**VII.2.1. Source Code Program Color.hpp**

#pragma once

#include <string>

#include <opencv2\highgui\highgui.hpp>

#include <opencv2\opencv.hpp>

using namespace std;

using namespace cv;

class Color{

private:

string type;

Scalar HSVmin, HSVmax;

Scalar Colored;

public:

Color();

Color(string name);

Scalar getHSVmin();

Scalar getHSVmax();

void setHSVmin(Scalar min);

void setHSVmax(Scalar max);

string getType(){return type;}

void setType(string t){type = t;}

Scalar getColor()

{return Colored;}

void setColor(Scalar z)

{Colored = z;}

~Color(void);

};

**VII.2.2. Source Code Program Color.cpp**

#include "Color.hpp"

Color::Color()

{

setType("Color");

setColor(Scalar(0,0,0));

}

Color::Color(string name){

setType(name);

if(name=="red")

{

setHSVmin(Scalar(141,0,0));

setHSVmax(Scalar(184,255,255));

setColor(Scalar(0,0,255));

}

if(name=="green")

{

setHSVmin(Scalar(34,50,50));

setHSVmax(Scalar(80,220,225));

setColor(Scalar(0,255,0));

}

if(name=="blue")

{

setHSVmin(Scalar(75, 25, 45));

setHSVmax(Scalar(130, 150, 200));

setColor(Scalar(255,0,0));

}

}

void Color::setHSVmin(Scalar min)

{Color::HSVmin = min;}

void Color::setHSVmax(Scalar max)

{Color::HSVmax = max;}

Scalar Color::getHSVmin()

{return Color::HSVmin;}

Scalar Color::getHSVmax()

{return Color::HSVmax;}

Color::~Color() ///Destructor

{}

**VII.2.3. Source Code Program main.cpp**

#include "Color.hpp"

#include <iostream>

#include <vector>

#include <opencv2\highgui\highgui.hpp>

#include <opencv2\opencv.hpp>

using namespace std;

using namespace cv;

Mat camera\_feed;

Mat filter\_feed;

Mat HSV\_filter;

Mat treshold;

string intToString(int number)

{

stringstream s;

s << number;

return s.str();

}

void Circle\_Detector(Color input\_color, Mat input, Mat &camerafeed)

{

vector<Vec3f> circles;

HoughCircles(input, circles, HOUGH\_GRADIENT,

2,

input.rows/4,

100,

100,

10, 100);

for( size\_t i = 0; i < circles.size(); i++ )

{

Vec3i a = circles[i];

Point center = Point(a[0], a[1]);

circle( camerafeed, center, 1, input\_color.getColor(), 3, LINE\_AA);

int radius = a[2];

circle( camerafeed, center, radius, input\_color.getColor(), 3, LINE\_AA); putText(camerafeed,format("%d",i+1),Point(a[0],a[1]-10),2,1,input\_color.getColor()); putText(camerafeed,format("r: %d",radius),Point(a[0]+9,a[1]+10),2,0.5,input\_color.getColor(),1);

putText(camerafeed,format("L: %d",(22/7)\*radius\*radius),Point(a[0]+9,a[1]+25),2,0.5,input\_color.getColor(),1);

putText(camerafeed,format("K: %d",(22/7)\*2\*radius),Point(a[0]+9,a[1]+35),2,0.5,input\_color.getColor(),1);

}

if(input\_color.getType()=="red")

{putText(camerafeed,intToString(circles.size()),Point(220,15),2,0.5,input\_color.getColor(),1);}

if(input\_color.getType()=="green")

{putText(camerafeed,intToString(circles.size()),Point(380,15),2,0.5,input\_color.getColor(),1);}

if(input\_color.getType()=="blue")

{putText(camerafeed,intToString(circles.size()),Point(530,15),2,0.5,input\_color.getColor(),1);}

putText(camerafeed,"Red Circle :",Point(100,15),2,0.5,Scalar(0,0,255),1);

putText(camerafeed,"Green Circle:",Point(260,15),2,0.5,Scalar(0,255,0),1);

putText(camerafeed,"Blue Circle :",Point(420,15),2,0.5,Scalar(255,0,0),1);

}

void Noise\_Reductor(Mat &input\_filter)

{

erode(input\_filter, input\_filter, getStructuringElement(MORPH\_ELLIPSE, Size(3, 3)) );

dilate( input\_filter, input\_filter, getStructuringElement(MORPH\_ELLIPSE, Size(8, 8)) );

dilate( input\_filter, input\_filter, getStructuringElement(MORPH\_ELLIPSE, Size(3, 3)) );

erode(input\_filter, input\_filter, getStructuringElement(MORPH\_ELLIPSE, Size(8, 8)) );

GaussianBlur(input\_filter, input\_filter, Size(9, 9), 2, 2 );

}

void drawlines(vector<Color> theObjects,Mat &frame, Mat &temp, vector< vector<Point> > contours, vector<Vec4i> hierarchy)

{

for(int i =0; i<theObjects.size(); i++)

{drawContours(frame,contours,i,theObjects.at(i).getColor(),1,LINE\_4,hierarchy);}

}

void Circle\_Tracker(Color colored,Mat threshold, Mat &camerafeed)

{

vector <Color> colors;

Mat temp;

threshold.copyTo(temp);

vector< vector<Point> > contours;

vector<Vec4i> hierarchy;

findContours(temp,contours,hierarchy,RETR\_LIST,CHAIN\_APPROX\_SIMPLE );

bool objectFound = false;

if (hierarchy.size() > 0) {

int numObjects = hierarchy.size();

if(numObjects<100)

{ for (int index = 0; index >= 0; index = hierarchy[index][0])

{ Moments moment = moments((Mat)contours[index]);

double area = moment.m00;

if(area>400)

{ Color object;

object.setType(colored.getType());

object.setColor(colored.getColor());

colors.push\_back(object);

objectFound = true;

}else objectFound = false;

}

if(objectFound ==true) {drawlines(colors,camerafeed,temp,contours,hierarchy);}

}else

putText(camerafeed,"TOO MUCH NOISE, ADJUST BRIGHTNESS",Point(0,50),1,1,Scalar(0,0,255),2);

}

}

void Executor(Mat camera1, Mat camera2, Mat frame1, Mat frame2, Color colour)

{

cvtColor(camera1,frame1,COLOR\_BGR2HSV);

inRange(frame1, colour.getHSVmin(),colour.getHSVmax(), frame2);

Noise\_Reductor(frame2);

Circle\_Detector(colour,frame2,camera1);

Circle\_Detector(colour,frame2,camera2); Circle\_Tracker(colour,frame2,camera2);

}

int main()

{

VideoCapture webcam(0);

if(!webcam.isOpened())

{

cout << "ERROR 69 : Cannot open webcam.3gp" << endl;

return -1;

}

int frameCounter = 0;

int tick = 0;

int fps;

time\_t timeBegin=time(0);

waitKey(1000);

while(1)

{

frameCounter++;

time\_t timeNow=time(0)-timeBegin;

if (timeNow - tick >= 1)

{

tick++;

fps = frameCounter;

frameCounter = 0;

}

bool Zucc = webcam.read(camera\_feed);

if(!Zucc)

{

cout << "Laptop anda kentang ya?" << endl;

break;

}

webcam.read(filter\_feed);

rectangle(camera\_feed,Point(0,20),Point(900,0),Scalar(0,0,0),-1,8); rectangle(filter\_feed,Point(0,900),Point(900,0),Scalar(0,0,0),-1,8);

rectangle(camera\_feed,Point(0,20),Point(900,0),Scalar(255,255,255),1,8);

rectangle(filter\_feed,Point(0,20),Point(900,0),Scalar(255,255,255),1,8);

putText(camera\_feed,format("FPS : %d", fps ),Point(5,15),5,0.5,Scalar(255,255,255),1);

putText(filter\_feed,format("FPS : %d", fps ),Point(5,15),5,0.5,Scalar(255,255,255),1);

Color red("red"), green("green"), blue("blue");

Executor(camera\_feed, filter\_feed, HSV\_filter, treshold, red);

Executor(camera\_feed, filter\_feed, HSV\_filter, treshold, green);

Executor(camera\_feed, filter\_feed, HSV\_filter, treshold, blue);

imshow("Filtered Feed", filter\_feed);

imshow("Actual Feed", camera\_feed);

if (waitKey(30) == 27) ///wait for 'esc' key press for 30ms. If 'esc' key is pressed, end loop

{

cout << "Exit .w." << endl;

break;

}

}

return 0;

}