock();

        updateDisplay();

      }
    }
  #else
    void tAtomicDisplay()
    {
      while(threadsEnabled)
      {

          // && updateDisplayCompleted==false)
        if(updateDisplayRequested)
        {
          updateDisplay();
          // this_thread::sleep_for(chrono::milliseconds(10));
          // updateDisplayCompleted=true;
          updateDisplayRequested=false;
        }
                    // else
          // this_thread::sleep_for(chrono::milliseconds (1));
                        // this_thread::yield();

      }
    }
  #endif
#endif
int process(VideoCapture& capture)
{
  /// Create Window
  namedWindow( source_window, WINDOW_AUTOSIZE );
  setMouseCallback( "Source", onMouse, 0 );
  //Show the filtered image too

  namedWindow( filter_window, WINDOW_AUTOSIZE );

  //Show the settings window

  namedWindow(settings_window,WINDOW_AUTOSIZE  | CV_GUI_NORMAL);
  createTrackbar( "ColorA Tolerance", settings_window, &colorATol
      , 256, 0 );
  createTrackbar( "ColorB Tolerance", settings_window, &colorBTol
      , 256, 0 );
  createTrackbar( "Min Radius (Hough)", settings_window, &
      minMinorAxis, 100, 0 );
  createTrackbar( "Max Radius (Hough)", settings_window, &
      maxMajorAxis, 200, 0 );
```

```
  createTrackbar( "Canny (Hough)", settings_window, &canny, 200,
      0 );
  createTrackbar( "Centre (Hough)", settings_window, &centre,
      200, 0 );
  // createTrackbar( "Theta", settings_window, &thetaD, 3.141591,
       0 );

  /// Show in a window
  namedWindow( "Contours", WINDOW_AUTOSIZE );
  namedWindow( "Hough", WINDOW_AUTOSIZE );


  ////Voodoo intializations
  // dipoles[0][0].current=0;
  // dipoles[0][0].count[0]=0;
  // dipoles[0][0].count[1]=0;
  /// Load source image
  // src = imread( argv[1], 1 );

  // std::string arg = argv[1];
///////////////

  cout<<capture.get(CV_CAP_PROP_FRAME_HEIGHT);
  capture.set(CV_CAP_PROP_FRAME_HEIGHT,480);
  capture.set(CV_CAP_PROP_FRAME_WIDTH,640);

  thread t1(tGrabFrame,capture);
  #ifdef MULTI_THREAD_DISPLAY
    #ifndef ATOMIC_DISPLAY
      thread t2(tUpdateDisplay);
    #else
      thread t2(tAtomicDisplay);
    #endif
  #endif



  for(;;)
  {
    // processingImage.lock();
    //if the video feed has over 1 frame

    // if(buf.size()>1)
    // {
    //   buf.erase(buf.begin()); //remove the oldest frame
    //   srcPreCrop=buf[buf.size()-1]; //grab the latest frame
    // }

    // frameGrabbed.copyTo(srcPreCrop);
        frameRequested=true;

    if(frameGrabbed==true && !srcPreCrop.empty())
```

```
            // #ifdef ATOMIC_DISPLAY
            //      // if(updateDisplayCompleted)
            // #endif

{
  frameRequested=false;     //this is so that the frame is
      not processed unless required
      frameGrabbed=false;   //this is so that we know the
          next time a frame is grabbed

  #ifdef MULTI_THREAD_DISPLAY
    // drawnow=true;
    #ifndef ATOMIC_DISPLAY
      drawnow.lock();
    #endif
  #endif

  {  //IMAGE CAPTURE and CROP
    // capture>>srcPreCrop;
    ///////////////COMPUTATION TIME CALCULATION
    tCstart=getTickCount();
    ///////////
    tLast=t;
    // t=getTickCount()/getTickFrequency();   //This is give
        time in seconds
    t=getTickCount();
    deltaT=(t-tLast)/getTickFrequency();

    if(dipoleRec)
    {
      //if this is not the last frame, add a frame
      dipoleData.push_back(seedDipole);
      //This is to avoid overflows
      // if (dipoleData.size()>DframeBufferLen)
        // dipoleData.erase(dipoleData.begin());
      //for the last frame
      dipoleData[dipoleData.size()-1].time=dipoleData[
          dipoleData.size()-2].time+deltaT;
    }

    // long cfInit=dipoleData.size()-1;  //last frame

    // //test for current frame
    // if(dipoleData[cfInit].time!=t)
    // {
    //   //if this is not the last frame, add a frame
    //   dipoleData.push_back(seedDipole);
    //   //This is to avoid overflows
    //   if (dipoleData.size()>DframeBufferLen)
    //     dipoleData.erase(dipoleData.begin());
    //   //for the last frame
    //   cfInit=dipoleData.size()-1;
```

```
    //   dipoleData[cfInit].time=t;
    // }



    if(srcPreCrop.empty())
    {
      cout<<"Didn't get an image";
      break;
    }
    if(!cropped==0)
    {
      src=srcPreCrop(selection);
    }
    else
      src=srcPreCrop;

    // imshow( source_window, srcPreCrop );

  }

  ////////////////////////
  {//COLOR FILTER
    //Input src, output src_gray
    Scalar lowerBound;
    Scalar upperBound;

    lowerBound = colorA-Scalar::all(colorATol);
    upperBound = colorA+Scalar::all(colorATol);
    // Now we want a mask for the these ranges
    inRange(src,lowerBound,upperBound, srcColorA);

    lowerBound = colorB-Scalar::all(colorBTol);
    upperBound = colorB+Scalar::all(colorBTol);
    // We do it for both the colours
    inRange(src,lowerBound,upperBound, srcColorB);

    // Now we create a combined filter for them
    addWeighted(srcColorA, 1, srcColorB, 1, 0, srcColorFilter
        );


    /// Convert image to gray
    cvtColor( src, src_process, COLOR_BGR2GRAY );

    /// Now keep only the required areas in the image
    // // // multiply(src_process,srcColorFilter,src_gray,1);
    src_gray=srcColorFilter.mul(src_process/255);
    // // // src_gray=srcColorFilter;

    // NOw blur it
    blur( src_gray, src_gray, Size(3,3) );
```

```
  // imshow( filter_window, src_gray);
}

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

// BLANK PROCESSING
// medianBlur( src, src, 5 );
// cvtColor( src, src_gray, COLOR_BGR2GRAY );

// // // blur( src_gray, src_gray, Size(3,3) );

/////////////////////////////
// This is contour Detection
/////////////////////////////////
Mat threshold_output;
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

/// Detect edges using Threshold
threshold( src_gray, threshold_output, thresh, 255,
    THRESH_BINARY );
/// Find contours
findContours( threshold_output, contours, hierarchy,
    RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0) );

/// Find the rotated rectangles and ellipses for each
    contour
vector<RotatedRect> minRect( contours.size() );
vector<RotatedRect> minEllipse( contours.size() );

for( size_t i = 0; i < contours.size(); i++ )
   { minRect[i] = minAreaRect( Mat(contours[i]) );
     if( contours[i].size() > 5 )
       { minEllipse[i] = fitEllipse( Mat(contours[i]) ); }
   }

for( size_t i = 0; i< minEllipse.size(); i++ )
   {
    //You can add aditional conditions to eliminate
        detected ellipses
    if(!(
      (minEllipse[i].size.height>minMinorAxis && minEllipse
          [i].size.width>minMinorAxis)
      &&
      (minEllipse[i].size.height<maxMajorAxis && minEllipse
          [i].size.width<maxMajorAxis)
      ))
    {
      // minEllipse[i]=RotatedRect(Point2f(0,0),Point2f
          (0,0),0);
      minEllipse.erase(minEllipse.begin()+i--);
```

```
        }
       }



    ////////////////////////
    //  Dipole Detection Algorithm
    ////////////////////////
    vector<bool> detected (minEllipse.size(),false);

    int k = !(dipoles[0][0].current);
    dipoles[0][0].current=k;
    dipoles[0][0].count[k]=0;

    // dipolesA[0].lastcount=0;
    for (int i=0; i<minEllipse.size();i++)
    {
      if(detected[i]==false)
      {
        for (int j=0; j<minEllipse.size();j++)
        {
          if((i!=j) && detected[j]==false)  //This is so that
              you don't test with yourself and with others that
               got paired
          {
              // if(abs(minEllipse[i].angle-minEllipse[j].angle
                  )< 15)  //if the orientation matches (less
                  than 5 degrees), then
              {
                //Find the distance between minEllipses

                Point differenceVector=Point(minEllipse[i].
                    center.x - minEllipse[j].center.x,
                    minEllipse[i].center.y - minEllipse[j].
                    center.y);  //find the difference vector
                float distanceSquared=differenceVector.x*
                    differenceVector.x + differenceVector.y*
                    differenceVector.y; //find the distance
                    squared

                //Find the major axis length
                float majorAxis=MAX( MAX(minEllipse[i].size.
                    width, minEllipse[i].size.height) , MAX(
                    minEllipse[j].size.width, minEllipse[j].
                    size.height)); //find the max dimension

                //The ratio is the ratio between the distance
                    between the ellipse and the small circle
                    and the length of the major axis
                float errorPlusOne = distanceSquared /
                    ((11.348/30)*(11.348/30)*majorAxis*
```

```
        majorAxis) ; //now to compare, just divide
        and see if it's close enough to one

    if (errorPlusOne>0.5 && errorPlusOne<2)  //if
        the error is small enough, then its a match
    {
        //This is to ensure these don't get paired
        detected[i]=true;
        detected[j]=true;

        //this is collection of the final result
        int c=dipoles[k][0].count[k]++; //dont get
            confused, count is static, so even
            dipoles[0][0] would've worked, ro for
            that matter, any valid index
        //Note the ++ is after because the count is
             always one greater than the index of
            the last element!


        // dipoles[k][c].angle=(minEllipse[i].angle
             + minEllipse[j].angle)/2.0;
        // dipoles[k][c].angle=(minEllipse[i].angle
            );

        // We're using two shapes, one ellipse and
            one circle.
        RotatedRect largerEllipse =  ( MAX(
            minEllipse[i].size.width, minEllipse[i
            ].size.height) > MAX(minEllipse[j].size
            .width, minEllipse[j].size.height)  )?
            minEllipse[i]:minEllipse[j];
        RotatedRect smallerEllipse =  ( MAX(
            minEllipse[i].size.width, minEllipse[i
            ].size.height) <= MAX(minEllipse[j].
            size.width, minEllipse[j].size.height)
             )?minEllipse[i]:minEllipse[j];
        dipoles[k][c].angle=(largerEllipse.angle);

        dipoles[k][c].order=MAX(largerEllipse.size.
            height, largerEllipse.size.width);

        dipoles[k][c].x=largerEllipse.center.x; //(
            minEllipse[i].center.x + minEllipse[j].
            center.x)/2.0;
        dipoles[k][c].y=largerEllipse.center.y; //(
            minEllipse[i].center.y + minEllipse[j].
            center.y)/2.0;

        //Now we use the circle to remove the mod
            180 problem and get the complete 360
            degree position
```

```
if((smallerEllipse.center.y -largerEllipse.
    center.y) < 0)
  dipoles[k][c].angle+=180;

dipoles[k][c].e1=i; //don't know why this
    is required
dipoles[k][c].e2=j;




//////////////THIS IS FOR RECORDING/SAVING
    THE DIPOLE MOVEMENT/////////////
if (dipoleRec==true)
{

  long cf=dipoleData.size()-1;  //last
      frame


  for(int q=0;q<seedDipole.data.size();q++)
  {
    //This is to test which dipole belongs
        where in accordance with the
        seedDipole frame
    // if(MAX(abs(seedDipole.data[q].x -
        dipoles[k][c].x), abs(seedDipole.
        data[q].y - dipoles[k][c].y)) < (
        seedDipole.order/2.0) )
    //Or you could use the last fraame for
        this
    if(
      (MAX(abs(dipoleData[cf-1].data[q].x -
           dipoles[k][c].x), abs(dipoleData
          [cf-1].data[q].y - dipoles[k][c].
          y)) < (dipoleData[cf-1].order
          /4.0) )
      &&
      (dipoleData[cf].data[q].detected==
          false)
      )
    {
      dipoles[k][c].id=q;
      // dipoleData.data[q] = dipoles[k][c]
      //TODO: Make a function for
          converting
      dipoleData[cf].data[q].x=dipoles[k][c
          ].x; //Copy the relavent data
          from the dipole data collected
          into the temp dipole
      dipoleData[cf].data[q].y=dipoles[k][c
          ].y;
```

```
                        dipoleData[cf].data[q].angle=dipoles[
                            k][c].angle;
                        dipoleData[cf].data[q].
                            instAngularVelocity=0;
                        dipoleData[cf].data[q].detected=true;
                               //This is true only when the
                            dipole's

                        dipoleData[cf].order=dipoles[k][c].
                            order; //This is bad programming
                            ..i should average, but doens't
                            matter
                        //Now that it has matched, terminate
                            the loop
                        q=seedDipole.data.size();
                      }
                    }


                }









              }
              // magnitude(differenceVector.x,
                  differenceVector.y,distance);

              // point positionVector ((minEllipse[i].x +
                  minEllipse[j].x)/2.0,(minEllipse[i].y +
                  minEllipse[j].y)/2.0);

            }
          }
        }
      }
    }

//////////////DRAWING THE CONTOUR AND DIPOLE
/// Draw contours + rotated rects + ellipses
drawing = Mat::zeros( threshold_output.size(), CV_8UC3 );
for( size_t i = 0; i< contours.size(); i++ )
   {
     // Scalar color = Scalar( rng.uniform(0, 255), rng.
         uniform(0,255), rng.uniform(0,255) );
    Scalar color = Scalar(0,0,255 );
     // contour
```

```
            drawContours( drawing, contours, (int)i, color, 1, 8,
                vector<Vec4i>(), 0, Point() );

            // ellipse
            if(i<minEllipse.size())
                                ellipse( drawing, minEllipse[i],
                                    color, 2, 8 );

            // rotated rectangle
            // Point2f rect_points[4]; minRect[i].points(
                rect_points );
            // for( int j = 0; j < 4; j++ )
            //    line( drawing, rect_points[j], rect_points[(j+1)
                %4], color, 1, 8 );
           // }


           // int xx=dipoles[k][i].x;
           // int yy=dipoles[k][i].y;
           // int theta=dipoles[k][i].angle;

           // line(drawing, Point2f(xx,yy),Point2f(xx + 5*cos(
               theta), yy + 5*sin(theta)), Scalar(0,0,255),1,8);









         }

        for( int i=0;i<dipoles[0][0].count[k];i++)
        {

          int xx=dipoles[k][i].x;
          int yy=dipoles[k][i].y;
          float theta = (3.1415926535/180) * dipoles[k][i].angle;

          line(drawing, Point2f(xx - 5*cos(theta), yy - 5*sin(theta
              )),Point2f(xx + 5*cos(theta), yy + 5*sin(theta)),
              Scalar(0,255,255),5,8);



          // Use "y" to show that the baseLine is about
          char text[30];
          // dipoles[0][0].count[0]=1;
          // sprintf(text,"%f",dipoles[0][dipoles[0][0].count[k
              ]-1].angle);

          int fontFace = FONT_HERSHEY_SCRIPT_SIMPLEX;
          double fontScale = 0.5;
```

```
    int thickness = 1;

    int baseline=0;
    Size textSize = getTextSize(text, fontFace,
                                fontScale, thickness, &
                                    baseline);
    baseline += thickness;

    // center the text
    Point textOrg((drawing.cols - textSize.width)/2,
                  (drawing.rows + textSize.height)/2);
      // // draw the box
      // rectangle(drawing, textOrg + Point(0, baseline),
      //           textOrg + Point(textSize.width, -textSize.
          height),
      //           Scalar(0,0,255));
      // // ... and the baseline first
      // line(drawing, textOrg + Point(0, thickness),
      //      textOrg + Point(textSize.width, thickness),
      //      Scalar(0, 0, 255));

      // then put the text itself
      // putText(drawing, text, textOrg, fontFace, fontScale,
           Scalar::all(255), thickness, 8);
    sprintf(text,"%1.1f",dipoles[k][i].angle);
    putText(drawing, text, Point(dipoles[k][i].x,dipoles[k][i
        ].y), fontFace, fontScale, Scalar::all(0), thickness
        *3, 8);
    putText(drawing, text, Point(dipoles[k][i].x,dipoles[k][i
        ].y), fontFace, fontScale, Scalar::all(255),
        thickness, 8);

    sprintf(text,"%d,%d",dipoles[k][i].id,i);
    putText(drawing, text, Point(dipoles[k][i].x,dipoles[k][i
        ].y-10), fontFace, fontScale, Scalar::all(0),
        thickness*3, 8);
    putText(drawing, text, Point(dipoles[k][i].x,dipoles[k][i
        ].y-10), fontFace, fontScale, Scalar(255,255,0),
        thickness, 8);

    //DEBUG ONLY
    if(i==0)
    {
      Mat cimg(src.rows,src.cols+500, CV_8UC3, Scalar(0,0,0))
          ;
      sprintf(text,"%1.1f",dipoles[k][i].angle);
      putText(cimg, text, Point(dipoles[k][i].x-50,dipoles[k
          ][i].y), fontFace, fontScale*12, Scalar::all(255),
          thickness*4, 8);
    }
  }
```

```
    ////////////////////////
    //  THIS IS HOUGH
    ////////////////////////
    // // cvtColor(img, cimg, CV_GRAY2BGR);
    // // cimg=src_gray;
    // // Mat cimg();
    // Mat cimg(src.rows,src.cols, CV_8UC3, Scalar(255,255,255)
        );

    // vector<Vec3f> circles;
    // HoughCircles(src_gray, circles, CV_HOUGH_GRADIENT, 1,
        10,
    //              canny, centre, minMinorAxis, maxMajorAxis
        // change the last two parameters
    //                             // (min_radius & max_radius)
         to detect larger circles
    //              );

    // // src_gray:s Input image (grayscale)
    // // circles: A vector that stores sets of 3 values: x_{c
        }, y_{c}, r for each detected circle.
    // // CV_HOUGH_GRADIENT: Define the detection method.
        Currently this is the only one available in OpenCV
    // // dp = 1: The inverse ratio of resolution
    // // min_dist = src_gray.rows/8: Minimum distance between
        detected centers
    // // param_1 = 200: Upper threshold for the internal Canny
         edge detector
    // // param_2 = 100*: Threshold for center detection.
    // // min_radius = 0: Minimum radio to be detected. If
        unknown, put zero as default.
    // // max_radius = 0: Maximum radius to be detected. If
        unknown, put zero as default

    // for( size_t i = 0; i < circles.size(); i++ )
    // {
    //     Vec3i c = circles[i];
    //     // Scalar color = Scalar( rng.uniform(0, 255), rng.
        uniform(0,255), rng.uniform(0,255) );
    //     Scalar color = Scalar( 255,255,0 );
    //     circle( cimg, Point(c[0], c[1]), c[2], color, 3,
        CV_AA);
    //     circle( cimg, Point(c[0], c[1]), 2, color, 3, CV_AA)
        ;
    // }

    // imshow("Hough", cimg);

    #ifndef MULTI_THREAD_DISPLAY
      updateDisplay();
    #endif
```

```
#ifdef MULTI_THREAD_DISPLAY
  // drawnow=true;
  #ifndef ATOMIC_DISPLAY
    drawnow.unlock();
  #else
    updateDisplayRequested=true;  //Both are required to
        trigger an update
    // updateDisplayCompleted=false; //The request flag is
        used to indicate processing is done, the second
        indicates the same here
    //however, once the frame has been updated, the update
        flag avoids unecessary refreshing of the frames.
  #endif

#endif
//////////////////////
// CLI
//////////////////////
char key = (char) waitKey(5); //delay N millis, usually
    long enough to display and capture input
int kMax; //sorry, bad programming, but relatively
    desparate for results..
switch (key)
{

    case 'c':
      mode=1;
      cout<<"Mouse will capture color now. Right click for
          one, left for the other";
      break;
    case 's':
      mode=0;
      cout<<"Screen crop mode selected. Mouse will capture
          start point at left click and the other point at
          right click";
      break;
    case 'p':
      cout<<"Frame will be used as a seed";
      dipoleRec=true; //Enable dipole recording
      seedDipole.data.clear();  //clear the data
      dipoleSkel tempDipole;  //create a temporary dipole
          skeleton
      k=dipoles[0][0].current;  //find the current buffer
          of dipoles detected (double buffered for possible
           multithreading)
      kMax=dipoles[0][0].count[k]; //find the number of
          dipoles detected in the last scan

      for(int c=0;c<kMax;c++)
      {
```

```
            tempDipole.x=dipoles[k][c].x; //Copy the relavent
                data from the dipole data collected into the
                temp dipole
            tempDipole.y=dipoles[k][c].y;
            tempDipole.angle=dipoles[k][c].angle;
            tempDipole.instAngularVelocity=0;
            tempDipole.detected=false;   //This is to ensure
                the dipole was detected, but for the seed frame
                , it is left false.
            seedDipole.data.push_back(tempDipole);  //Add the
                data in the seedframe's data stream

            seedDipole.order+=dipoles[k][c].order;  //to get
                teh average order
            if(c>0)
            {
              seedDipole.order/=2.0;
            }
          }
          seedDipole.time=0; //Initial time is to be stored as
              zero
          dipoleData.push_back(seedDipole);

          break;
        case 'w':
          cout<<"Writing angle vs time for the first dipole to
              file";
          if(dipoleRec==true)
          {
            sprintf(fileName,"latticeAnalyser_%d",getTickCount
                ());
            pFile = fopen (fileName,"w");

            //Loop through all the frames
            for (vector<dipoleFrame>::iterator dD = dipoleData.
                begin() ; dD != dipoleData.end(); ++dD)
            {
              //Within each frame, loop through all dipoles?
              // for(vector<dipoleSkel>::iterator dS = dD.data.
                  begin() ; dS!=dD.data.end() ; ++dS)
              // {

              // }
              //or just print the first dipole
              if(dD->data[0].detected)
                fprintf (pFile, "%f,%f\n",dD->data[0].angle,dD
                    ->time);
              // fprintf (pFile, "%d,%d\n",dD->data[0].angle,dD
                  ->time);
            }
```

```
            // for (int p=0;p<dipoleData.size();p++)
            // {
            //   fprintf(pFile,"%d,%d\n",dipoleData[p].data.
                size(),dipoleData[p].time);
            // }
            fclose (pFile);
            // fprintf (pFile, "Name %d [%-10.10s]\n",n,name);

          }
          break;
        case 'W':

          pFile=fopen("TestComputation","w");
          for(vector<double>::iterator d=computationTime.begin
              ();d!=computationTime.end();++d)
          {
            fprintf(pFile,"%f\n",*d);
          }
          fclose(pFile);
          break;
        case 'q':
        case 'Q':
        case 27: //escape key
            destroyWindow(source_window);
            destroyWindow(filter_window);
            destroyWindow(settings_window);
            destroyWindow("Contours");
            destroyWindow("Hough");

            threadsEnabled=false;
            t1.join();
            #ifdef MULTI_THREAD_DISPLAY
              t2.join();
            #endif
            // destroyAllWindows();
            // this_thread::sleep_for( chrono::milliseconds
                (5000) );
            // waitKey(1000);
            return 0;
        // case ' ': //Save an image
        //     sprintf(filename, "filename%.3d.jpg", n++);
        //     imwrite(filename, frame);
        //     cout << "Saved " << filename << endl;
        //     break;
        default:
            break;
      }
      tCend=getTickCount();
      tCdelta=tCend-tCstart;
      computationTime.push_back(tCdelta/getTickFrequency());
    }
    // processingImage.unlock();
```

```
  }
  return 0;


}

/**
 * @function main
 */
int main( int ac, char** argv )
{
  cout<<"Loading";
    cout<<endl<<endl
        <<"Lattice Analyser | version "<<version<<endl
        <<"-------------"<<endl
        <<"Created at the National Physical Laboratory, New Delhi
            "<<endl
        <<endl
        <<"Project Repository Folder: github.com/toAtulArora/
            IISER_repo/Summers_2013/NPL"<<endl
        <<endl
        <<"For help type"<<endl
        <<"help"<<endl
        <<"(Like you couldn't guess!)"<<endl<<endl
        <<"\t now what?  ";


  for(;;)
  {
    string a;
    cin>>a;

    if(!a.compare("help"))
    {
      cout<<endl; //for multi line, beauty stuff

      cout<<"Command \t Description"<<endl
      <<"------- \t -------------"<<endl
      <<"<number> \t Initiates analysis of dipoles using the
          corresponding camera"<<endl;

      cout<<endl; //again for multi line console outputs, to
          maintain beuty
    }
    else if(!a.compare("exit") || !a.compare("quit"))
    {
      break;
    }
    else if(atoi(a.c_str())!=0 || !a.compare("o"))
    {
      threadsEnabled=true;
```

```
      VideoCapture capture; //try to open string, this will
          attempt to open it as a video file
      // if (!capture.isOpened()) //if this fails, try to open as
           a video camera, through the use of an integer param

      capture.open(atoi(a.c_str()));
      if (capture.isOpened())
      {
        process(capture);

      }
      else
      {
          cerr << "Failed to open the video device specified" <<
              endl;
          // return 1;
      }
    }
    cout<<endl<<"\t now what? ";
  }



  return 0;
}
```

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede, for LaTeX.
The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*".

The latest version of this document is available online at:

https://github.com/toAtulArora/IISER_repo