Laporan Tugas Akhir PPCD

Object Trcaking menggunakan OpenCV dan Arduino

Ardhi Maarik (G64120032), Sarah Shanaz Shaztika (G64120037), Muhammad Syarif Radhi (G64120101), Adek Ayu putri Juliani (G64120129)

PENDAHULUAN

Latar Belakang

Perkembangan teknologi dewasa ini semakin maju seiring dengan meningkatnya kebutuhan dan pekerjaan yang dilakukan manusia. Banyak waktu yang dihabiskan hanya untuk menyelesaikan pekerjaan yang sebenarnya dapat dilakukan secara otomatis oleh sebuah sistem. Sistem pengendali otomatis merupakan salah satu solusi alternatif untuk menyelesaikan permasalahan tersebut. Sistem pengendali otomatis adalah suatu sistem yang dapat mengenali objek dan memberikan aksi kepada objek tersebut secara otomatis.

Salah satu kasus yang akan dibahas dalam penelitian ini adalah video tutorial. Pembuatan video tutorial biasanya terdiri dari pengajar yang akan memberikan materi, dan orang yang bertanggung jawab dalam pengambilan gambar. Pengambilan gambar harus memperhatikan setiap *angle*, *camera movement*, dan *framing*. *Camera movement* adalah pengambilan gambar yang memperhatikan fokus objek dan jarak antara objek dengan kamera. Fokus objek pada video tutorial biasanya ada dua kondisi. Pertama, kamera akan fokus pada wajah saat pengajar menghadap ke arah kamera. Kedua, fokus kamera kan meluas saat pengajar membelakangi kamera seperti saat pengajar sedang menulis di papan tulis, fokus kamera akan lebih luas untuk merekam apa yang sedang dilakukan oleh pengajar.

Berdasarkan kasus di atas, dibutuhkan sistem pengendali otomatis yang dapat melakukan pengambilan gambar secara otomatis terhadap objek (*object tracking*) tanpa bantuan manusia. Pada penelitian ini akan dibangun sebuah sistem yang dapat menggerakkan kamera untuk mengambil gambar dengan memperhatikan dua kondisi. Kondisi pertama adalah mendeteksi wajah, kondisi ini untuk mengambil fokus gambar saat pengajar mengahadap ke aras kamera. Kondisi kedua adalah mendeteksi warna tertentu, kondisi ini untuk mengambil fokus apa yang dilakukan pengajar saat tidak terdeteksi wajah.

Perumusan Masalah

Perumusan masalah dalam penelitian ini adalah:

1. Bagaimana cara mengenali objek menggunakan OpenCV?
2. Parameter apa saja yang digunakan untuk mengenali objek pada citra digital?
3. Bagaimana cara mengimplementasikan kamera yang dapat mengikuti objek dengan warna tertentu?

Tujuan Penelitian

Tujuan penelitian ini adalah dihasilkannya sebuah program yang dapat:

1. Mengenali warna objek secara otomatis
2. Membedakan objek-objek secara otomatis menggunakan kamera yang digerakkan oleh sebuah *mini controller arduino* yang terintegrasi dengan openCV sebagai *library* program pemrosesan citra
3. Mengikuti pergerakan objek menggunakan sebuah kamera.

Manfaat Penelitian

Penelitian ini diharapkan dapat membantu pengembangan interface webcam atau CCTV untuk mengidentifikasi dan memilah benda yang diinginkan dari background yang tertangkap oleh kamera atau CCTV, serta melakukan proses identifikasi jika dikembangkan lebih lanjut.

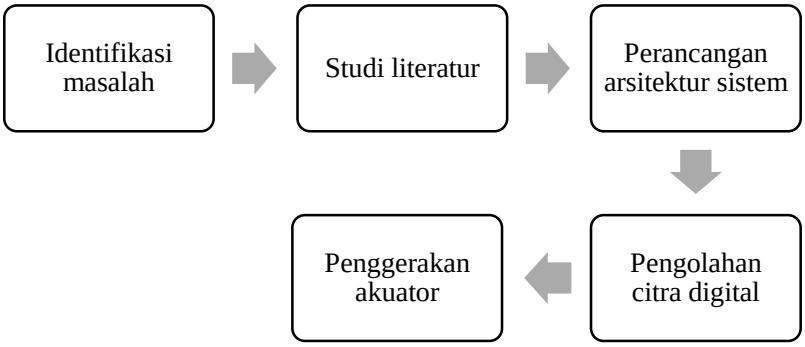
Ruang Lingkup Penelitian

Pengembangan menggunakan Arduino dan Servo sebagai akuator. *Library* yang digunakan untuk memproses citra adalah OpenCV.

METODE PENELITIAN

Tahapan Penelitian

Tahapan penelitian yang dilakukan dalam pembuatan *object tracking* menggunakan OpenCV pada Arduino seperti pada Gambar 1.



Gambar 1 Diagram Alir penelitian

Identifikasi masalah

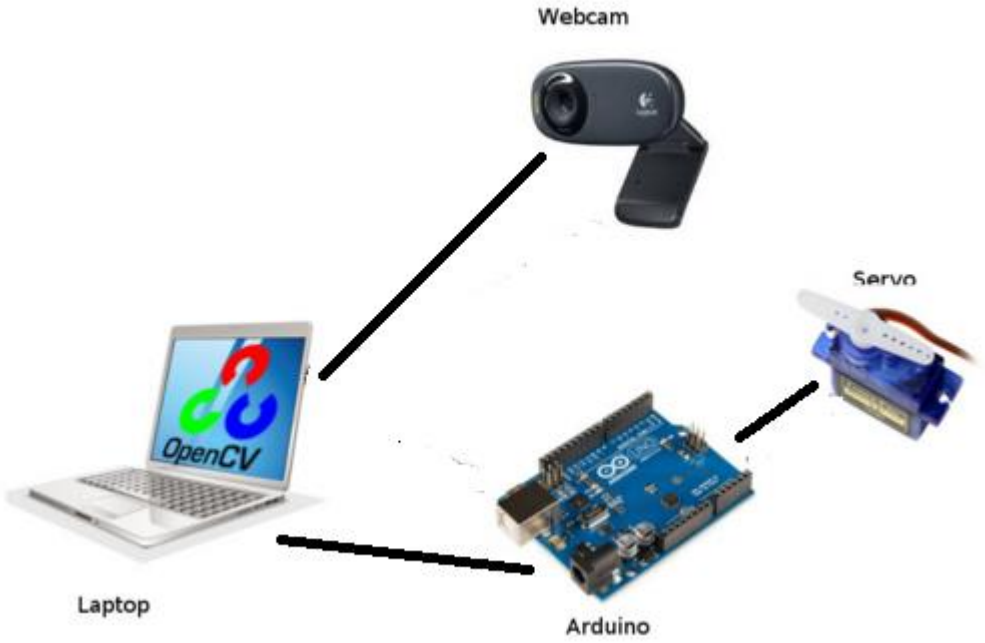
Pada tahap ini dilakukan identifikasi terhadap masalah-masalah yang ada dalam identifikasi objek. beberapa masalah dalam identifikasi objek adalah faktor atau variable apa yang dapat mengidentifikasi objek, dan metode apa yang digunakan untuk mengidentifikasi objek. Identifikasi masalah dilakukan agar penelitian ini dapat membantu menyelesaikan masalah yang ada di lapangan.

Studi literatur

Studi literatur dilaksanakan untuk mencari informasi tambahan untuk melengkapi data-data dari masalah yang ada. Pada tahap ini dilakukan pencarian dan pengumpulan literatur yang berguna untuk melengkapi data-data dan memperkuat ide dasar, serta observasi terhadap sistem-sistem sejenis yang memiliki kesamaan dalam hal objek kajian, metode yang digunakan, dan lingkungan pengembangan yang digunakan. Tahap ini juga nantinya berguna untuk memahami sistem yang akan dibuat secara lebih mendalam.

Perancangan arsitektur sistem

Pada tahap ini dilakukan perancangan arsitektur sistem untuk *tracking object*. Adapun rancangan sistem yang dibuat seperti pada Gambar 2.

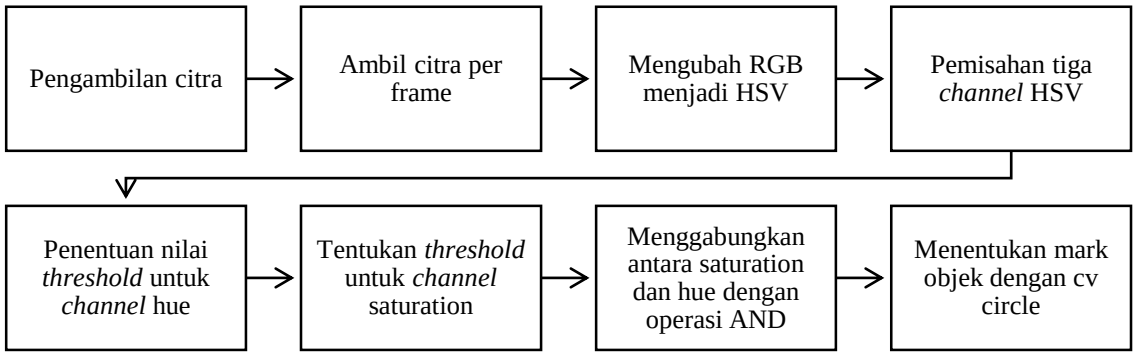


Gambar 2 Perancangan arsitektur sistem

Arsitektur perancangan sistem dimulai dengan meyambungkan *web-cam* dengan laptop yang digunakan dalam penelitian ini. Setelah menyambungan *web-cam* dengan laptop telah berhasil dilakukan, proses selanjutnya adalah mengimplementasikan kode program untuk proses pengolahan citra menggunakan *library* OpenCV. Selanjutnya, OpenCV digunakan untuk mengirimkan koordinat objek ke servo. Servo menerima informasi koordinat objek lalu menggerakan akuator sesuai perpindahan koordinat objek.

Pengolahan citra digital

Pengolahan citra digital dilakukan dengan tujuan mendapatkan koordinat objet yang akan di-*tracking*. Langkah-langkah yang dilakukan dalam pengolahan citra digital seperti pada Gambar 3.



Gambar 3 Tahap pengolahan citra digital

Pengambilan citra

Pada tahap ini dilakukan proses pengambilan gambar secara *real time* dari lingkungan menggunakan *web-cam*.

Ambil citra per frame

Pengambilan citra per frame dilakukan untuk mengambil frame dari video yang telah direkam.

Mengubah RGB menjadi HSV

Warna pada model RGB (*Red*, *Green*, *Blue*) merupakan hasil campuran dari warna-warna primer merah, hijau, dan biru berdasarkan komposisi tertentu. Model warna HSV mendefinisikan warna dalam terminologi *hue*, *saturation* dan *value*. *Hue* menyatakan warna sebenarnya, s*aturation* menyatakan kemurnian atau kekuatan warna dan *value* menyatakan kecerahan dari warna.

Pada tahap ini dilakukan pengubahan model warna dari RGB menjadi HSV. Pengubahan warna ini didasarkan pada kelebihan yang dimiliki model warna HSV, yaitu dapat menangkap warna-warna yang sama dengan yang ditangkap oleh indera manusia.

Pengubahan warna RGB menjadi HSV pada OpenCV menggunakan persamaan-persamaan seperti berikut ini.

$$r = \frac{R}{(R+G+B)} \;; g = \frac{G}{(R+G+B)} \;; b = \frac{B}{(R+G+B)} \tag{1}$$

$$V = \max(r, g, b) \tag{2}$$

$$S = \begin{cases} 0 & ; jika\ V = 0 \\ 1 - \frac{\min(r,g,b)}{V} & ; jika\ V > 0 \end{cases} \tag{3}$$

$$H = \begin{cases} 0, & S = 0 \\ \frac{60*(g-b)}{S*V}, & V = r \\ 60 * \left[2 + \frac{b-r}{S*V}\right], & V = g \\ 60 * \left[4 + \frac{r-g}{S*V}\right], & V = b \end{cases} \tag{4}$$

$$H = H + 360 \;; jika\ H < 0 \tag{5}$$

Persamaan (1) menunjukkan pengubahan RBG ke dalam *floating point*. Selanjutnya, pengubahan RGB menjadi HSV ditunjukkan pada persamaan (2), (3), dan (4). Persamaan (2) menentukan nilai V, persamaan (3) menentukan nilai S, dan persamaan (4) menentukan nilai H. Persamaan (4) dapat menimbulkan nilai negatif, oleh karena itu, harus dinormalisasi dengan persamaan (5).

Pemisahan tiga *channel* HSV

Pada tahap ini dilakukan pemisahan *channel* untuk pemilihan warna. Citra yang sudah diubah ke dalam model warna HSV dipisahkan kedalam tiga *channel* yang berbeda yaitu hue, saturation, dan value. Pemisahan *channel* ini dilakukan untuk mendapatkan nilai hue dan saturation dari citra. Nilai hue ini akan digunakan untuk memilih warna pada objek yang akan di *tracking*. Sedangkan nilai saturation digunakan untuk kemurnian suatu warna.

Penentuan nilai *threshold* untuk *channel* hue

Nilai *threshold* ditentukan untuk mendapatkan warna objek yang ingin di *tracking*. Penentuan nilai *threshold* dilakukan secara manual sesuai dengan objek yang akan di – *tracking*.

Tentukan *threshold* untuk *channel* saturation

Nilai *threshold* untuk *channel* saturation digunakan untuk mengambil kemurnian warna. Kemudian nilai saturation digunakan untuk menjelaskan warna objek yang akan di *tracking* sehingga objek tersebut terdeteksi sesuai atau mendekati warna aslinya.

Menggabungkan antara saturation dan hue dengan operasi AND

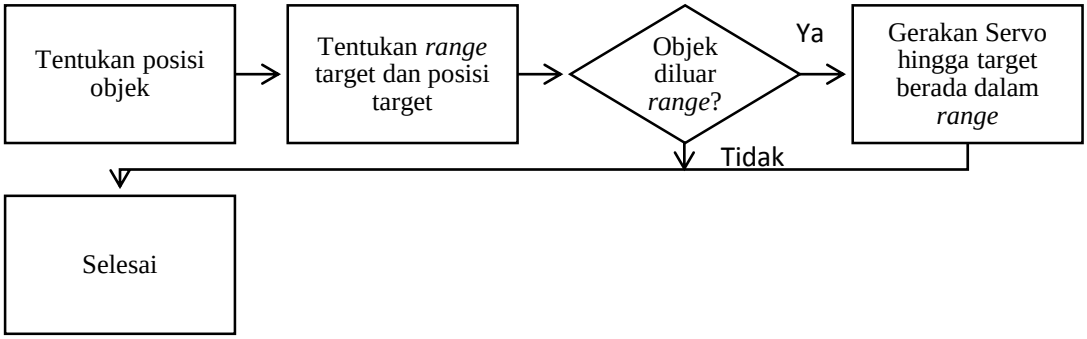
Penggabungan nilai hue dan saturation bertujuan untuk mendapatkan warna yang paling mirip dengan objek asli yang akan di *tracking*. Penggabungan kedua nilai tersebut dilakukan dengan menggunakan operasi AND yang bertujuan mendapatkan irisan nilai citra dari hue dan saturation.

Menentukan mark objek dengan cv circle

Setelah nilai irisan antara hue dan saturation diperoleh, langkah selanjutnya adalah menentukan posisi objek yang akan di *tracking*. Posisi objek (koordinat) diperoleh dari pengolahan citra dengan menggunakan OpenCV. Posisi objek ini merupakan nilai tengah dari objek yang akan di *tracking*.

Penggerakan akuator

Penggerakan akuator dilakukan dalam beberapa tahap seperti pada Gambar 3. Tahap awal dimulai dengan mendapatkan posisi objek dari hasil pengolahan citra pada OpenCV. Posisi objek yang diperoleh merupakan koordinat nilai tengah dari objek yang akan dilakukan *tracking*.



Gambar 4 Tahap pengolahan citra digital

Langkah selanjutnya adalah menentukan jarak target dan posisi target. Tahap berguna untuk penggerak kamera terhadap objek. Jika objek berada di luar jarak yang ditentukan maka servo akan menggerakan kamera hingga objek berada dalam jarak yang ditentukan. Namun, jika tidak berada diluar jarak yang ditentukan maka hal ini menunjukkan bahwa kamera telah menangkap objek sehingga tidak ada pergerakan dari servo.

HASIL DAN PEMBAHASAN

Berdasarkan percobaan yang telah dilakukan, yaitu *tracking object* terhadap benda bergerak, citra objek akan diproses dalam bentuk video dengan kode program sebagai berikut.

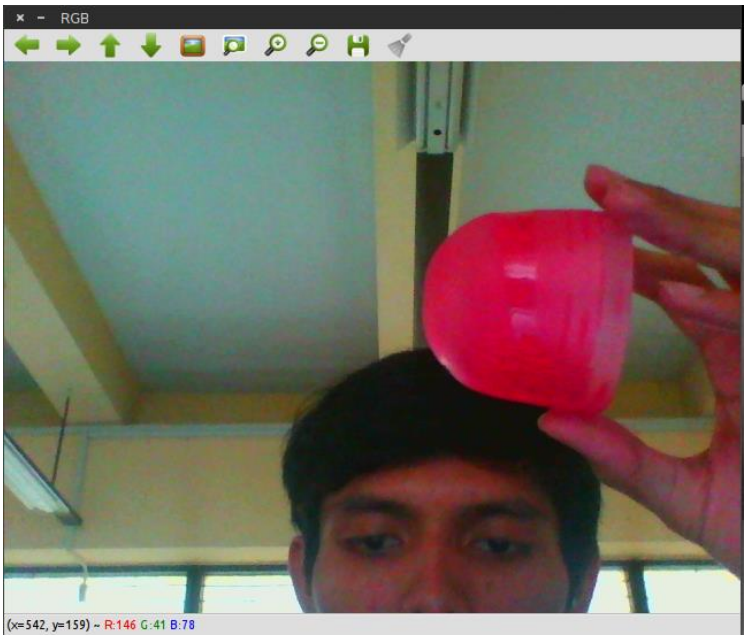
```
CVCapture* capture = cvCreateCameraCapture(0)
```

Video yang diterima sebagai input memiliki banyak *frame* didalamnya. Sebelum *frame* diproses, dilakukan proses *mirroring* terhadap *frame* untuk menyamakan posisi antara objek real dengan output objek yang akan ditampilkan. Adapun kode program untuk memproses tiap frame dan di-*mirrorkan* sebagai berikut.

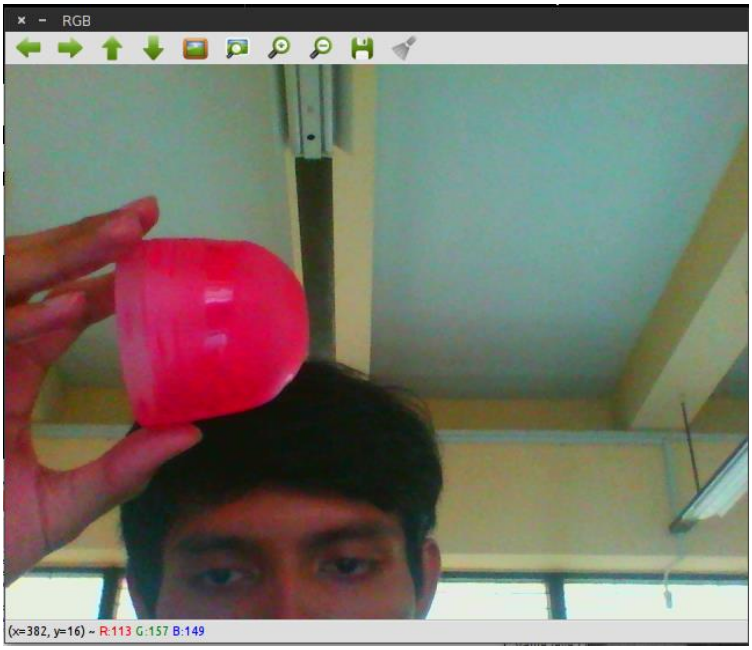
```
IplImage* frame = cvQueryFrame(capture):

cvFlip(frame,frame,1)
```

Gambar 5 merupakan gambar original, sedangkan Gambar 6 merupakan gambar setelah dilakukan *mirroring*.



Gambar 5 Tampilan *original*



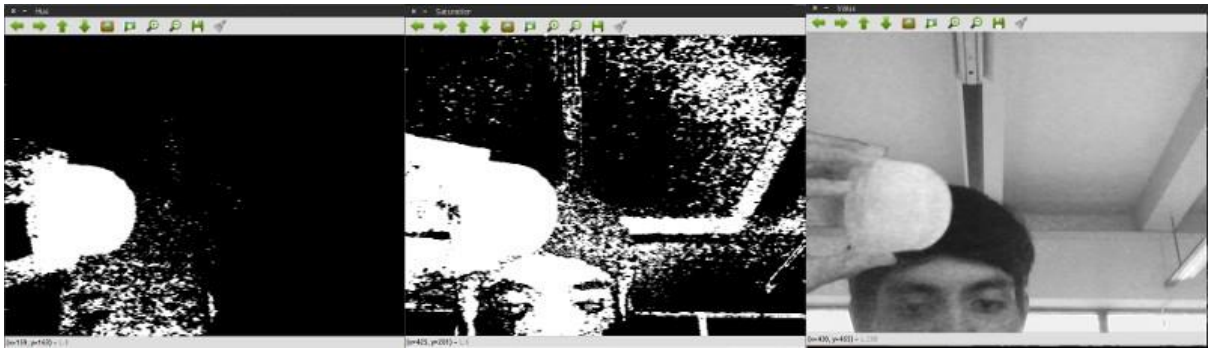
Gambar 6 Tampilan hasil *mirroring*

Kemudian, proses selanjutnya adalah melakukan konversi model warna RGB menjadi model warna HSV dengan kode program sebagai berikut.

```
cvCvtColor(frame, hsv, CV_BGR2HSV)
```

Nilai HSV yang diterima akan dipisah kedalam tiga *channel*, yaitu hue, saturation, dan value. Adapun kode program dan hasil output ditunjukkan pada Gambar 7.

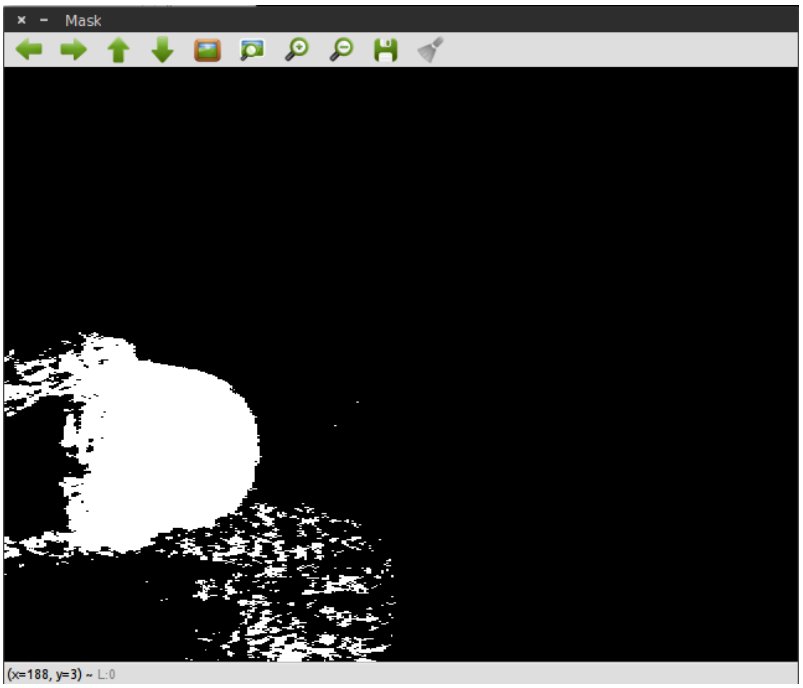
```
cvSplit(hsv, chan_hue, chan_sat, chan_val, 0)
```



Gambar 7 Hasil pemisahan 3 *channel* HSV

Setelah dilakukan pemisahan 3 *channel* HSV, akan digabungkan *channel* hue dan *channel* saturation menggunakan operator AND dengan kode program dan hasil output yang ditunjukkan oleh Gambar 8.

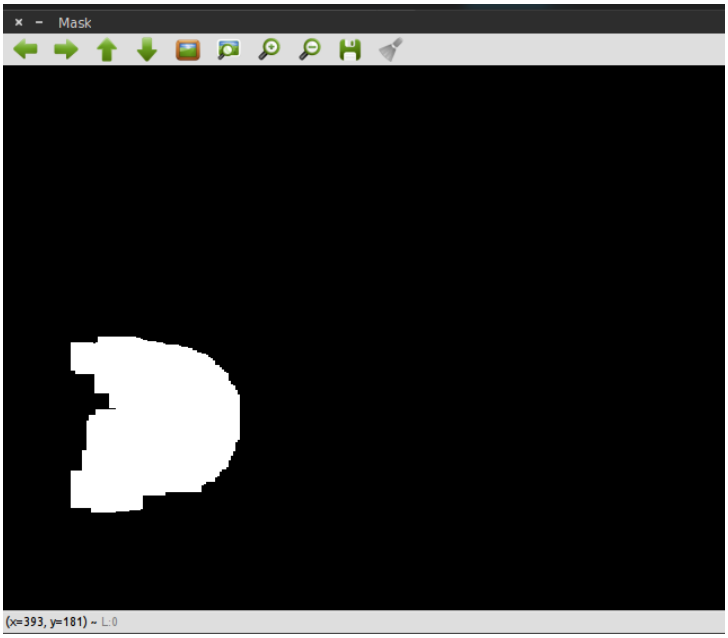
cvAnd(chan_hue, chan_sat, chan3)



Gambar 8 Hasil penambahan *channel* hue dan saturation

Setelah itu, langkah selanjutnya adalah melakukan proses *opening*. Proses ini dimulai dengan melakukan erosi kemudian dilasi pada citra. Kode program dah hasil output ditunjukkan pada Gambar 9.

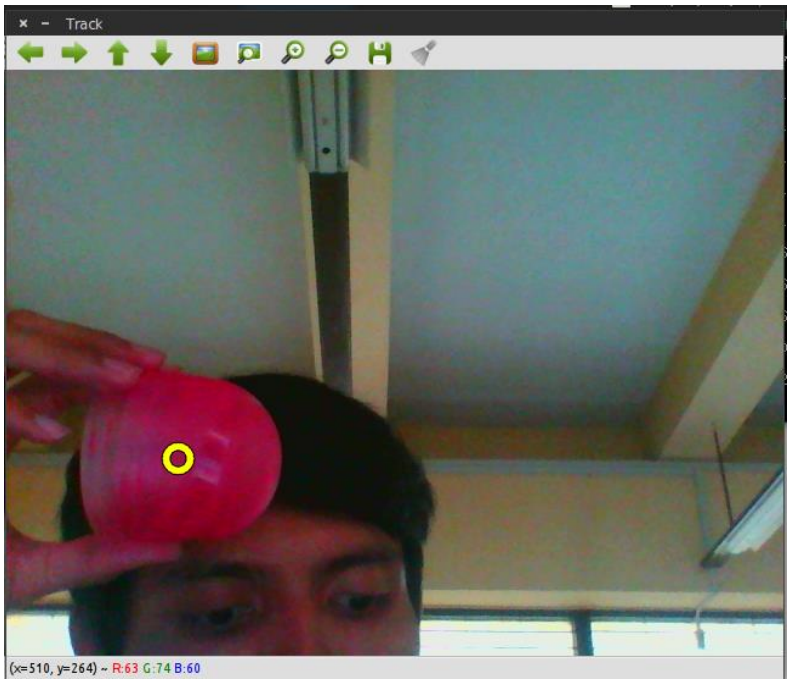
cvErode(chan3, chan3, 0, 9); cvDilate(chan3, chan3, 0,9)



Gambar 9 Hasil proses *opening*

Langkah berikutnya adalah mencari koordinat nilai tengah dari objek yang akan di *tracking*. Setelah koordinat ditemukan, maka akan ditandai dengan gambar lingkaran yang menunjukkan koordinat nilai tengah dari gambar. Adapun kode program dan hasil output sebagai berikut.

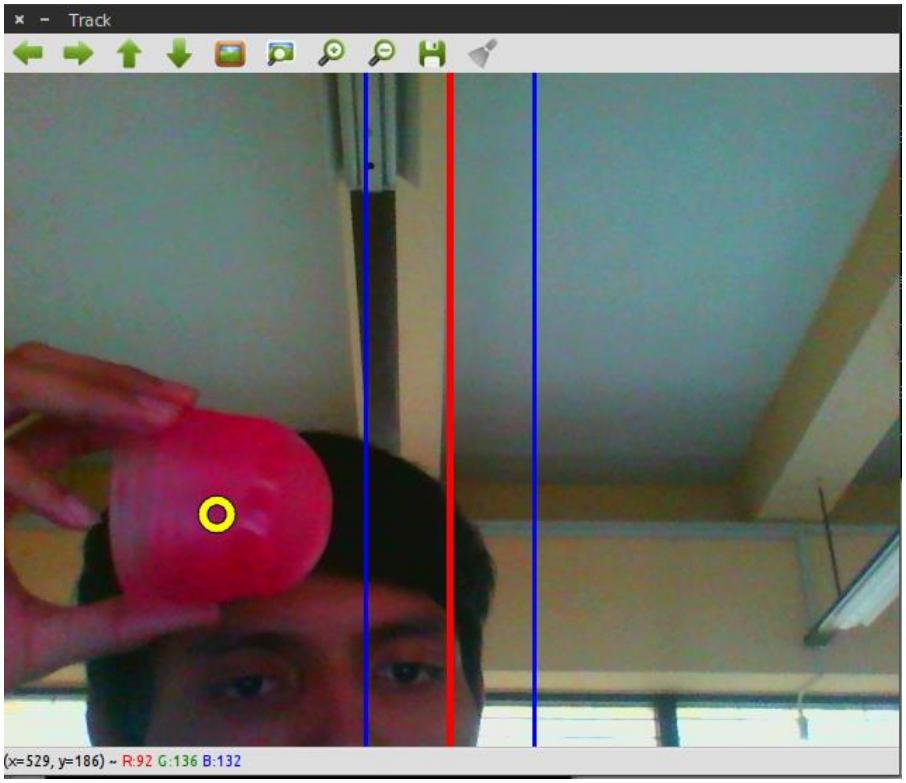
```
cvMoments(chan3, &moments, 1);
int r = sqrt(moments.m00);
int x = moments.m10/moments.m00; int y = moments.m01/moments.m00;
if (x>0&&y>0){cvCircle(monitor, cvPoint(x,y), r, cvScalar(0, 0, 0), 4, CV_AA, 0);
cvCircle(monitor, cvPoint(x,y), r, cvScalar(255, 255, 255), 2, CV_AA, 0)}
```



Gambar 10 Hasil penandaan objek

Pada langkah terakhir, dilakukan penentuan jarak posisi piksel untuk mengarahkan objek ke area yang sudah ditentukan. Berikut kode program dan hasil output.

```
cvLine(result, cvPoint(target – 60,0), cvPoint(target – 60, 480), cvScalar(255, 0, 0), 2);
cvLine(result, cvPoint(target, 0), cvPoint(target, 480), cvScalar(0, 0, 255), 3);
cvLine(result, cvPoint(target +60, 0), cvPoint(target +60, 480), cvScalar(255, 0, 0), 2);
```



Gambar 11 Hasil pengaturan jarak objek

KESIMPULAN

Dari percobaan yang kami lakukan, dapat disimpulkan bahwa pengenalan objek dapat dilakukan dengan cara mengekstraksi warna dari objek amatan. Parameter yang dapat digunakan untuk proses ekstraksi warna adalah nilai HSV dari setiap piksel. Penggunaan model warna HSV menghasilkan output warna objek yang sesuai dengan warna objek aslinya. Penentuan jarak atau *range* dari posisi awal dengan posisi target objek membuat kamera dapat mengikuti objek dengan baik.

LAMPIRAN

```
#include <unistd.h>
#include <iostream>
#include <sys/time.h>
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

//-------- PID parameters --------

// See http://en.wikipedia.org/wiki/PID_controller

// These values must be chosen CAREFULLY. The strategy to find good
// values is to set `ci' and `cd' to 0.0, then try to find a value of
// `cp' that works the best (without too much oscillation) then, from
// that, lower `cp' and increase `cd' until the system is able to
// stalibilize more quickly. Increase `ci' if the system take time to
// move to the target position. There are more complexes method to
// find the "right" values.

static double cp = 0.2;
static double ci = 0.0;
static double cd = 0.02;

int hue_level_start = 157;  // The minimum Hue level.
int hue_level_stop = 198;   // The maximum Hue level.
int sat_level = 78;        // The minimum saturation level.
int val_level = 130;        // value level
int target = 320;           // Target position.


/*
 * Get the current time in seconds
 */
static double
gettime()
{
    struct timeval tv;
    gettimeofday(&tv, 0);
    return tv.tv_sec + tv.tv_usec / 1e6;
}

void buatwindow()
{
    //window name
    cvNamedWindow("RGB", CV_WINDOW_AUTOSIZE);   //Window to display the input image.
    cvNamedWindow("Mask", CV_WINDOW_AUTOSIZE);  //Window to display the mask (the selected part of the input image)
    cvNamedWindow("Track", CV_WINDOW_AUTOSIZE); //Window to display the tracking state
    cvNamedWindow("Settings", 0);               //window to display traking bar
    cvNamedWindow("Hue", CV_WINDOW_AUTOSIZE);   //Window to display the input image.
    cvNamedWindow("Saturation", CV_WINDOW_AUTOSIZE);   //Window to display the input image.
    cvNamedWindow("Value", CV_WINDOW_AUTOSIZE);   //Window to display the input image.

    //position of window
    cvMoveWindow("RGB", 0, 505);        //position of window to display the input image.
    cvMoveWindow("Mask", 0, 0);         //position of window to display the mask (the selected part of the input
image)
    cvMoveWindow("Settings", 652, 505); //position of window to display traking bar
    cvMoveWindow("Track", 646, 0);      //position of window to display the tracking state
}

void buattrackbar()
{
    //create trackbar
    cvCreateTrackbar("Hue level (start)", "Settings", &hue_level_start, 255, 0);    // trackbar of minimum Hue level
    cvCreateTrackbar("Hue level (stop)", "Settings", &hue_level_stop, 255, 0);      // trackbar of maximum Hue level
    cvCreateTrackbar("Saturation level", "Settings", &sat_level, 255, 0);           // trackbar of minimum saturation
level
    cvCreateTrackbar("Value level", "Settings", &sat_level, 255, 0);           // trackbar of minimum saturation
level
    cvCreateTrackbar("Target", "Settings", &target, 640, 0);                        // trackbar of target position
}

int main()
{
    CvCapture*          capture = cvCreateCameraCapture(0);
    if (capture == 0)
    {
        std::cerr << "Failed to open the camera.\n";
        return 1;
    }
    buatwindow();   //show window
```

```
    buattrackbar(); //show trackbar


    IplImage* chan_hue = 0;
    IplImage* chan_sat = 0;
    IplImage* chan_val = 0;
    IplImage* chan3 = 0;
    IplImage* hsv = 0;
    IplImage* monitor = 0;
    IplImage* result = 0;

    double    angle = 1500.0; // the position of the camera
    int       tcolor = 0; // target color - Used to switch to predefined hue levels.
    double    last_time = 0.0; // last time we updated PID
    int       last_known_x = 320; // last known position of the target.
    double    last_error = 0.0;
    double    i = 0.0; // integral term (here because it is accumulating)
    int       last_sent_value = -1;

    FILE* serial = fopen("/dev/ttyACM0", "w"); //open port arduino
    if (serial == 0)
    {
        printf("Failed to open serial port\n");
    }
    sleep(1);

    while(1)
    {
        //-------- Get the input image --------
        IplImage* frame = cvQueryFrame(capture); //process each frame in video capture
        if (frame == 0) break;

        cvFlip(frame, frame,1); //flip frame left-right   =>   right-left

        cvShowImage("RGB", frame); // show image to window RGB

        //-------- Allocate images --------
        if (hsv == 0)
        {
            // Allocate images in first iteration.
            hsv        = cvCreateImage(cvGetSize(frame), 8, 3); // 8 bits, 3 channels, to store image convertion from
BGR2HSV
            chan_hue   = cvCreateImage(cvGetSize(frame), 8, 1); // 8 bits, 1 channels, to store hue channel from
convertion
            chan_sat   = cvCreateImage(cvGetSize(frame), 8, 1); // 8 bits, 1 channels, to store saturation channel
from convertion
            chan_val   = cvCreateImage(cvGetSize(frame), 8, 1); // 8 bits, 1 channels, to store value channel from
convertion
            chan3      = cvCreateImage(cvGetSize(frame), 8, 1); // 8 bits, 1 channels, to store result of process 2
channel before
            monitor    = cvCreateImage(cvGetSize(frame), 8, 3); // 8 bits, 3 channels,
            result     = cvCreateImage(cvGetSize(frame), 8, 3); // 8 bits, 3 channels,
        }

        //-------- Process the input image --------

        cvCvtColor(frame, hsv, CV_BGR2HSV); // convert to HSV (Hue-Saturation-Value)
        cvSplit(hsv, chan_hue, chan_sat, chan_val, 0); // extract to Hue & Saturation

        // Create a mask matching only the selected range of hue values.
        if (hue_level_start <= hue_level_stop)
        {
            cvInRangeS(chan_hue, cvScalar(hue_level_start), cvScalar(hue_level_stop), chan_hue);
        }
        else
        {
            cvInRangeS(chan_hue, cvScalar(hue_level_stop), cvScalar(hue_level_start), chan_hue);
            cvSubRS(chan_hue, cvScalar(255), chan_hue);
        }
        // Create a mask matching only the selected saturation levels. greater then
        cvCmpS(chan_sat, sat_level, chan_sat, CV_CMP_GT); // Test Saturation
        //CMP_EQ src1 is equal to src2
        //CMP_GT src1 is greater than src2
        //CMP_GE src1 is greater than or equal to src2
        //CMP_LT src1 is less than src2
        //CMP_LE src1 is less than or equal to src2
        //CMP_NE src1 is unequal to src2

        cvShowImage("Hue", chan_hue); // show hue to window RGB
        cvShowImage("Saturation", chan_sat); // show saturatio to window RGB
        cvShowImage("Value", chan_val); // show value to window RGB

        cvAnd(chan_hue, chan_sat, chan3); // Merge masks
        cvErode(chan3, chan3, 0, 1); // Suppress noise
        cvDilate(chan3, chan3, 0, 9); // scale object
```

```
        //cvErode(chan3, chan3, 0,)

        // Find the position (moment) of the selected regions.
        CvMoments           moments;
        cvMoments(chan3, &moments, 1);
        int                 r = sqrt(moments.m00);
        int                 x = moments.m10/moments.m00;
        int                 y = moments.m01/moments.m00;

        //-------- Mask window --------

        // blue = hue selection, green = saturation selection, red = selected regions
        cvConvertScale(frame, monitor, .3, 0); // faded out input
        //cvSet(monitor, cvScalar(255, 0, 0), chan_hue); // blue overlay
        //cvSet(monitor, cvScalar(0, 255, 0), chan_sat); // green overlay
        //cvSet(monitor, cvScalar(0, 0, 255), chan3); // red overlay

        if (x > 0 && y > 0)
        {
            cvCircle(monitor, cvPoint(x, y), r, cvScalar(0, 0, 0), 4, CV_AA, 0); //lingkaran outline
            cvCircle(monitor, cvPoint(x, y), r, cvScalar(255, 255, 255), 2, CV_AA, 0); //lingkaran fill in
        }
        cvShowImage("Mask", chan3);

        //-------- Tracking state --------

        cvCopy(frame, result, 0); // input image
        cvLine(result, cvPoint(target - 60, 0), cvPoint(target - 60, 480), cvScalar(255, 0, 0), 2);
        cvLine(result, cvPoint(target, 0), cvPoint(target, 480), cvScalar(0, 0, 255), 3);
        cvLine(result, cvPoint(target + 60, 0), cvPoint(target + 60, 480), cvScalar(255, 0, 0), 2);

        CvPoint pt1((x), (y));
        CvPoint pt2((x), (y));
        cvRectangle(result, pt1, pt2, cvScalar(255,0,0),7,0);
        cv::Mat framemat = cv::cvarrToMat(frame);
//        cv::Rect myROI(10,10,100,100);
//        //cv::Mat cropRef(framemat, myROI);
//        //imshow(myROI);
//        cv::Rect roi(10,10,100,100);
//        cv::Mat image_roi = image(roi);
//        image_roi.copyTo(cropimage);
//        imshow(cropimage);
        //cv::Mat cropedImage;// = cv::Rect rrrr(10,10,100,100);
        //imshow(cropedImage);
        //cvGetRectSubPix(framemat,cropedImage,)
        if (x > 0 && y > 0)
        {
            cvCircle(result, cvPoint(x, y), 10, cvScalar(0, 0, 0), 6, CV_AA, 0);
            cvCircle(result, cvPoint(x, y), 10, cvScalar(0, 255, 255), 4, CV_AA, 0);
        }
        cvShowImage("Track", result);

        //-------- Handle keyboard events --------

        int key = cvWaitKey(33);
        if (key == 27) break;
        switch (key)
        {
        case 'r':
            // Reset the current position. This is used to check how fast
            // the system can return to the correct position.
            angle = 1500;
            break;
        case 't':
            // Quickly switch to a different tracking color (red or blue)
            tcolor = 1 - tcolor;
            if (tcolor == 0)
            {
                cvSetTrackbarPos("Hue level (start)", "Settings", 0);
                cvSetTrackbarPos("Hue level (stop)", "Settings", 12);
            }
            else
            {
                cvSetTrackbarPos("Hue level (start)", "Settings", 109);
                cvSetTrackbarPos("Hue level (stop)", "Settings", 116);
            }
            break;
        default:
            // ignore other keys.
            break;
        }

        //-------- PID processing --------

        // If the object is out of the window, use last known position to
```

```
        // find it.
        if (x < 0 || x > 640)
        {
            x = 2.5 * (last_known_x - 320) + 320;
        }
        else
        {
            last_known_x = x;
        }

        double              time = gettime();
        double              dt = time - last_time;
        if (last_time == 0.0)
            dt = 1.0;
        last_time = time;

        // the error we want to correct
        double              error = x - target;

        // the proportional term
        double              p = error * cp;

        // update the integral term
        i += error * dt * ci;

        // Clamp integral term
        if (i > 30.0)
            i = 30.0;
        else if (i < -30.0)
            i = -30.0;

        // the derivative term
        double              d = (error - last_error) / dt * cd;
        last_error = error;

        // the PID value
        double              pid = p + i + d;

        // Clamp PID
        if (pid < -100)
            pid = -100;
        else if (pid > 100)
            pid = 100;

        // Update the position from the PID value.
        angle += pid;

        // Clamp angle
        if (angle < 0)
            angle = 0;
        else if (angle > 2000)
            angle = 2000;

        //printf("pos = %d, P = %f, I = %f, D = %f, angle = %f, dt = %f\n", x, p, i, d, angle, dt);

        //-------- Send the position to Arduino --------

        if (serial != 0)
        {
            int                 current_value = angle;
            // Send the new position if it changed since the last time.
            if (current_value != last_sent_value)
            {
                fprintf(serial, "%d\n", current_value);
                printf("SENT %d\n", current_value);
                last_sent_value = current_value;
            }
        }
    }
    cvReleaseCapture(&capture);
}
```