Penggunaan *Pattern Matching* Dalam Perangkat Lunak *Video Stabilizing* Sederhana

Ardi Wicaksono - 13512063
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512063@std.stei.itb.ac.id

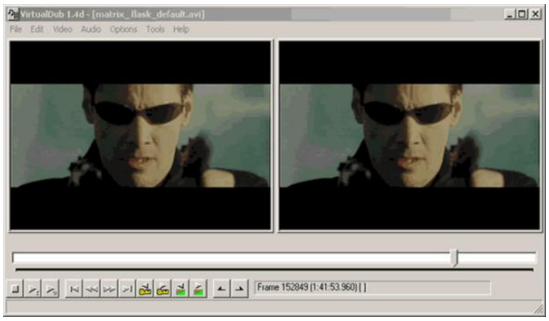
***Abstrak*—Makalah ini menjelaskan tentang keterlibatan algoritma pattern matching pada fitur *Video Stabilizing* yang dimiliki beberapa perangkat lunak *Video Editing*. Video merupakan kumpulan gambar atau *frame* yang terurut dan dijalankan dengan sangat cepat. Pada perekaman video, sering perekamnya tidak memiliki kemampuan atau menggunakan alat bantu yang membuat hasil rekamannya tetap stabil sehingga membuat hasil rekamannya banyak berguncang dan kurang nyaman ditonton. Solusi dari permasalahan tersebut adalah dengan menggunakan perangkat lunak atau fitur *video stabilizing* yang dapat membuat video lebih stabil dengan memanipulasi tiap *frame* dari video tersebut pada tahap paska produksi dari pengerjaan video tersebut. Fungsi dari fitur atau perangkat lunak *video stabilizing* tersebut melibatkan pencarian bagian tengah dari suatu frame dan melacaknya pada satu atau banyak frame berikutnya. Pelacakan tersebut melibatkan algoritma *pattern matching* yang mencari keberadaan sebuah pola dengan menyocokkannya dengan sebuah data yang berukuran lebih besar dari pola yang dicari dan memberikan hasil tentang dimana terjadinya kecocokkan tersebut.**

Kata Kunci— Frame, pattern matching,* Video, *Video editing, Video stabilization.

I. PENDAHULUAN

Perekaman video tidak selalu dipersiapkan dengan sempurna dan hasilnya tidak jarang kurang dari harapan. Salah satu contoh kurangnya persiapan adalah dengan tidak digunakannya alat bantu seperti *tripod* (kecuali memang rencananya hanya merekam dengan kamera dipegang tangan) dalam pengambilan video sehingga pengguna terpaksa hanya mengandalkan kekuatan tangannya saat merekam video dan menahan kameranya. Baik direncanakan ataupun tidak, merekam video dengan kamera yang hanya dipegang tangan cenderung mengakibatkan banyak guncangan yang tidak diharapkan pada video hasilnya ditambah jika video yang direkam mengharuskan perekamnya menahan kamera dalam waktu yang lama dan mengakibatkan kelelahan di tangan atau cara memegang kamera dari perekamnya juga kurang baik. Hal ini dapat mengurangi kualitas dari film atau video yang dihasilkan nantinya dan mengurangi kepuasan penonton atau sampai membuat penonton sakit mata atau pusing.

Solusi dari hasil rekaman video yang banyak guncangannya adalah dengan menggunakan fungsi *video stabilizing* saat *editing* video pada tahap paska produksi dari pengerjaan video tersebut. Berbagai macam perangkat lunak *video editing* baik yang terkenal dan mahal seperti Adobe(R) Premiere Pro atau Sony(R) Vegas maupun yang bersifat freeware seperti VirtualDub sampai YouTube juga telah memiliki fitur ataupun *plug-in* yang membolehkan penggunanya melakukan video stabilizing dan menghasilkan video yang lebih stabil dan enak dilihat. Rata-rata proses *video stabilizing* tersebut melibatkan pelacakan bagian tengah dari *frame* awal pada *frame-frame* berikutnya yang dapat memanfaatkan algoritma pattern matching seperti Brute Force, Knuth-Morris-Pratt dan Boyer-Moore.



Gambar 1.1: Tampilan VirtualDub, salah satu aplikasi yang dapat diberi *plug-in video stabilizing* [9]

II. TEORI DASAR

2.1. Video Digital dan Pixel

Video digital mengacu pada pengambilan, manipulasi dan penyimpanan gambar bergerak yang dapat ditampilkan di layar komputer[1]. Video digital menggantikan video analog dikarenakan fleksibilitasnya untuk dimanipulasi dalam *editing* paska produksi, data yang lebih awet tersimpan dan dapat ditransfer secara bebas dengan cukup aman[1].

Video sendiri sebenarnya merupakan sekumpulan gambar yang dijalankan dengan sangat cepat (>18 gambar per detik). Gambar-gambar ini disebut dengan *frame* dan dijalankan dengan kecepatan yang tergantung dari format video yang bersangkutan, dengan satuan yang mungkin pernah kita dengar dengan *frame per second* atau fps. Gambar-gambar ini terlihat bergerak karena otak kita tidak sempat memproses satu per satu *frame* tersebut

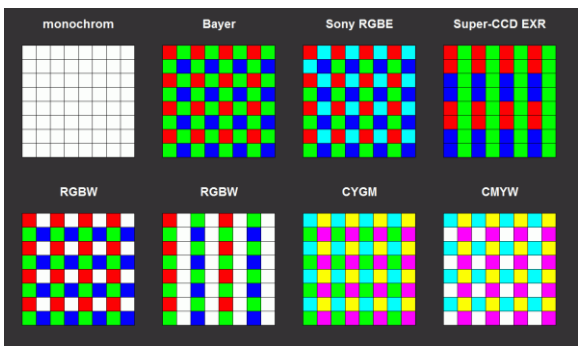
sebagai gambar yang individual[8].

Saat diproses, tiap *frame* dari suatu video digital yang disimpan di suatu memori yang disebut *framebuffer* untuk ditampilkan pada waktunya, di dalam *framebuffer* ini suatu video memiliki representasi data untuk tiap *frame* nya yang nantinya akan ditampilkan pada layar monitor untuk sepersekian detik, tepatnya 1/fps detik.

Gambar 2.1: Video seorang petenis yang digambarkan *frame per frame*[8]

Tiap *frame* tersebut merupakan gambar digital yang tersusun atas pixel-pixel kecil. Pada video kita sering mendengar istilah resolusi, resolusi tersebut adalah jumlah dari seluruh pixel-pixel tersebut yang ditampung pada data digital video dalam tiap *frame* nya. Misalkan resolusi 1280 x 720 berarti pada satu *frame* terdapat 1280 pixel yang berderet secara horizontal dan masing-masing deret pixel tersebut memiliki 720 pixel yang berderet secara vertikal. Sehingga pada video beresolusi 1280 x 720, dalam satu *frame* terdapat sejumlah data pixel dalam matriks[1280][720].

Data pada memori *framebuffer* terdiri dari data elemen warna dari masing-masing pixel pada *frame* yang akan ditampilkan pada layar. Misalnya pixel pada kordinat matriks [10][10] menyimpan data warna ungu muda, maka pada layar komputer, di titik (10,10) akan berwarna ungu muda. Data yang disimpan tiap pixel biasanya berupa intensitas dari 3 atau 4 elemen, yaitu *Red, Green, Blue* (RGB) atau *Cyan, Magenta, Yellow, Black* (CMYK)[4] meskipun masih ada banyak macam sistem pewarnaan digital lainnya.



Gambar 2.1: Contoh warna pixel pada beberapa sistem pewarnaan[7]

Pada sistem warna RGB, nilai intensitas dari masing-masing warna tersebut berkisar sesuai dengan *bits per pixel* (bpp) yang dapat digunakan oleh perangkat yang menggambarkannya, pada komputer masa kini telah digunakan 24 bpp. Pada 24 bpp, data ketiga warna utama ini akan disimpan dalam data biner berukuran 3 byte/24 bit. Karena tiap elemen warnanya memiliki ukuran 1 byte, maka intensitas tiap warnanya dapat berkisar dari 0 sampai 255 dengan 255 berarti warna tersebut berintensitas maksimal di pixel yang bersangkutan. BPP yang digunakan tadi menentukan ukuran gambar pada memori komputer dan bagaimana representasi datanya untuk tiap pixel. Misal pada suatu pixel yang berwarna putih, nilai R adalah 255, G adalah 255 dan B adalah 255, angka 255 pada heksadesimal tertulis 0xFF, maka pada memori pixel tersebut akan menjadi tulisan heksadesimal 0xFFFFFF.

2.2. *Pattern Matching*

Pattern Matching merupakan algoritma pencocokan sebuah pola (*pattern*) dengan sebuah sampel yang lebih besar ukurannya dari pola yang ingin dicocokkan tadi, dengan tujuan mencari ada atau tidaknya pola tersebut pada sampel atau untuk mencari dimana letak pola tersebut pada sampel. *Pattern matching* biasa digunakan untuk pencocokkan teks atau gambar. Pencocokkan gambar pun pada teknisnya hampir sama dengan pencocokkan teks karena yang dicocokkan adalah gambar tersebut dalam representasi data yang berupa matriks kode biner yang dapat dicocokkan.

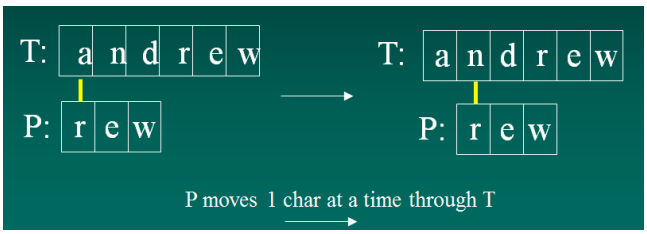
Algoritma *pattern matching* yang umum ada tiga yaitu Brute Force, Knuth-Morris-Pratt (KMP) dan Boyer-Moore. Namun pada kasus yang dibahas dalam makalah ini yang akan digunakan hanya salah satunya saja yaitu algoritma *Pattern Matching* Brute Force yang akan diberi sedikit perubahan untuk mentoleransi perubahan kecil yang terjadi pada pergantian *frame* suatu video. Algoritma ini terpilih karena pola berbentuk matriks dan apabila menggunakan KMP atau Boyer-Moore yang memiliki kemampuan character-jump maka pelompatan akan tidak jelas ke arah mana apabila ketidakcocokan terjadi di tengah matriks.

2.2.1 Algoritma Brute Force

Brute Force adalah algoritma dengan pendekatan yang *straightforward*. Algoritma brute force mencoba menyelesaikan suatu persoalan dengan mencoba seluruh kemungkinan satu per satu tanpa ada teknik khusus. Penyelesaian dengan algoritma ini cenderung sederhana namun lebih lambat. Istilah brute force pun tidak hanya digunakan pada *pattern matching* namun juga pada beberapa persoalan lain yang diselesaikan dengan cara mencoba seluruh kemungkinan.

Pada *pattern matching*, algoritma brute force mencari keberadaan pola P dengan cara mencocokkan pola P dengan sampel T secara satu per satu karakter. Apabila terjadi ketidak cocokkan, secara *default* algoritma ini akan

menggeser pola satu karakter ke kanan dan melakukan pencocokkan lagi, namun karena pada kasus ini pola berbentuk matriks, maka penggeserannya akan dilakukan sedikit modifikasi.



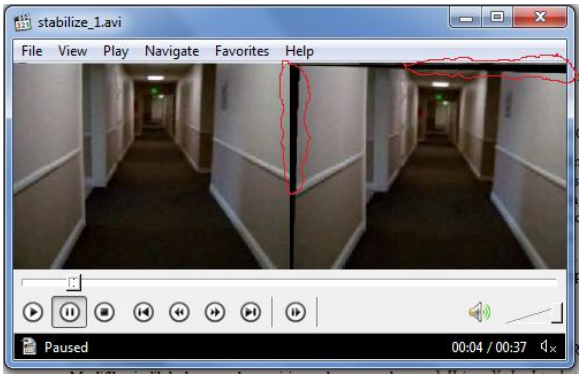
Gambar 2.2: Ilustrasi *pattern matching* dengan Brute Force[6]

Pada *string matching* menggunakan brute force, kompleksitas yang dimilikinya adalah O(mn) dengan best case O(m) dan average case O(m+n). Algoritma brute force cenderung lebih efektif apabila jenis elemen yang dicocokkan banyak. Algoritma brute force juga tidak perlu melakukan pra-proses seperti yang dilakukan algoritma KMP atau Boyer-Moore[3].

III. Video Stabilizing dan Ruang Lingkup

Video Stabilizing adalah proses yang dilakukan untuk menstabilkan video yang terekam. Proses ini sebenarnya tidak selalu dilakukan oleh komputer saat paska produksi sebuah video, namun beberapa kamera juga memiliki fungsi ini dengan lensa atau *body* kamera yang menggunakan motor-motor kecil yang disebut *servo*, namun hal tersebut dilakukan oleh perangkat keras, bukan dengan manipulasi data sehingga secara teknis berbeda dengan *video stabilizing* saat *editing*.

Video stabilizing yang dibahas pada kasus ini disebut juga *Digital image stabilization* atau *stabilization filter*[10] dan melibatkan proses manipulasi data dari *frame-frame* video yang akan diproses. Proses ini dilakukan dengan melacak keberadaan bagian tengah dari sebuah *frame* pada *frame* berikutnya, kemudian saat letaknya sudah ditemukan, keseluruhan *frame* yang belakangan digeser sehingga bagian tengah *frame* awal tadi berada pada bagian tengah dari *frame* yang kedua pula, sehingga tidak ada pergeseran bagian tengah tersebut. Namun, proses tersebut akan menyebabkan adanya bagian *frame* kedua yang tidak ada datanya karena sebenarnya bagian itu telah keluar dari layar sebelum *frame* digeser menjadi di tengah. Kasus ini bisa dibiarkan begitu saja dengan mengisi daerah yang tidak ada datanya dengan warna hitam atau mosaic atau yang lebih baik dengan menggunakan data pada *frame* awal untuk "menambal" bagian pada *frame* kedua atau meng-*crop* kedua frame sehingga bagian yang keluar tidak akan perlu dimunculkan.



Gambar 3.1: Contoh video stabilizing yang mengisi bagian yang tidak ada data dengan hitam (dilingkari merah) [11]

Teknik penambalan maupun teknik *crop* pun masih memiliki kelemahan. Apabila guncangannya terlalu hebat, teknik *crop* akan mengakibatkan gambar yang diambil akan sangat kecil sehingga akan mengakibatkan berkurangnya kualitas gambar. Sedangkan teknik penambalan akan mengalami keanehan apabila ada objek baru yang masuk dari luar gambar di *frame* yang muncul belakangan namun belum ada di *frame* awal. Selain itu juga apabila guncangan yang terjadi mengakibatkan rotasi, teknik-teknik di atas nampaknya masih belum bisa mengatasinya.

Teknik-teknik yang telah disebutkan yang bisa dibilang masih cukup sederhana juga belum bisa mengatasi kasus apabila ternyata guncangan itu adalah pergeseran yang disengaja untuk pengambilan gambar. Namun beberapa aplikasi dan fitur *video stabilizing* yang lebih kompleks bisa mengatasinya dengan mengecek beberapa *frame* sekaligus untuk mengambil data tambalan dan untuk menentukan apakah pergeseran bagian tengah yang terjadi merupakan guncangan ataukah disengaja dan juga. Selain itu algoritma yang cukup kompleks yang dimiliki aplikasi-aplikasi tersebut juga telah bisa mengatasi guncangan yang mengakibatkan rotasi.

Pada makalah ini, kasus yang akan dibahas dan digunakan penerapan pattern matching hanyalah algoritma *video stabilizing* yang sederhana yang sebatas untuk mencari letak bagian tengah *frame* awal pada *frame* berikutnya dan menggeser *frame* yang belakangan hingga bagian tengah *frame* awal berada di tengah *frame* berikutnya pula, untuk teknik penambalan dan cropping tidak akan dibahas lebih detail meskipun teknik penambalan yang baik juga sebenarnya bisa mengaplikasikan pattern matching.

IV. Penerapan Pattern Matching Pada Video Stabilizing

Algoritma Brute Force terpilih karena pada kasus ini yang diperlukan adalah mencari letak pola yang merupakan bagian tengah dari suatu *frame* pada *frame* berikutnya dan karena pelompatan yang dilakukan algoritma KMP dan Boyer-Moore akan menyebabkan kebingungan ke arah mana pola akan melompat karena

pola berbentuk matriks. Algoritma Brute Force yang digunakan pun akan diberikan sedikit modifikasi mengenai adanya toleransi dan modifikasi mengenai posisi pencocokan awal dan pergeserannya.

4.1. Toleransi

Pada penggunaan *pattern matching* dalam kasus ini akan dipergunakan sebuah toleransi. Toleransi ini merupakan sebuah *range* dari dua titik pixel yang akan dikatakan cocok sehingga untuk mengatakan kedua pixel yang berada di pola dan di sampel adalah cocok kedua pixel tidak harus sama persis.

Toleransi diberlakukan karena pada video dapat terjadi perubahan yang tidak terduga, misalnya objek dalam video yang bergerak mengakibatkan sebuah pixel dapat berubah isinya meskipun gambar tidak goyang sama sekali atau terjadi perubahan pencahayaan yang mengakibatkan sejumlah pixel berubah datanya menjadi lebih gelap atau lebih terang.

4.2. Modifikasi

Modifikasi dilakukan pada posisi awal pencocokan pola dan pada pergeseran pola dalam *pattern matching* yang dilakukan, dari yang biasanya pola dimulai dari ujung kiri dan pergeseran dilakukan satu posisi ke kanan, maka dalam kasus ini pola dimulai dari tengah dan pergeseran yang dilakukan berbentuk spiral yang mengarah ke ujung luar gambar. Hal ini dilakukan karena letak pola pencarian yang merupakan bagian tengah dari sebuah *frame* tidak akan jauh dari bagian tengah *frame* berikutnya dan apabila pencarian pola dilakukan dari ujung kiri atas akan tidak efisien.

4.3. Algoritma

Dibuat 4 fungsi Brute Force *pattern matching*, masing-masing memiliki arah pergeseran yang berbeda dan jumlah maksimal pegeserannya, masing-masing fungsi juga melakukan pencocokan dengan urutan yang berbeda.

Struktur Matrikspixel (frame atau pola yang telah diubah):

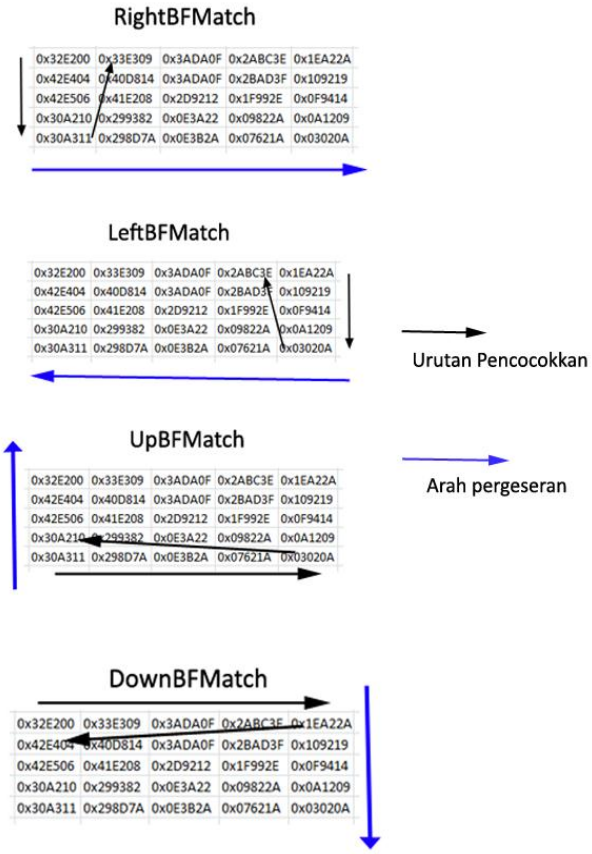
Mpixel: <matriks[length][width] of pixeldata, int length, int width>

Dengan getter untuk suatu data pixel di kordinat tertentu:

function datapixel(i: integer, j: integer) -> datapixel

Kemudian Prototipe dari keempat arah algoritma BFMatch adalah berikut

- RightBFMatch(Mpixel P, Mpixel T, int nshift, int xs, int ys); //pencocokkan dimulai dari kiri atas, kemudian ke baris bawah baru di kolom yang sama baru kemudian ke kolom berikutnya.
- LeftBFMatch(Mpixel P, Mpixel T, int nshift, int xs, int ys);// pencocokkan dimulai dari kanan atas, ke bawah kemudian ke kolom di kiri.
- DownBFMatch(Mpixel P, Mpixel T, int nshift, int xs, int ys); //pencocokkan dimulai dari kiri atas, kemudian ke kanan hingga baris pertama selesai kemudian ke baris di bawahnyadari kolom pertama lagi.
- UpBFMatch(Mpixel P, Mpixel T, int nshift, int xs, int ys); //pencocokkan dimulai dari kiri bawah hingga baris selesai baru ke baris di atasnya dimulai dari kolom pertama lagi.



Gambar 4.1: Ilustrasi keempat fungsi BFMatch modifikasi dengan contoh pola berukuran 5x5 datapixel

Pada iterasi pertama, nshift bernilai 1 dan dilakukan RightBFMatch dan saat RightBFMatch selesai, dilakukan DownBFMatch. Setelah DownBFMatch selesai, nshift di-*increment* dan dilakukan LeftBFMatch yang diteruskan oleh UpBFMatch dan kemudian nshift diincrement lagi. Langkah tersebut diiterasi lagi hingga ditemukan pola yang cocok atau keluar dari layar.

Gambar 4.2: Urutan pencocokkan gambar dari tengah secara spiral untuk satu iterasi, ukuran pola dan pergeseran diperbesar

Gambar 4.3: Frame 1 sebagai acuan, dan frame 2 sebelum dan sesudah *stabilizing*

Setelah pola ditemukan pada *frame* kedua, dilakukan proses *alignment* yang menggeser seluruh *frame* agar pola yang ditemukan berada di tengah layar dan bagian yang tidak ada datanya dibuat hitam dan yang keluar dari layar dipotong. Hal ini dilakukan seterusnya untuk *frame* ketiga dengan patokan *frame* kedua dan seterusnya.

4.4. Pseudo-code

Contoh untuk salah satu arah BFMatch yaitu RightBFMatch

```
RightBFMatch(Mpixel P, Mpixel T, int nshift, int xs, int ys){
  int Px = P.Length();
  int Py = P.Width();
  int ix = xs ; int iy = ys; //xs:xstart dan y:ystart
  int jy;
  int jx = 0;
  boolean stop=false;
  for(int k=0; k <= (nshift); k++) {
    while(jx<Px && !stop){
        jy=0;
        while((jy < Py) && P.pixel(jx,jy).match (T.pixel(ix+jx,iy+jy))) {//fungsi == diganti dengan match untuk yang mengatasi toleransi
                jy++;
        }
        if(jy<Py){
                stop=true;
        }
        else{
                jx++;
        }
    }
  }
  if( jx==Px && jy==Py){
     return Point(jx-Px,jy-Py); //match sampai habis
  }
  else{
     return Point(-1,-1); //tidak match
  }
   }
```

Kemudian berikut adalah PseudoCode untuk algoritma video stabilizing pada satu tahap (*frame* 2 dengan *frame* 1). Asumsi koordinat 0,0 terletak pada ujung kiri atas, dan koordinat maksimal berada pada ujung kanan bawah.

```
VideoStabilize(Frame F1, Frame F2){
    pixeldata[][] P = GenerateCenterPattern(F1,80,80); //menggenerate pola yang dicari dari bagian tengah F1 berukuran 80x80
    pixeldata[][] T = GenerateMatrixofPixelData(F2);
    int nshift = 1;
    boolean found=false;
    Point P;
    int xs = T.Xcenter()-40; //x start diatur dari pusat T dikurang 0,5* ukuran pattern
    int ys = T.Ycenter()-40; //y start diatur dari pusat T dikurang 0,5* ukuran pattern
    while(xs<=T.maxX()-P.length() && ys<=TmaxY()-P.Width()){
        //satu iterasi dimulai
        P = RightBFMatch(P,T,nshift,xs,ys);
        if(P!=<-1,-1>){
                break;
        }
        xs = xs + nshift + 1;
        P = DownBFMatch(P,T,nshift,xs,ys);
        if(P!=<-1,-1>){
                break;
        }
        ys = ys + nshift + 1;
        nshift++;
        P = LeftBFMatch(P,T,nshift,xs,ys);
        if(P!=<-1,-1>){
                break;
        }
        xs = xs – nshift – 1;
        P = UpBFMatch(P,T,nshift,xs,ys);
        if(P!=<-1,-1>){
                break;
        }
        ys = ys – nshift – 1;
        nshift ++;
```

```
    }
  If(xs<=T.maxX()-P.length()  &&  ys<=TmaxY()-P.Width()){ //ditemukan match sebelum diujung
      AlignFrame(F2,P,80,80); //F2 diletakkan sedemikian rupa hingga titik kiri atas bagian tengahnya yang berukuran 80x80 berada  di P
  }
  else{
      //tidak lakukan apa-apa, biarkan F2 seperti apa adanya
  }
}
```

V. KESIMPULAN

Video Stabilizing sederhana melibatkan pelacakan pola bagian tengah pada *frame* awal di *frame* berikutnya dan kemudian meng-*align frame* kedua sedemikiran rupa hingga bagian tengah dari *frame* awal berada di bagian tengah *frame* kedua juga. Tahap pelacakan pola dapat dilakukan dengan *pattern matching* yang menggunakan algoritma Brute Force yang dimodifikasi.

Modifikasi yang membuat pencocokkan dimulai dari tengah dan men-spiral hingga keluar layar meningkatkan efisiensi karena letak pola bagian tengah *frame* awal tidak akan jauh dari bagian tengah *frame* berikutnya sehingga akan lebih efektif dibanding mencocokkan dari ujung kiri atas ke kanan bawah.

Mengingat elemen-elemen yang akan dicocokkan pada kasus ini adalah warna yang terdiri dari kombinasi 3 warna utama yang masing-masing berkisar dari 0-255 sehingga terdapat 16.777.216 kemungkinan warna, maka Algoritma brute force merupakan algoritma yang cukup baik untuk digunakan karena algoritma brute force efektif apabila jenis dari elemen pola yang dicocokkan banyak.

Algoritma *video stabilizing* yang dibahas pada makalah ini sudah cukup bisa membuat *video stabilizing* yang sederhana. Namun v*ideo stabilizing* yang dilakukan pada makalah ini masih banyak kekurangan dan baru merupakan satu bagian dari *Video Stabilizing* yang lebih lengkap dan kompleks. Algoritma ini masih bisa dikembangkan lebih lanjut dengan banyak cara untuk menghasilkan teknik *video stabilizing* yang lebih lengkap dan kompleks.

VI. PENUTUP

Ucapan terima kasih penulis berikan kepada Tuhan Yang Maha Esa, Institut Teknologi Bandung dan kedua dosen mata kuliah IF2211-Strategi Algoritma yaitu Bapak Rinaldi Munir dan Ibu Masayu Leyla Khodra yang telah menugaskan penulis membuat makalah ini. Diharapkan makalah ini dapat memberi manfaat bagi pembaca dan menginspirasi untuk dibuat pengembangannya yang lebih lanjut.

REFERENSI

1. Hsiung, Yu-Lu. *Introduction to Digital Video*. University of Texas.
2. Octavianus, Andrian. *Pencarian Potongan Gambar Menggunakan Algoritma Boyer Moore*.
3. Gotama, Willy. *Image Matching untuk Penggabungan Panoramic Picture*.
4. http://en.wikipedia.org/wiki/Pixel, diakses pada 18 Mei 2014, pukul 17.42
5. http://en.wikipedia.org/wiki/Frame_buffer, diakses pada 18 Mei 2014, pukul 18.02
6. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/Pencocokan%20String%20(2014).ppt

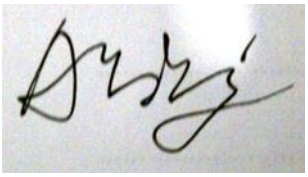
7.http://commons.wikimedia.org/wiki/File:CFA_Pattern_fuer_quadratische_und_rechteckige_Pixel.png, diakses pada 18 Mei 2014, pukul 19.02

8. http://www.engineersgarage.com/mygarage/how-video-is-displayed-on-screen, diakses pada 18 Mei 2014, pukul 20.17
9. http://www.qweas.com/downloads/audio/video-tools/screenshots-virtualdub.html, diakses pada 18 Mei 2014, pukul 21.09
10. http://en.wikipedia.org/wiki/Image_stabilization, diakses pada 17 Mei 2014, pukul 18.12
11. http://www.roborealm.com/help/Stabilize.php, diakses pada 17 Mei 2014, pukul 22.14

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Mei 2014



Ardi Wicaksono (13512063)