CENTRO UNIVERSITARIO DE CIENCIAS EXÁCTAS E INGENIERÍAS

# Detector de Color

## Practica 02

Aldo Alexandro Vargas Meza

15/10/2017

## RESUMEN

Para esta práctica se utilizarán 2 elementos externos a la interfaz de QT. Una cámara IP y un Arduino para comunicación serial.

El funcionamiento de la práctica es el de un detector de color. Cuando la cámara detecte un objeto de color rojo, por medio de una comunicación serial con Arduino encenderá el LED correspondiente.

Cada detección de color, llevará a cabo un registro de evento, con fecha y hora, así como el color detectado. El registro se llevará en un archivo csv. Este archivo podrá ser enviado a un correo electrónico dado.

## INTRODUCCIÓN

Al principio de los archivos fuente, se declaran las librerías necesarias para el funcionamiento de todos los aspectos de la práctica. También declaran una serie de variables y banderas de control. También se declara un objeto del tipo color Detector, su funcionamiento se describe en dos archivos que conforman una librería.

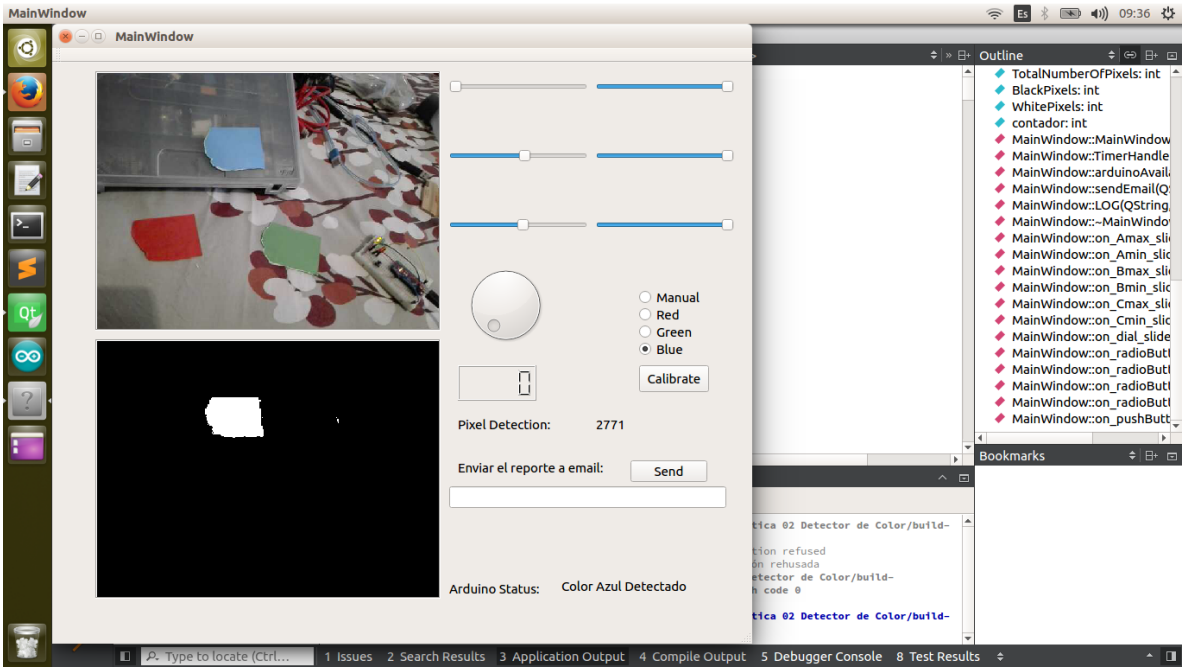
```
class colorDetector{
public:
   QDate fecha;
   QTime hora;
   int sizex = 400;
   int sizey = 300;

   int SliderParameter[6];
   int contador;

   int TotalNumberOfPixels = sizex * sizey;
   int BlackPixels, WhitePixels;

   Mat IMAGEN, IMAGENSmall;
   Mat GAUSS, HSV, COLOR;

   QString tiempoLocal, evento, color;
   QString nombreArchivo;

   QSerialPort *arduino;
   QString arduino_port_name;


   quint16 arduino_uno_vendor_id;
   quint16 arduino_uno_product_id;
   bool arduino_is_available;

   bool logg;
   bool redDet;
   bool greenDet;
   bool blueDet;

public:

   colorDetector();
   void sendEmail(QString d);
   void logData(QString event, QString col);
   void detect(int color[], QString led);
   void arduinoAvailable();
};
```

color Detector está definido como una clase, con variables distintas para datos como enteros y strings, así como fechas, horas, banderas, conexiones y matrices para imágenes, entro otros. También contiene una serie de funciones para detectar el color, enviar correo, guardar registros y conectar el Arduino.

La definición de la clase se encuentra en otro archivo, el cual implementa las funciones, siendo la más importante la de detección.

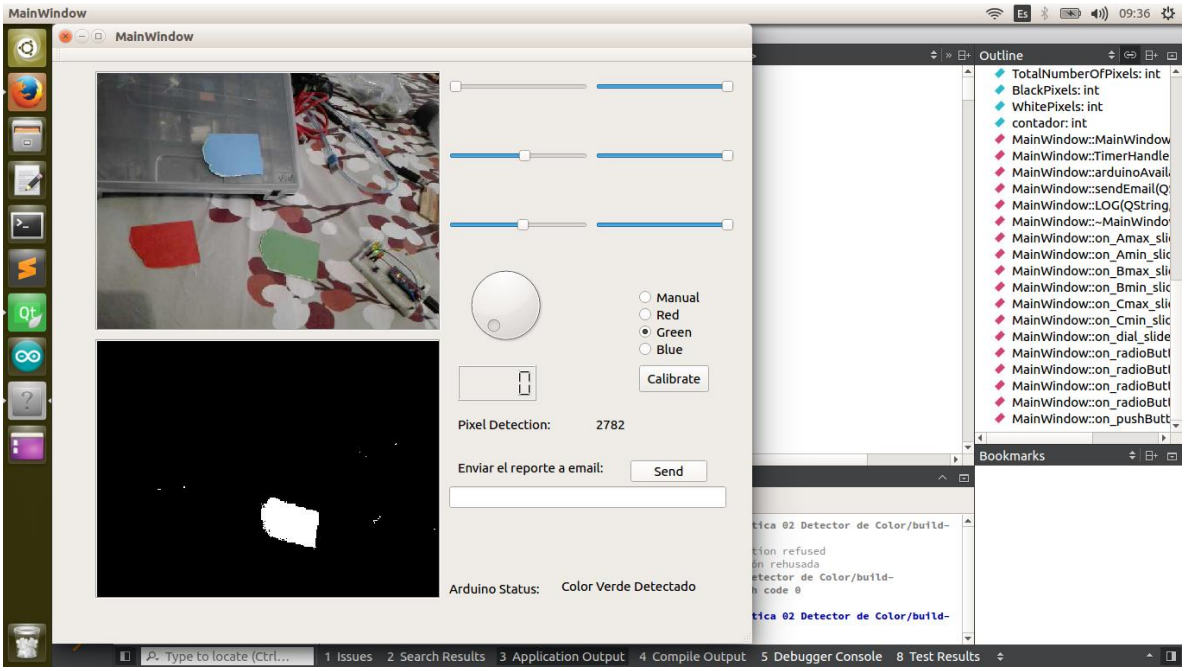
La función de detección calcula al iniciar los pixeles blancos de la imagen, tras ser tratada por los filtros, y ser alojada en una matriz. Dependiendo de si el límite de pixeles blancos es alcanzado, se ejecuta la detección. Si está por debajo de un valor especificado, las banderas de control se desactivan. Si es valor es superado, se encienden los LEDS en el Arduino y se genera un registro.

La función toma como parámetros el color a detectar, y la configuración del color, en un arreglo de enteros.




Iniciando el Widget de la aplicación, se crean dos timer ejecutándose cada 30 ms, y se abre la cámara IP.

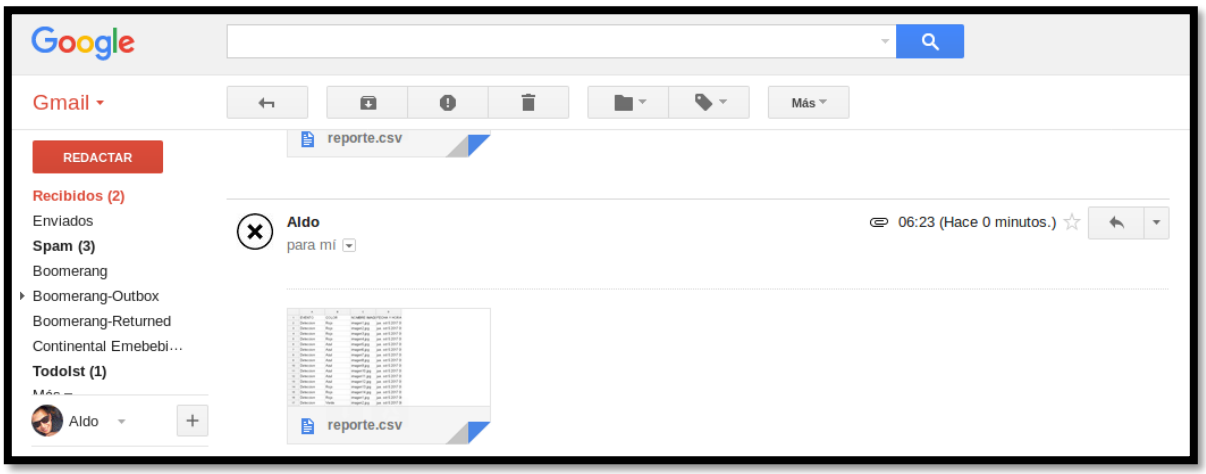
La función general repetida, se encarga de ajustar los valores para las calibraciones, así como las banderas de las mismas. Una vez calibrados los 3 colores se puede acceder a la configuración automática.



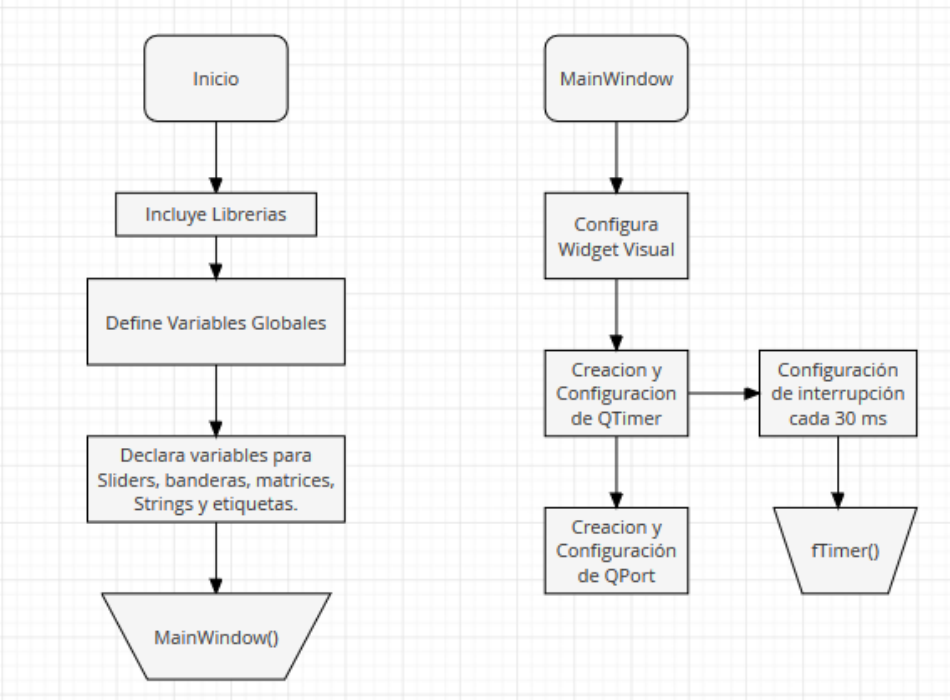

La segunda función, ejecuta por turnos la detección de color, cada 30 segundos cambiando entre uno y otro.

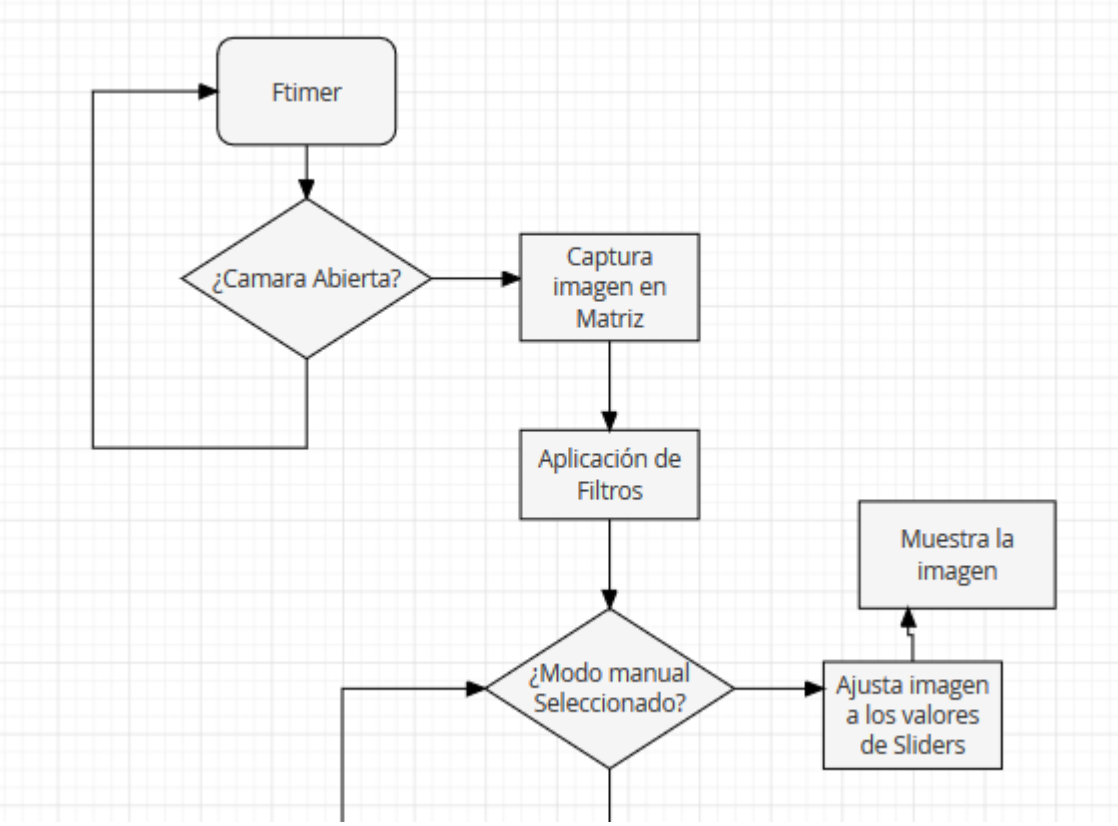
Además, se encuentran funciones para Sliders y el filtro, los cuales son aplicados en tiempo real.

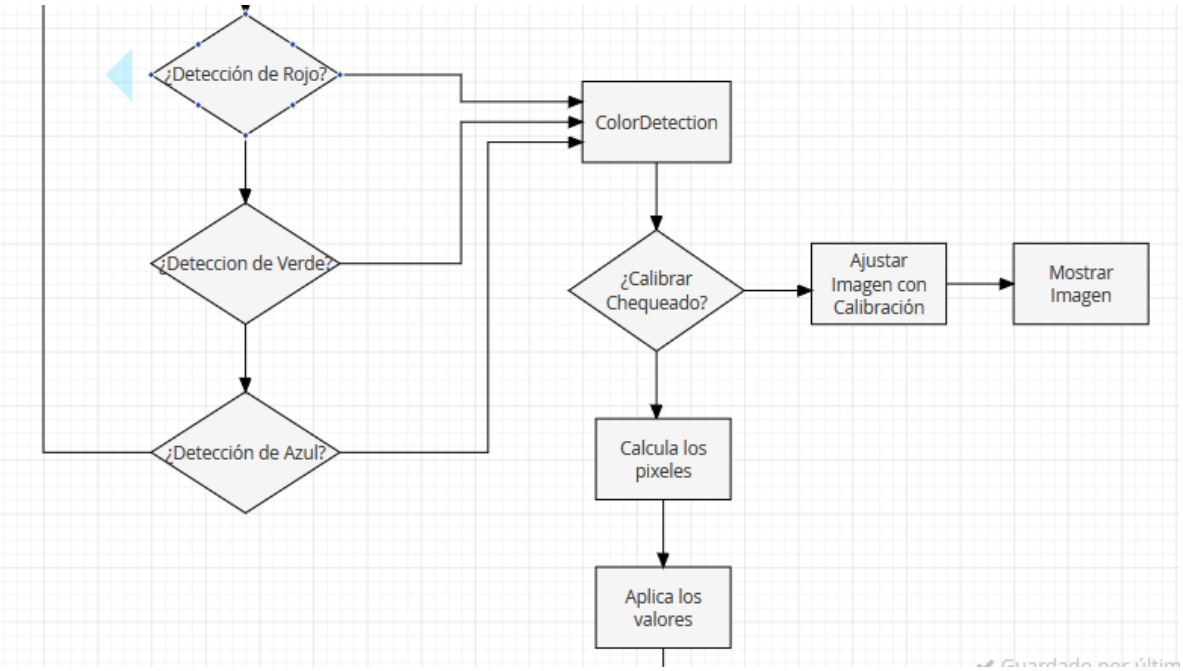
La interfaz consiste en las dos imágenes, la original y la filtrada. También hay 6 sliders para valores en los filtros, y uno para filtro gaussiano, además de un botón para ingresar una dirección de correo electrónico y enviarlo.

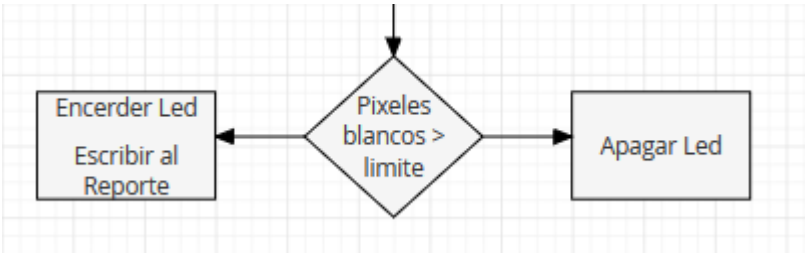


## Diagrama de Flujo






## Códigos

MainWindows.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#define IP "http://192.168.0.5:8080/video"
using namespace cv;

/*  Variables Globales
 *
 */
VideoCapture cam_01(IP);
/* Sliders para Deteccion */
int A1 = 0;
int A2 = 0;
int B1 = 0;
int B2 = 0;
int C1 = 0;
int C2 = 0;
/* Banderas de Estados */
bool MANUAL = false;
```

```
bool RED    = false;
bool GREEN  = false;
bool BLUE   = false;
/* Deteccion RGB */
bool startDetection = false;
int colordetection = 0;
/* Banderas de calibracion */
bool R_calib = false;
bool G_calib = false;
bool B_calib = false;
/* Arreglos configuracion */
int blue[6]     = {};
int green[6]    = {};
int red[6]      = {};
/* Filtro Gaussiano */
int vnt = 1;
/* Objeto detector de colores */
colorDetector PR2;

/*  MainWindows
 *     - Establece configuracion inicial
 *     - Creacion y configuracion de 2 interrupciones por Timer
 *     - Apertura de Camara IP
 */
MainWindow::MainWindow(QWidget* parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);

   /* Timer General */
   QTimer* cronometro = new QTimer(this);
   connect(cronometro, SIGNAL(timeout()), this, SLOT(TimerGeneral()));
   cronometro->start(30);

   /* Timer Lecturas */
   QTimer* cronometro2 = new QTimer(this);
   connect(cronometro2, SIGNAL(timeout()), this, SLOT(TimerReadings()));
   cronometro2->start(30);
   /* Abre camara IP */
   if (!cam_01.isOpened()) {
      cam_01.open(IP);
   }
}
/*  Timer Readings
 *     - Detecta cada 30 ms un color diferente
 *       previamente establecido
 *     -
 */
void MainWindow::TimerReadings()
{
   /* Deteccion de Colores */
```

```
    if (startDetection) {
        switch (colordetection) {
        case 0:
            PR2.detect(red, "Rojo");
            break;
        case 1:
            PR2.detect(green, "Verde");
            break;
        case 2:
            PR2.detect(blue, "Azul");
            break;
        }

        colordetection++;
        if (colordetection > 2) colordetection = 0;
    }
    /* Actualizacion de Etiquetas */
    ui->label_3->setText(QString::number(PR2.WhitePixels));
}
/*  Timer General
 *     - Detecta cada 30 ms un color diferente
 *       previamente establecido
 *     -
 */
void MainWindow::TimerGeneral()
{
    if (cam_01.isOpened()) {
        /* Guarda imagen en matrices */
        cam_01 >> PR2.IMAGEN;
        cv::resize(PR2.IMAGEN, PR2.IMAGENSmall, Size(PR2.sizex, PR2.sizey), 0, 0,
            0);
        cv::GaussianBlur(PR2.IMAGENSmall, PR2.GAUSS, Size(vnt, vnt), 0, 0, 0);
        cv::cvtColor(PR2.GAUSS, PR2.HSV, CV_BGR2HSV);
        /* Colores Calibrados */
        if (R_calib && G_calib && B_calib) {

            if (MANUAL) {

                if (ui->pushButton->isChecked()) {

                    R_calib = false;
                    G_calib = false;
                    B_calib = false;
                } else {

                    startDetection = true;
                }

            } else {
                cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2), PR2.COLOR);
            }
```

```
} else {
   /* Colores no Calibrados */
   startDetection = false;
   if (MANUAL) {
      cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2), PR2.COLOR);
      ui->pushButton->setVisible(false);

   }

   else if (RED) {
      /* Color Seleccionado */
      ui->pushButton->setVisible(true);
      /* Si el boton está presionado */
      if (ui->pushButton->isChecked()) {
         /* Guarda los valores en arreglo */
         cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2),
            PR2.COLOR);

         fillColorArray(&red[0], &R_calib);

      } else {

         if (R_calib) {
            cv::inRange(PR2.HSV, Scalar(red[0], red[2], red[4]),
               Scalar(red[1], red[3], red[5]), PR2.COLOR);
         } else {
            cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2),
               PR2.COLOR);
         }
      }
   }

   else if (GREEN) {
      ui->pushButton->setVisible(true);
      if (ui->pushButton->isChecked()) {

         cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2),
            PR2.COLOR);
         fillColorArray(&green[0], &G_calib);

      } else {
         if (G_calib) {
            cv::inRange(PR2.HSV, Scalar(green[0], green[2], green[4]),
               Scalar(green[1], green[3], green[5]), PR2.COLOR);
         } else {
            cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2),
               PR2.COLOR);
         }
      }
   }
```

```
        else if (BLUE) {
            ui->pushButton->setVisible(true);

            if (ui->pushButton->isChecked()) {

                cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2),
                    PR2.COLOR);

                fillColorArray(&blue[0], &B_calib);

            }

            else {
                if (B_calib) {
                    cv::inRange(PR2.HSV, Scalar(blue[0], blue[2], blue[4]),
                        Scalar(blue[1], blue[3], blue[5]), PR2.COLOR);
                } else {
                    cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2),
                        PR2.COLOR);
                }
            }

        }

        else {
            cv::inRange(PR2.HSV, Scalar(A1, B1, C1), Scalar(A2, B2, C2), PR2.COLOR);
        }
    }

    /* Conexion a Arduino*/
    PR2.arduinoAvailable();
    /* Muestra imagenes en label*/
    QImage qImage = Mat2QImage(PR2.GAUSS);
    QPixmap pixmap = QPixmap::fromImage(qImage);

    ui->label->clear();
    ui->label->setPixmap(pixmap);

    qImage = Mat2QImage(PR2.COLOR);
    pixmap = QPixmap::fromImage(qImage);

    ui->label_2->clear();
    ui->label_2->setPixmap(pixmap);
  }
}

MainWindow::~MainWindow()
{
  delete ui;
}
/*  Sliders
```

```
 *
 *
 */
void MainWindow::on_Amax_sliderMoved(int position)
{
   A2 = position;
}

void MainWindow::on_Amin_sliderMoved(int position)
{
   A1 = position;
}

void MainWindow::on_Bmax_sliderMoved(int position)
{
   B2 = position;
}

void MainWindow::on_Bmin_sliderMoved(int position)
{
   B1 = position;
}

void MainWindow::on_Cmax_sliderMoved(int position)
{
   C2 = position;
}

void MainWindow::on_Cmin_sliderMoved(int position)
{
   C1 = position;
}

void MainWindow::on_dial_sliderMoved(int position)
{
   switch (position) {
   case 0:
      vnt = 1;
      break;
   case 1:
      vnt = 7;
      break;
   case 2:
      vnt = 13;
      break;
   case 3:
      vnt = 19;
      break;
   case 4:
      vnt = 25;
      break;
   case 5:
```

```
        vnt = 31;
        break;
    case 6:
        vnt = 37;
        break;
    case 7:
        vnt = 43;
        break;
    case 8:
        vnt = 49;
        break;
    case 9:
        vnt = 55;
        break;
    default:
        vnt = 61;
        break;
    }

    ui->lcdNumber->display(position);
}

/*  Botones Radiales
 *
 *
 */
void MainWindow::on_radioButton_clicked()
{
    MANUAL = true;
    RED = false;
    GREEN = false;
    BLUE = false;
}

void MainWindow::on_radioButton_2_clicked()
{
    MANUAL = false;
    RED = true;
    GREEN = false;
    BLUE = false;
}

void MainWindow::on_radioButton_3_clicked()
{
    MANUAL = false;
    RED = false;
    GREEN = true;
    BLUE = false;
}

void MainWindow::on_radioButton_4_clicked()
{
```

```
    MANUAL = false;
    RED = false;
    GREEN = false;
    BLUE = true;
}
/*  sendEmail
 *
 *
 */

void MainWindow::on_pushButton_2_clicked()
{
    PR2.sendEmail(ui->lineEdit->text());
}

/*  fillColorArray
 *    - Llena el arreglo de color con valores de Sliders
 *
 */
void MainWindow::fillColorArray(int* array, bool* flag)
{
    array[0] = A1;
    array[1] = A2;
    array[2] = B1;
    array[3] = B2;
    array[4] = C1;
    array[5] = C2;

    *flag = true;
}
```

MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

/*  Librerias
 *
 *
 */

#include <QSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
#include <QDesktopWidget>
#include <QScreen>
#include <QMessageBox>
#include <QMetaEnum>
#include <QTimer>

#include<mat2qimage.h>
#include<opencv2/core/core.hpp>
#include<opencv2/ml/ml.hpp>
```

```
#include<opencv/cv.h>
#include<opencv2/imgproc/imgproc.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/video/background_segm.hpp>
#include<opencv2/videoio.hpp>
#include<opencv2/imgcodecs.hpp>

#include <QDateTime>
#include <QTime>
#include <QDate>
#include <QFile>
#include <QFileInfo>

#include <QMainWindow>

/*  Librería personalizada
 *
 *
 */
#include "practica_02.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   explicit MainWindow(QWidget *parent = 0);
   ~MainWindow();

   void fillColorArray(int *array, bool *flag);

public slots:
   void TimerReadings();
   void TimerGeneral();

private slots:
   void on_Amin_sliderMoved(int position);
   void on_Amax_sliderMoved(int position);
   void on_Bmax_sliderMoved(int position);
   void on_Bmin_sliderMoved(int position);
   void on_Cmax_sliderMoved(int position);
   void on_Cmin_sliderMoved(int position);
   void on_dial_sliderMoved(int position);

   void on_radioButton_clicked();

   void on_radioButton_2_clicked();
```

```
    void on_radioButton_3_clicked();

    void on_radioButton_4_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::MainWindow *ui;
    QSerialPort *arduino;
    QString arduino_port_name;

    static const quint16 arduino_uno_vendor_id = 0x1a86;
    static const quint16 arduino_uno_product_id = 0x7523;
    bool arduino_is_available;
};

#endif // MAINWINDOW_H
```

practica2.h

```
#ifndef PRACTICA_02_H
#define PRACTICA_02_H

#include "mainwindow.h"

#define LIM_DETECT 1500

using namespace cv;

class colorDetector{
public:
    QDate fecha;
    QTime hora;
    int sizex = 400;
    int sizey = 300;

    int SliderParameter[6];
    int contador;

    int TotalNumberOfPixels = sizex * sizey;
    int BlackPixels, WhitePixels;

    Mat IMAGEN, IMAGENSmall;
    Mat GAUSS, HSV, COLOR;

    QString tiempoLocal, evento, color;
    QString nombreArchivo;

    QSerialPort *arduino;
    QString arduino_port_name;
```

```
    quint16 arduino_uno_vendor_id;
    quint16 arduino_uno_product_id;
    bool arduino_is_available;

    bool logg;
    bool redDet;
    bool greenDet;
    bool blueDet;

public:

    colorDetector();
    void sendEmail(QString d);
    void logData(QString event, QString col);
    void detect(int color[], QString led);
    void arduinoAvailable();
};

#endif // PRACTICA_02_H
```

practica2.cpp

```
#include "practica_02.h"
/*  Constructor
 *
 *
 */
colorDetector::colorDetector() {
  arduino_is_available = false;
  arduino_port_name = "";
  arduino = new QSerialPort;

  nombreArchivo = "detectionReport.csv";
  arduino_uno_vendor_id = 0x1a86;
  arduino_uno_product_id = 0x7523;
  redDet = false;
  greenDet = false;
  blueDet = false;
}

/*  sendEmail
 *    - Envia correo a la direccion ingresada
 *      con el archivo del reporte
 */

void colorDetector::sendEmail(QString d) {
  // QString destinatario = "aldo.vargas.meza94@gmail.com";
  QString destinatario = d;
  QString command = "mpack -s subject reporte.csv " + destinatario;
```

```
  int arr = destinatario.indexOf("@");
  int com = destinatario.indexOf(".com");
  int udg = destinatario.indexOf(".udg");

  if ((arr >= 0 && com >= 0) || (arr >= 0 && udg >= 0)) {
    system(command.toUtf8().constData());
  } else {

    qDebug() << "Error con el email.";
  }
}

/*  logData
 *    - Gaurda registro de Eventos de deteccion
 *
 */

void colorDetector::logData(QString event, QString col) {
  QFile reporte(nombreArchivo);
  QFileInfo existe(nombreArchivo);

  color = col;
  evento = event;

  fecha = QDate::currentDate();
  hora = QTime::currentTime();

  tiempoLocal = fecha.toString() + " " + hora.toString();

  if (!existe.exists()) {
    reporte.open(QIODevice::WriteOnly);
    QTextStream out(&reporte);
    out << "EVENTO"
        << ",";
    out << "COLOR"
        << ",";
    out << "NOMBRE IMAGEN"
        << ",";
    out << "FECHA Y HORA \n";
    reporte.close();
  }

  if (existe.isFile() && existe.exists()) {
      QString nombreImagen = "detection_0" + QString::number(contador) + ".jpg";
      cv::imwrite(nombreImagen.toUtf8().constData(), COLOR);
    if (reporte.open(QIODevice::WriteOnly | QIODevice::Append)) {
      QTextStream out(&reporte);
      out << evento.toUtf8().constData() << ",";
      out << color.toUtf8().constData() << ",";
      out << "detection_0" << contador << ".jpg"<< ",";
      out << tiempoLocal.toUtf8().constData() << "\n";
```

```
    reporte.close();

   }

 }

 contador++;
}

/*  detect
 *    Detecta el color
 *    param: Arreglo de valores, Color
 */
void colorDetector::detect(int color[], QString led) {
 cv::inRange(HSV, Scalar(color[0], color[2], color[4]),
          Scalar(color[1], color[3], color[5]), COLOR);

 BlackPixels = TotalNumberOfPixels - countNonZero(COLOR);
 WhitePixels = TotalNumberOfPixels - BlackPixels;

 if (WhitePixels > LIM_DETECT) {
  if (arduino_is_available) {
   if (arduino->isWritable()) {
    if (led == "Rojo") {
     arduino->write("5"); // off Azul
     arduino->write("3"); // off Verde

     arduino->write("0"); // on rojo

     if (logg == false && redDet == false) {
      logData("DETECCION", led);
      logg = true;
      redDet = true;
      qDebug() << "Deteccion " << led;
     }
     else{

     }

    } else if (led == "Verde") {
     arduino->write("5"); // off Azul
     arduino->write("1"); // off Rojo

     arduino->write("2"); // on Verde

     if (logg == false && greenDet == false) {
      logData("DETECCION", led);
      logg = true;
      greenDet = true;
      qDebug() << "Deteccion " << led;
     }
```

```
        else{

        }

       } else if (led == "Azul") {
        arduino->write("3"); // off Verde
        arduino->write("1"); // off Rojo

        arduino->write("4"); // on Azul

        if (logg == false && blueDet == false) {
         logData("DETECCION", led);
         logg = true;
         blueDet = true;
         qDebug() << "Deteccion " << led;
        }
        else{

        }
       }
      }

    } else {
     qDebug() << "Arduino NO ESCRIBIBLE";
    }
   }

   else if (WhitePixels < 100) {
    if (led == "Rojo") {
     redDet = false;
    } else if (led == "Verde") {
     greenDet = false;
    } else if (led == "Azul") {
     blueDet = false;
    }

    logg = false;
    arduino->write("3"); // off Verde
    arduino->write("1"); // off Rojo
    arduino->write("5"); // off Azul
    qDebug() << "Deteccion en curso";
   }
  }

  void colorDetector::arduinoAvailable() {
   if (!arduino_is_available) {
    qDebug() << "Arduino no Conectado";

    foreach (const QSerialPortInfo &serialPortInfo,
          QSerialPortInfo::availablePorts()) {
     qDebug() << "Puerto Usb Conectado";
```

```
    if (serialPortInfo.hasVendorIdentifier() &&
        serialPortInfo.hasProductIdentifier()) {
      if (serialPortInfo.vendorIdentifier() == arduino_uno_vendor_id) {
        if (serialPortInfo.productIdentifier() == arduino_uno_product_id) {
          arduino_port_name = serialPortInfo.portName();
          arduino->setPortName(arduino_port_name);
          arduino->open(QIODevice::ReadWrite);
          arduino->setDataBits(QSerialPort::Data8);
          arduino->setBaudRate(QSerialPort::Baud9600);
          arduino->setParity(QSerialPort::NoParity);
          arduino->setStopBits(QSerialPort::OneStop);
          arduino->setFlowControl(QSerialPort::NoFlowControl);
          arduino_is_available = true;

          qDebug() << "Arduino Conectado";
        } else {
          qDebug() << "El Arduino no corresponde";
        }
      } else {
        qDebug() << "Dispositivo no es Arduino";
      }
    } else {

      qDebug() << "Dispositivo sin identificador";
    }
  }
 } else {
 }
}
```