ROCO318 - Mobile and Humanoid Robots
Report - Vision

Alfred Ingi Wilmot[1]

I. Introduction

This report is a summary of the work completed by Emily-Jane Rolley-Parnell and Alfred Ingi Wilmot as part of the module ROCO318 to create a program that, using a standard RGB camera, can detect and track a shape of any colour. This work was completed as a group. We used the functionality provided by the Open CV libraries in the programming language of C++. In order to optimise the code we also used Object-Oriented Programming (OOP) through classes and separated `cpp` and `hpp` files, for each section of functionality.

The OOP paradigm offers the benefits of encapsulation, inheritance, and code safety– i.e. the user and other classes can only access what the API allows them to, according to attribute member-function scope declarations, and only in the manner imposed by the API via access methods.

The general idea was that each object would essentially be an image-processing element that would be passed a pointer to a matrix that was to be filtered/ analyzed. These objects could then simply be appended to one another in order to yield more complex and interesting behaviour. This approach should also scale nicely, provided that inheritance can be performed in a seamless fashion. Some of the objects made were simple, such as Gaussian blur with track-bar, but others ended up being rather complex such as the contour-fitting object– this can be dealt with by organizing a convoluted class such as this into more discrete parts.

OOP also allows for straightforward object classification, as each object that is to be classified in an image can have an OOP class associated with it, and then multiple instances of that class can be stored in vectors of that class type.

II. Functionality

Section II describes each stage of the process in order to track and display the position of a rectangular piece of card within a camera field of view.

A. Gaussian Blur

To improve robustness, a Gaussian Blur is applied to the input image. Gaussian filtering is done by convolving each point in the input array with a Gaussian kernel and then summing them all to produce the output array [1]. The kernel size varies throughout the application from between a 2x2 pixel matrix to a 50x50 pixel matrix. The optimum kernel size is one that is large enough to average the HSV of all

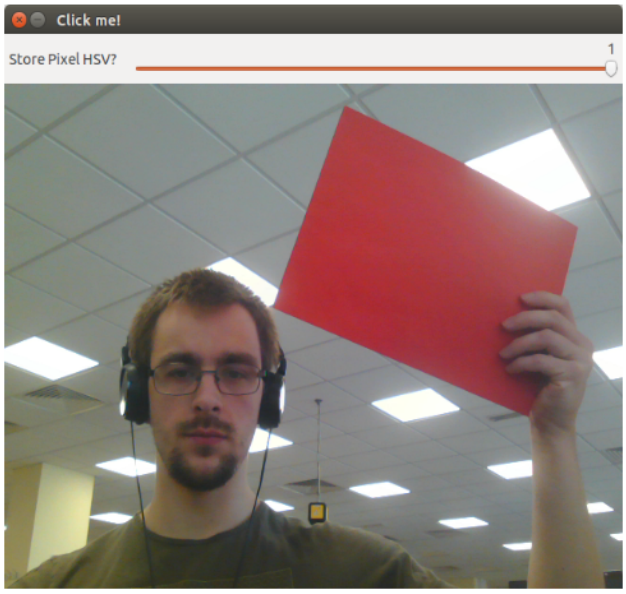

Fig. 1: Unblurred input RGB image.

of the pixels within the Region Of Interest (ROI), in this case, the tracked rectangle. The kernel size must also be small enough that it does not include the HSV of the pixels that are outside of this region. Therefore the kernel size varies based on pixel width of the tracked object, which for a fixed size object, varies with distance. In order to select the best kernel size, we have implemented a slider mechanism so that the kernel size can be changed at run time. A minor drawback of Gaussian Blurring is that, as the kernel size increases, the more mathematical calculations are performed. Therefore, with a larger kernel, the processing time for each frame received is much longer. In this specific application, the effect is minor and does not impact much on the performance. Another option for blurring was Bilateral Filtering[2], which applies blurring while maintaining crisp edges between colours. We chose not to use this method, as it is more computationally expensive.

```
void GaussianBlur(InputArray src, OutputArray dst,
Size ksize, double sigmaX, double sigmaY=0,
int borderType=BORDER_DEFAULT );
```

[3]

B. Colour Rejection

In order to optimise the reliability of tracking an object, colour is considered as well. More specifically, the range of HSV values are restricted to only include the colour of the

[1]University of Plymouth, Drake Circus, Plymouth, Devon PL4 8AA, United Kingdom. {alfred.wilmot}@students.plymouth.ac.uk

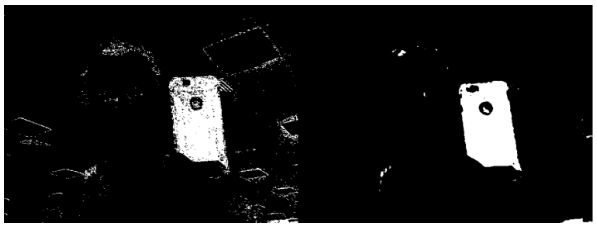

Fig. 2: Effect of Gaussian Blurring on a binarised image. The amount of peripheral noise is reduced.

object you wish to track. HSV are 3 values for a colour, Hue, Saturation, and Value.

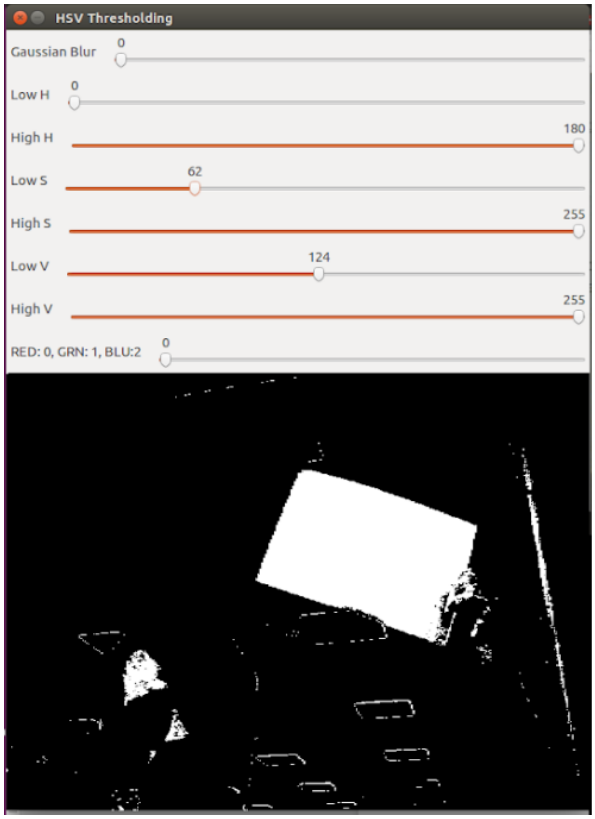

Fig. 3: HSV Thresholded image with the slider ranges for Hue, Saturation and Value. For Red colours, the range of H is between 180-¿0 in that direction as it is at a point on the colour spectrum that is only at either extreme.

In our case, the colour calibration requires that a seed pixel is picked. A seed pixel is a starting pixel that is clicked on by the user that gives the RGB value of the desired colour object. To begin colour tracking, first the RGB pixel must be converted to HSV with the command:

```
cv::cvtColor(RGB, HSV,CV_BGR2HSV);
```

[4] Where RGB is the Vec3b for the RGB input pixel and HSV is the corresponding Vec3b of the output pixel.

C. Masking

Masking is the operation of applying a kernel to a matrix, in this case the matrix is an array of pixels and the kernel is a matrix of 1s. A bitwise OR is applied between the kernel and the image frame to produce a frame containing only the pixels under the kernel mask. For the first cycle of tracking, a mask expands from the selected initial seed pixel up to the contour boundary input frame. For subsequent frames, the mask of the previous frame is used as specified below. The mask of frame f_{t-1} is slightly enlarged so that there is a more noticeable difference between frames for better object tracking. The enlarged mask is then convolved with the HSV-thresholded input frame to remove irrelevant distant noise from the ROI. The resulting frame of this convolving operation is then morphologically opened, as described in II-D, in order to remove any noise within the mask that is within to the ROI. This ”final masked and processed image” is used as the basis of the mask for frame f_t for the next HSV-thresholded input frame.

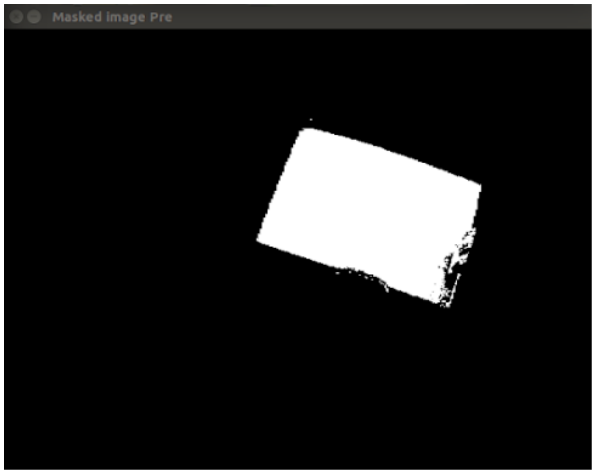

Fig. 4: Image to show the masked image, only showing the relevant area of pixel information as selected using the previous frame ROI.

D. Morphological Operations

To reduce noise in the masked image of the object to be tracked, we perform the morphological operation ”Open”.

```
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
```

[5]

This is an operation that ”erodes” the image then ”dilates” it. When the image is eroded, a kernel moves across the image and is convolved with the original image. A pixel in the original binarised image will only be considered 1 if all the pixels under the kernel are 1, otherwise the resulting pixel is made to 0. The effect of this is that boundary pixels and small white noise will be removed, so the size of the white region decreases in the image. Dilation is the opposite operation. A kernel moves across the image and is convolved, but if any pixel under the kernel is ’1’, the resulting pixel is ’1’. This operation enlarges any white space in the image. It is useful for rejoining parts of an image with broken white space.

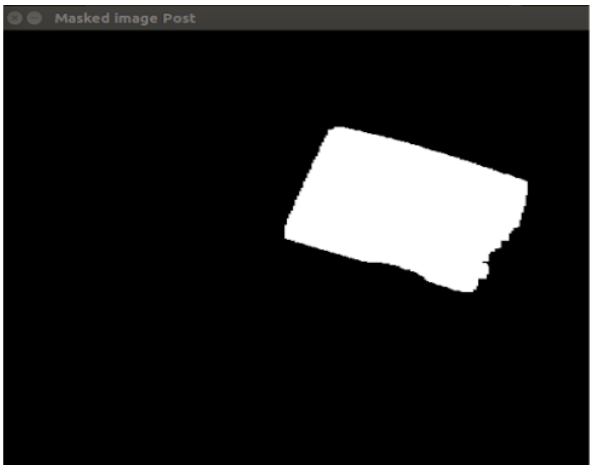

Fig. 5: Frame showing the masked image after having the morphological operation "Open" performed. When compared with Fig 4, it can be seen that although the edge of the paper is a less crisp line, there is less noise as the pixel clusters are more "blocky".

E. Contours

Contours are described as a curve joining all continuous points, with same intensity or color. We use contours as an initial approximation of the shape we will track.

```
findContours(*this->_input_frame, this->contours, hierarchy,
RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

At this stage, contours can be found easily with "CV::findContours", as the HSV Thresholding has already been applied. An alternative method for contour detection is Canny Edge Detection, which processes a grey scale image. In our case, the input matrix is already binarised, therefore findContours is the simpler method. We use the External retrieval method for finding contours [6] which retrieves only the extreme outer contours. We also use the simple Chain Approximation method of recording the contours which compresses horizontal, vertical, and diagonal segments and leaves only the end points. For example, an up-right rectangular contour is encoded with 4 points [7]. To visualise the contour, it can be seen in Fig 6 as the thin green line near the bounding rectangle. The contour does not fit exactly to the

F. Rectangle Application

To apply a rectangle surrounding the pixels on the image frame of the tracked object, several steps must be followed. First, the relevant contour for the object must be found. The code loops though each of the closed contours and checks if the contour contains the initial seed pixel by using the following line of code:

```
contains_seed = pointPolygonTest(this->contours,
Point2f(this->_seed_x, this->_seed_y), false);
```

[8]

```
minRect = minAreaRect(this->contours);
```

[9] Assuming that the contour contains the seed, minAreaRect creates a rectangle bounding box for a given contour of type RotatedRect. This rectangle is then copied to the next input frame and visualised on a screen to show the location of the tracked object.

G. Distance Calculation

Every frame, a distance estimate is made using the Pinhole Camera Calculation [10]. In this case, we wish to find Z for the real world distance. To find this the equation is given by:

$$Z = \frac{f * D}{d}$$

Where f is the focal length of the camera, D is the real world width of the object, and d is the pixel width of the object. As the size of the object and the focal length of the camera are declared within the code when calibrating initially, the only runtime variable is d which is measured across the width of the RotatedRect variable used in Section II-F.

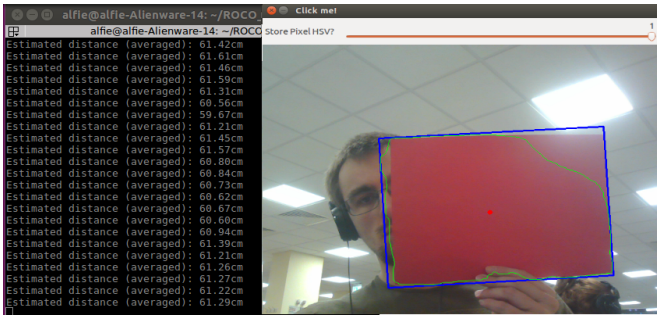

Fig. 6: Output of various distances estimated over time for an A4 piece of red card.

III. Distance Calibration

For the purposes of estimating the distance of the object from the camera, a number of variables are fixed. The focal length of the camera is fixed for each camera device, and the physical size of the object must be fixed for each object tracked. Both of these values are, in our case, hard coded.

You can use the pinhole camera projection formula [10] to calibrate the camera and find the focal length of a given camera, assuming that the position of the object is perpendicular to the axes of the camera frame. For a sheet of A4 paper, the height and width are known to be be 21cm x 29.7cm. If we hold the piece of paper 50cm away from the camera, the pixel height and width are perceived as 306 pixels x 446 pixels. With these values we can calculate the focal length of the camera in the x axis and y axis. Based on the camera projection formula, the focal length in the x axis is given by:

$$f = \frac{d * Z}{D}$$

Where the size of the paper is D (29.7 cm). Place it at a known distance Z (50cm), in front of the camera and measure its apparent width d (446 pixels) in pixels. The focal length in x and y is therefore 750.8 and 728.5 respectively. However

because these values are similar, we approximate that the focal length $f_x = f_y$ for ease of calculation of depth at run time.

IV. Object Identification

Object Identification/ Recognition/ Classification refers to the practice of generating class labels of objects contained within an input image. This is achieved by implementing a recognition algorithm that has been trained with thousands of images of the classes that the designer wants it to look for. This labelled data-set must include images of the target class object(s) in a variety of poses, lighting conditions, and environments, in order for the system to be able to reliably identify them in different image scenes. These algorithms are only as versatile as their data-sets are rich. Many classifiers are so called binary classifiers, as they are designed to distinguish one specific class of object in an image scene containing other objects they have not been trained to identify. These include the Viola-Jones object detection framework (arguably the first detection algorithm to operate in real-time, when it was used to demonstrate face-detection in 2001)[11], and the Histograms of Oriented Gradients (HOG) feature descriptor used for pedestrian detection[12].

Object detection (the main focus of the ROCO318 coursework) refers to the process of identifying the location, and quantity, of target objects in an image, whereas classification refers specifically to the classification of an image as containing some object (but not necessarily knowing it's location in the image)[13].

The features searched for in an image by a detection algorithm can then be used to classify a class of object[14]. When looking for rectangles, for instance, the detection algorithm could search for features such as perpendicular corners and parallel lines.

Object tracking refers to the practice of comparing concurrent frames from a camera feed and identifying the differences between them in order to infer target velocity. tracking tends to be more efficient than detection, as once an object is being tracked it's appearance is already well known. Tracking algorithms are susceptible to cumulative errors, but this error can be mitigated by applying a detection algorithm every so often[15].

The objective of the computer vision coursework was to Detect a simple object: a rectangular card of uniform color. It's identifiable features include it's color, and geometry. The upcoming section covering the specificity of the coursework solution goes into more detail regarding the target-detection issues imposed by unreliable lighting conditions, and variable/ dynamic target pose.

V. Specificity

Given that the solution presented in this coursework is based around HSV color-space segmentation, it will work optimally if the color of the target is sufficiently dissimilar from objects adjacent to it. A HSV-thresholded image is a barbarized image in which white regions are logic high, and black regions are logic low. Once the user has selected a seed-pixel location and the ROI mask has been established, after the first frame following this pixel selection: The object detection algorithm is immune to noise beyond the buffer surrounding the ROI, and fairly robust to noise within the buffer provided that it is not too well formed. The ROI (including the buffer) is subjected to a morphological Open.

A more effective solution would have involved the synergistic analysis of multiple features in the image-feed that are unique to the target. For example, the target is a rectangular piece of card with a known geometry (Aspect ratio, number of sides/ vertices, etc). If this information were also considered, then the system could be made more robust to the issues of noise in the HSV color-space, leading to improved object tracking/ detection. Although the target geometry would offer its own set of problems (such as pose variations in which the target isnt facing the camera and so its aspect ratio wont appear constant), by combining the known target information, the shortcomings of one detection system can be compensated by the other, and vice versa.

To emphasize the limitation of HSV image segmentation, the color Red has a Hue value at either extreme of the Hue spectrum, and so all Hue values will be included when detecting a target of this color. Filtering in this situation can only be achieved by adjusting the Saturation and Value ranges.

VI. Flow Chart Description

This section describes in detail the flowchart shown in Figure 7.

1) The input camera frame is captured, and a separate unadulterated copy is stored.

2) The copied frame is shown on the Click me! window (See Figure 1), in which a user can select a seed-pixel location (later used for target tracking).

3) The original camera frame is convoluted with a Gaussian blur kernel (whose kernel-size can be adjusted via a track-bar), and the resulting blurred frame is used as a reference for the HSV of the user-selected seed pixel from the Click me! window– this is the case as the blurred version of the selected pixel will be the average Hue, Saturation, Value (HSV) of pixels within the Gaussian-kernel. Therefore the kernel size should be large enough to get an adequate representation of average pixel HSV values within the Region of Interest (ROI), but not too large so as to include the HSV readings of pixels outside of the ROI. It is therefore recommended for the user to select an appropriate kernel-size for the blur prior to making a new pixel selection to acquire a HSV value that is more representative of the overall target.

4) The blurred image is then passed through a HSV threshold function, and presented in the HSV Thresholding window (see Figure 3).

5) Once the user has selected a pixel, the average HSV of the pixels around it within the Gaussian kernel is used to shift the anchoring position of the HSV parameter track-bars (the range of the lower and upper value of each parameter from the selected anchor HSV is hard-coded). Once a HSV value anchor has been moved to, the HSV parameter lower and upper boundaries can be adjusted manually using track-bars– enabling the system to be adjusted to fit a wider range of lighting conditions.

6) If the Store Pixel HSV? track-bar in the Click me! window is set to one, then the coordinates of any newly selected pixels will be used to grab the HSV of the corresponding pixel in the Gaussian convoluted frame. If this track-bar is set to 0, however, the stored HSV value does not change, just the coordinates of the seed are updated. This is particularly useful for manually adjusting to different colored targets on the fly, and also for selecting a new seed location whilst retaining user-modified HSV parameters.

7) The threshold image is passed to a contouring function. Only the contour enclosing the selected seed pixel, along with the corresponding enclosing minimum area rectangle and its Center of Mass (CoM), will be drawn onto the unadulterated image shown in the Click me! window. A mask is generated from this drawn rectangle, and the rectangles CoM is used as the new seed location.

8) This mask is slightly enlarged. The greater the enlargement, the higher the risk of introducing additional noise into the mask. On the other hand, a larger mask can track target movement more reliably.

9) This enlarged mask is masked over the input frame using a bit-wise-OR operation, effectively removing any far-noise away from the ROI (see Figure 4).

10) This masked input is then convoluted with a morphological "open" kernel (erosion followed by dilation) which has the effect of attenuating regions that contain any noise (i.e. white regions containing black noise are attenuated, but solid white regions are preserved). This removes any remaining blobs adjacent to the ROI within the enlarged mask area, which contain noise (see Figure 5). If there is any noise within the target, however, then the "open" operation will also attenuate it, and if blobs captured by the enlarged ROI contain little-to-no noise, then the ROI mask can mistake it as being part of the target.

11) Once the first masked input has been generated, it will be used to generated the contour for the next mask for the next input frame, and this process continues until the program is closed or a new pixel is selected by the user.

VII. Conclusion

While we completed the given task of creating a program that can estimate the distance of an object and track its location in the frame, given more time, there are a number of things we would like to improve. First we would like to experiment with the different types of edge detection such as Sobel, Canny and Laplacian. Although each of these methods use grey scale images, they may have improved accuracy when it comes to applying contours. Another thing to consider would be to rearrange the code so that both the focal length in X *and* Y axes may be considered. We did not do this due to time constraints but the resulting depth would be more accurate. To make a general improvement in accuracy for tracking, distance estimation and detection, we would like to implement some form of machine learning that can enable us to generate a program that is not specific and can be used in more general cases.

References

[1] "Opencv blurring documentation, opencv," https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html, accessed: 07-01-2019.

[2] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," vol. 98, 02 1998, pp. 839–846.

[3] "Opencv gaussian blurring api, opencv," https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#gaussianblur, accessed: 07-01-2019.

[4] "Opencv gaussian colour conversion api, opencv," https://docs.opencv.org/3.1.0/d7/d1b/group__imgproc__misc.html#ga397ae87e1288a81d2363b61574eb8cab, accessed: 07-01-2019.

[5] "Opencv morphological operations api, opencv," https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html, accessed: 08-01-2019.

[6] "Opencv contour retrieval methods api, opencv," https://docs.opencv.org/3.3.1/d3/dc0/group__imgproc__shape.html#ga819779b9857cc2f8601e6526a3a5bc71, accessed: 08-01-2019.

[7] "Opencv contour approximation methods api, opencv," https://docs.opencv.org/3.3.1/d3/dc0/group__imgproc__shape.html#ga4303f45752694956374734a03c54d5ff, accessed: 08-01-2019.

[8] "Opencv point-in-polygon test api, opencv," https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=pointpolygontest#pointpolygontest, accessed: 08-01-2019.

[9] "Opencv minarearect api, opencv," https://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga3d476a3417130ae5154aea421ca7ead9, accessed: 08-01-2019.

[10] "Pinhole camera model, wikipedia," https://en.wikipedia.org/wiki/Pinhole_camera_model, accessed: 07-01-2019.

[11] "Jones object detection framework, wikipedia," https://en.wikipedia.org/wiki/ViolaJones_object_detection_framework, accessed: 07-01-2019.

[12] "Histogram of oriented gradients, hog," https://www.learnopencv.com/histogram-of-oriented-gradients/, accessed: 07-01-2019.

[13] "Image recognition object detection, opencv," https://www.learnopencv.com/image-recognition-and-object-detection-part1/, accessed: 07-01-2019.

[14] "Object detection, wikipedia," https://en.wikipedia.org/wiki/Object_detection, accessed: 07-01-2019.

[15] "Object tracking, opencv," https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/, accessed: 07-01-2019.

APPENDIX

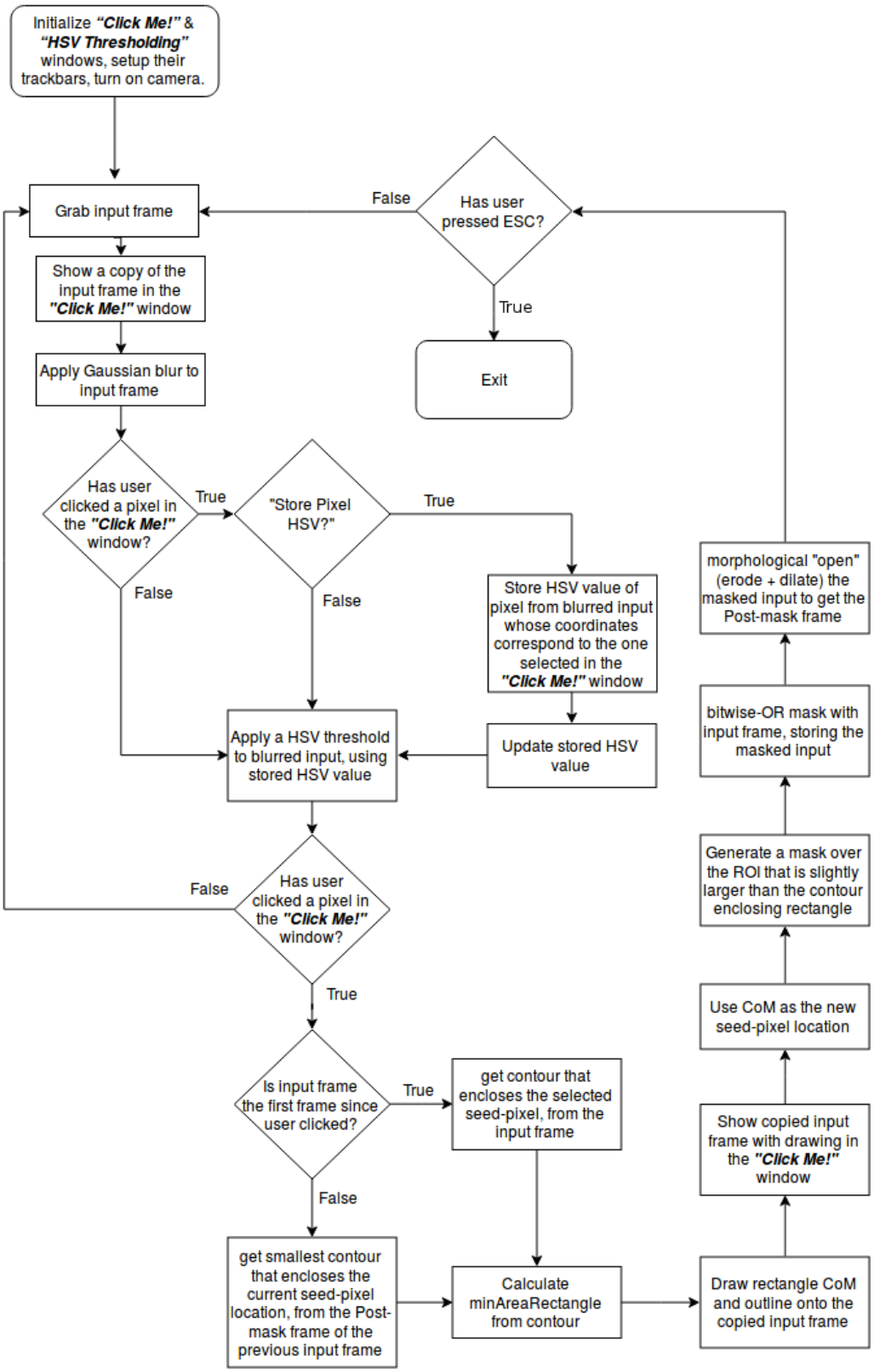

Fig. 7: Flowchart diagram to describe work flow.

VIII. Main.cpp

```
/* OpenCV & Standard libraries */
#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <iostream>

/* user-made libraries */
//#include "RectDetect1.hpp"
#include "./ModularClasses/ContourRectangles.hpp"
#include "./ModularClasses/GaussianBlurTrackbar.hpp"
#include "./ModularClasses/CannyThresholdTrackbar.hpp"
#include "./ModularClasses/HsvThresholdTrackbar.hpp"
#include "./ModularClasses/ClickForPixelData.hpp"

using namespace cv;
using namespace std;

Mat input_frame;
Mat output_frame_hsv;
Mat output_frame_canny;
Mat output_frame_contour;


char pre_process_window[32]  = "Preprocess";
const String contour_window  = "Contor Window";
const String hsv_window      = "HSV Window";
const String test_win        = "Test";

int main(int, char **)
{
    VideoCapture cap(0); // open the default camera
    if (!cap.isOpened()) // check if we succeeded and return -1 if not
    {
        return -1;
    }

    //HsvThresholdTrackbar    myHsvObj(&input_frame, &output_frame_hsv, hsv_window);

    //CannyThresholdTrackbar  myCannyObj(&output_frame_hsv, &output_frame_canny, contour_window);
    ContourRectangles       myRects(&input_frame, contour_window);

    while(1)
    {
        cap >> input_frame; //Capture image from camera.

        /* To show generated contour overlayed on the input frame */
        //output_frame_contour = input_frame.clone();

        //Step 1: Display original image.
        //imshow(pre_process_window, input_frame);

        myRects.GrabOriginalFrame(&input_frame);

        //Step 2: Apply Gaussian-blur & threshold for desired HSV value.
        //myGaussObj.gauss_blur();
        myRects.run_HSV_thresh();

        //Step 3: Apply canny thresholding to image from step 2, and place resultant rectangles ontop of original image.
        //myCannyObj.canny_thresh();
        myRects.FindRectangles();

        if (waitKey(10) == 27)
        {
            printf("%d\n\r", input_frame.channels());
            destroyAllWindows();
            break;
        }
    }

    // the camera will be deinitialized automatically in VideoCapture destructor
    return 0;




}
```

IX. ModularClasses

A. CannyThresholdTrackbar

```
#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <iostream>

using namespace cv;
using namespace std;


#include "CannyThresholdTrackbar.hpp"


/* Gathering I/O frames and display window */
CannyThresholdTrackbar::CannyThresholdTrackbar(Mat *infrm, Mat *outfrm, String glugg_nafn)
{
    this->_input_frame  = infrm;
    this->_output_frame = outfrm;
    this->window_name   = glugg_nafn;
    namedWindow(this->window_name);
    createTrackbar("Canny Threshold", this->window_name, &this->_thresh, this->_max_thresh, CannyThresholdTrackbar::onCannyTrack, this
}



/* Canny & Sobel Trackbar callbacks */
void CannyThresholdTrackbar::canny_thresh_callback(int val)
{
     this->_thresh = val;
}


/* Processing methods */
void CannyThresholdTrackbar::canny_thresh()
{

    // make sure input frame isn't empty.
    this->errorHandling();

    /* Convert an unbinarized image into grayscale */
    if(this->_input_frame->channels() > 1)
    {
        //Convert from BGR to HSV colorspace
        cvtColor( *this->_input_frame, *this->_output_frame, CV_BGR2GRAY );
    }
    else
    {
        *this->_output_frame = *this->_input_frame;
    }

    blur( *this->_output_frame, *this->_output_frame, Size(3,3) );
    Canny( *this->_output_frame, *this->_output_frame, this->_thresh, this->_thresh*2, 3);

    imshow(this->window_name, *this->_output_frame);
}



/* Redirection necessary to setup trackbars within a class */

void CannyThresholdTrackbar::onCannyTrack(int val, void* ptr)
{
    CannyThresholdTrackbar* tmp = (CannyThresholdTrackbar*)(ptr);
    tmp->canny_thresh_callback(val);
}




/* make sure input frame isn't empty. */
void CannyThresholdTrackbar::errorHandling()
{
    if(this->_input_frame->empty())
    {
        throw std::invalid_argument( "Input frame for CannyThresholdTrackbar object is empty!");
    }
}
```

```
/*-------------------------------------------------------------------------------------------*/
/*  Attaches Canny edge-detection trackbar onto a window that shows the filtered input Matrix
    (trackbar adjusts thresholding) */
/*-------------------------------------------------------------------------------------------*/
#ifndef __CANNYTHRESHOLDTRACKBAR__
#define __CANNYTHRESHOLDTRACKBAR__


#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <iostream>


class CannyThresholdTrackbar
{
private:

    cv::Mat *_input_frame;
    cv::Mat *_output_frame;
    cv::String window_name;

    int _thresh = 3, _max_thresh = 200; // Canny Threshold Trackbar parameters

    static void onCannyTrack(int val, void *ptr);
    void canny_thresh_callback(int val);

    void errorHandling();

public:
    CannyThresholdTrackbar(cv::Mat *infrm, cv::Mat *outfrm, const cv::String glugg_nafn);
    void canny_thresh();
};

#endif
```

B. ClickForPixelData

```
#include "ClickForPixelData.hpp"

#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <stdexcept>


using namespace cv;
using namespace std;


/*---- Class initializer ----*/
ClickForPixelData::ClickForPixelData(String glugg_nafn)
{
    this->click_display_window = glugg_nafn;

    namedWindow(this->click_display_window);

    /* Setup mouse-click event on input frame */
    setMouseCallback(this->click_display_window, ClickForPixelData::onMouseEvt, this);

    /* Trackbar determines if selected pixel HSV is capture or not */
    createTrackbar("Store␣Pixel␣HSV?", this->click_display_window, &this->capture_hsv, 1, ClickForPixelData::onCaptureHSV, this);
}

/*---- Grabs and displays the passed frame in window that user can click to gather HSV data ----*/
void ClickForPixelData::FrameToClick(Mat clk_frm)
{
    this-> _frm_to_clk = clk_frm.clone();
    cv::imshow(this->click_display_window, this->_frm_to_clk);
}

/*---- Grabs a frame and uses that for inspecting the HSV value at the pixel location designated by the user mouse click ----*/
void ClickForPixelData::FrameToCheck(Mat chk_frm)
{
   this-> _frm_to_check = chk_frm.clone();
}
```

```
/*---- Handler method that reacts to user selecting pixel in interactive window ----*/
int ClickForPixelData::mouseEvent(int evt, int x, int y, int flags)
{

    if (evt == CV_EVENT_LBUTTONDOWN)
    {
        this->_mouse_clk = true;      //set flag.

        /* Update the new mouse-selected seed pixel coordinates */
        this->_seed_x = x;
        this->_seed_y = y;

        /* Print pixel data to terminal */
        get_seed_pixel_hsv(false);          // Subclass will want to use _mouse_clk flag later, so don't reset it.

    }
}


/*---- Calculates and stores HSV value of pixel at input coordinate x,y ----*/
int ClickForPixelData::get_seed_pixel_hsv(bool clear_mouse_clk)
{

    // Can only update & display the stored HSV data once per mouse-click.
    if(this->_mouse_clk)
    {

        /* The selected pixel will be used to inspect the pixel at the corresponding coordinate
            in this matrix*/
        Mat tmp;

        /* If the frame to check is empty,
            then use the frame to click instead
            (assumes both are the same size)
        */
        if(this->_frm_to_check.empty())
        {
            if(this->_frm_to_clk.empty())
            {
                /* if both frames are empty, then complain */
                printf("Reference␣frame␣is␣empty!\n\r");
                this->_mouse_clk = false; //reset flag, until next mouse click.
                return -1;
            }
            else
            {
                tmp = this->_frm_to_clk;
            }
        }
        else
        {
            tmp = this->_frm_to_check;
        }




        /* decode pixel RGB values */
        Vec3b rgb = tmp.at<Vec3b>(this->_seed_x, this->_seed_y);
        int B=rgb.val[0];
        int G=rgb.val[1];
        int R=rgb.val[2];

        /* Only update the HSV value if the trackbar says so*/
        if(capture_hsv == 1)
        {
            Mat HSV;
            Mat RGB= tmp(Rect(this->_seed_x, this->_seed_y, 1, 1));   //Single-value matrix that is the pixel at point [x,y], RGB enco
            cvtColor(RGB, HSV,CV_BGR2HSV);

            Vec3b hsv=HSV.at<Vec3b>(0,0);

            this->H = hsv.val[0];
            this->S = hsv.val[1];
            this->V = hsv.val[2];
        }


        /* Display values to terminal for debugging */
        printf("[%d,␣%d]␣H:%d,␣S:%d,␣V:%d\n\r",
                this->_seed_x, this->_seed_y,
                this->H, this->S, this->V);

        /* clear the _mouse_clk flag only if indicated to do so externally */
```

```
        if(clear_mouse_clk)
        {

            /* reset flag */
            this->_mouse_clk = false;
        }
        else
        {
            /* Ensure flag is set so it can be reset later */
            this->_mouse_clk = true;
        }

        return 0;

    }

    return 1;
}



/* Redirection to get handler working */
void ClickForPixelData::onMouseEvt(int evt, int x, int y, int flags, void* ptr)
{
    ClickForPixelData* tmp = (ClickForPixelData*)(ptr);
    tmp->mouseEvent(evt, x, y, flags);
}



/* Ignore pixel HSV: just update seed pixel x,y coordinates and don't update seed HSV */
void ClickForPixelData::captureHSV(int val)
{
    this-> capture_hsv = val;
}

/* Redirection necessary to setup ignoreHSV trackbar */
void ClickForPixelData::onCaptureHSV(int val, void* ptr)
{
    ClickForPixelData* tmp = (ClickForPixelData*)(ptr);
    tmp->captureHSV(val);
}



/*-------------------------------------------------------------------------------------------*/
/*  Class object is used to access pixel information of an input matrix.
    Can be used by other classes to designate a ROI based around a selected seed-pixel. */
/*-------------------------------------------------------------------------------------------*/
#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <iostream>
#include <stdexcept>

#ifndef __CLICKFORPIXELDATA__
#define __CLICKFORPIXELDATA__


/*-----------------------------------------------------------*/
/* Infering pixel color & (x,y) coordinates at mouse click */
/*-----------------------------------------------------------*/
//reference here: http://opencvexamples.blogspot.com/2014/01/detect-mouse-clicks-and-moves-on-image.html
//Converting from RGB to HSV (reference): oh dear...

/*-----------------------------------------------------------------------------------------------------------------*/
/* Trackbar callback methods and redirection to get it working:
https://stackoverflow.com/questions/28418064/casting-c-class-into-another-one-to-create-opencv-trackbar-handle

"createTrackbar() requires address of a static function.
'this' ptr is used as an arg in the actual trackbar instantiation,
and is later resolved to the class instance."" */
/*-----------------------------------------------------------------------------------------------------------------*/

class ClickForPixelData
{
    private:


        /* Trackbar determines if pixel HSV is ignored or not */
        void captureHSV(int val);
        static void onCaptureHSV(int val, void* ptr);
```

```
    protected:

        /* Useful for subclasses to see if user wants to update seed HSV */
        int capture_hsv = 1;

        /* Interactive window*/
        cv::String click_display_window;


        /* Frame displayed onto the interactive window*/
        // THIS MUST BE ASSIGNED BEFORE PERFORMING ANY OTHER OPERATION ON INPUT IMAGE
        cv::Mat _frm_to_clk;
        cv::Mat _frm_to_check;

        /* Last known pixel HSV values */
        int H = 0;
        int S = 0;
        int V = 0;

        /* Thresholding variables */
        const int max_value_H = 180;
        const int max_value = 255;
        int low_H = 0, low_S = 0, low_V = 0;
        int high_H = this->max_value_H;
        int high_S = this->max_value, high_V = this->max_value;

        /* Indicate if next pixel selections are used to train based off an initially selected value */
        bool train_target_HSV = false;


        /* Mouse-click event stuff */
        bool _mouse_clk = false;                // Want flag to persist until all events related to a mouse click have transpired
        int _seed_x = 0;
        int _seed_y = 0;


        /* Update H, S, V variables with those corresponding to the pixel selected (if the mouse has been clicked recently).*/
        int get_seed_pixel_hsv(bool clear_mouse_clk = true);


        /* Mouse event handling methods */
        static void onMouseEvt(int evt, int x, int y, int flags, void* ptr);
        int mouseEvent(int evt, int x, int y, int flags);


    public:
        ClickForPixelData(cv::String glugg_nafn);
        /* Frame whose pixel coordinates will be used when the user clicks on it */
        void FrameToClick(cv::Mat clk_frm);
        /* The coordinates selected by user are used to check the HSV of the corresponding pixel coordinates
            in the frame passed to FrameToCheck (so frame can be processed externally to average HSV values via blurring operation
            prior to pixel HSV inspection, but the user can directly interact with an unadulterated version of the input frame).*/
        void FrameToCheck(cv::Mat chk_frm);
};


#endif
```

C. *ContourRectangles*

```
#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <stdexcept>


using namespace cv;
using namespace std;

#include "ContourRectangles.hpp"

/* Class constructor */
ContourRectangles::ContourRectangles(Mat *infrm):
HsvThresholdTrackbar(infrm, "HSV␣Thresholding")
{
    this->window_name   = this->click_display_window;

    this->masked_input  = Mat(this->_input_frame->size(), CV_8UC3);
}
```

```
/* Places an enclosing rectangle around the ROI contour, and tracks it's position by following the contour's CoM... */
void ContourRectangles::FindRectangles()
{

 /* Clear contours list from last frame */
    this->contours.clear();

    /* Ensure that the input/ output matricies are valid before continuing*/
    this->errorHandling();


    /*  Whenever a new pixel is selected use the vanialla input frame instead of the previous masked frame,
        & reset contour comparator */
    if(this->get_seed_pixel_hsv(true) == 0)
    {
         findContours(*this->_input_frame, this->contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
    }


    /* New mask for a new input frame */
    Mat mask(this->_input_frame->size(), CV_8UC1, Scalar(0,0,0));

    /* Only start drawing bounding boxes once the seed has been defined by the first mouse-click */
    if(this->_seed_x && this->_seed_y)
    {

        /* Contours from previously masked input guide behaviour of current frame mask */
        if(this->contours.empty())
        {
            findContours(this->masked_input, this->contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
        }

        /* Generate rotated-rectangle ROI, from contours */
        vector<RotatedRect> minRect( this->contours.size() );

        for( size_t i = 0; i< this->contours.size(); i++ )
        {
            /* Draw the bounding box, ie. the ROI, that contains the seed */
            contains_seed = pointPolygonTest(this->contours[i], Point2f(this->_seed_x, this->_seed_y), false); //checks if contour enc
            if(contains_seed == 1)
            {
                /* Calculating ROI CoM */
                mu = moments( this->contours[i], false );
                mc = Point2f( static_cast<float>(mu.m10/mu.m00) , static_cast<float>(mu.m01/mu.m00) );

                /* Drawing CoM */
                circle(this->_frm_to_clk, mc, 4, redDot, -1, 8, 0 );
                /* Update seed location to that of CoM to track ROI*/
                this->_seed_x = int(this->mc.x);
                this->_seed_y = int(this->mc.y);

                /* Defining bounding box for a given contour */
                minRect[i] = minAreaRect( this->contours[i] );

                /* rotated rectangle verticies */
                Point2f rect_points[4];
                minRect[i].points( rect_points );

                /*---------------------------*/
                /* Distance estimation code */
                /*---------------------------*/
                this->distance_estimate = calibration_card_width * focal_length / minRect[i].size.width;

                if(this->avg_count < this->count_limit)
                {
                    this->avg_count++;

                    this->avg_distance += this->distance_estimate;  // running sum.

                }
                else
                {
                    this->avg_count = 0;                            // reset the counter.

                    this->avg_distance/=this->count_limit;          // perform averaging.

                    printf("Estimated␣distance␣(averaged):␣%1.2fcm\n\r",  this->avg_distance);

                    this->avg_distance = 0.0;                       // start averaging anew, using a fresh set of values (mitigates di

                }
```

```
                /* Enlarged rotated rectangle (for simpler tracking implementation)*/
                Size2f new_size = minRect[i].size;
                new_size.height = new_size.height + 10;
                new_size.width  = new_size.width + 10;
                RotatedRect mask_rect(minRect[i].center, new_size, minRect[i].angle);

                Point vertices[4];
                Point2f larger_rect_points[4];
                mask_rect.points(larger_rect_points);


                /*  Draw the outline of the ROI RotatedRectangle onto the output image,
                    Convert the vertices into type Point to then fill the ROI for the mask.*/
                for ( int j = 0; j < 4; j++ )
                {
                    line( this->_frm_to_clk, rect_points[j], rect_points[(j+1)%4], this->ROI_box, 2);
                    vertices[j] = larger_rect_points[j];
                }

                /* Fill mask. */
                fillConvexPoly(mask, vertices, 4, this->ROI_box);

                /* Draw contours inside ROI. */
                drawContours( this->_frm_to_clk, this->contours, (int)i, cv::Scalar(0,255,0));



                /* Mask this ROI onto the Matrix at the input image address,
                    BEFORE calculating the next ROI:
                    -> generate the first instance of the mask here.
                    -> then use it to generate the mask for the next input frame, ad nauseum.
                */


                /* Perform masking of HSV thresholded image with generated mask */
                bitwise_and(*this->_input_frame, mask, this->masked_input);
                imshow("Masked␣image␣Pre", this->masked_input);

                /*  Morphological Opening is performed on pre-mask to mitigate noise around ROI
                    (ROI needs to be well defined relative to environment for this to work well) */
                this->morph(2);
                imshow("Masked␣image␣Post", this->masked_input);
            }
        }


    }

    /* Display ROI Rectangle & CoM overlay onto target on the interactive image */
    imshow(this->window_name, this->_frm_to_clk);
}

/* Perform morphological operations on the masked input frame */
void ContourRectangles::morph(int op)
{
    // Erode    = 0
    // Dilate   = 1
    // Opening  = 2
    // Closing  = 3
    // Gradient = 4

    int operation  = op;
    int morph_size = 2;
    int iterations = 2;
    Mat element = getStructuringElement( 0, Size( 2*morph_size + 1, 2*morph_size+1 ), Point( morph_size, morph_size ) );

    /// Apply the specified morphology operation
    morphologyEx(this->masked_input, this->masked_input, operation, element, Point(-1,-1), iterations);
}

/* Return an error message if the matrix is not a binary image (needed for contour-fitting) */
void ContourRectangles::errorHandling()
{
    if(_input_frame->empty())
    {
        throw std::invalid_argument( "Input␣frame␣is␣empty!");
    }
    else if(this->_input_frame->channels() != 1)
    {
        printf("input␣frame␣has␣%d␣channels.\n\r", this->_input_frame->channels());
        throw std::invalid_argument( "Input␣frame␣must␣be␣single␣channel.\n\r");
    }

    /* Need to fill the output frame before inserting contours/ rectangles, if it's empty */
    if(this->_frm_to_clk.empty())
```

```
    {
        this->_frm_to_clk = Mat::zeros( this->_input_frame->size(), CV_8UC3 );
    }
}



/*-------------------------------------------------------------------------------------------------------------------------*/
/* Places an enclosing rectangle around the ROI contour, and tracks it's position by following the contour's CoM... */
/*-------------------------------------------------------------------------------------------------------------------------*/
/*  1) First mouse click: contours are fitted to the HSV-thresholded input frame, the corresponding mask is generated.
    2) This mask is slightly enlarged (for better object tracking).
    3) The enlarged mask is then masked with the HSV-thresholded input frame (removing distant noise from ROI).
    4) The result of this masking is then morphologically opened in order to remove any noise within the mask that's adjacent to the I
    5) This "final masked and processed image" is used as the basis of the mask for the next HSV-thresholded input frame...

        5a) The contours of the previous "final masked and processed image"
            are used to generate the mask for the current HSV-thresholded input frame.
        5b) Repeat from step 2).
*/
/*-------------------------------------------------------------------------------------------------------------------------*/

#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <stdexcept>

#include "HsvThresholdTrackbar.hpp"


/* Some reference code: https://raw.githubusercontent.com/kylehounslow/opencv-tuts/master/object-tracking-tut/objectTrackingTut.cpp */
/* From here: https://www.youtube.com/watch?v=bSeFrPrqZ2A */

/*-----------------------------------------------*/
/* Removing noise from HSV-thresholded image */
/*-----------------------------------------------*/
//reference here: https://docs.opencv.org/2.4/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html


/* Generates an image of detected rectangles from an binarized input image, by using contours */
class ContourRectangles : public HsvThresholdTrackbar
{
private:

    cv::String window_name;

    /* Return an error message if the matrix is not a binary image (needed for contour-fitting) */
    void errorHandling();


    /* Contour ROI stuff */
    std::vector<std::vector<cv::Point>> contours;
    std::vector<cv::Vec4i> hierarchy;

    /* ROI Center of Mass */
    cv::Moments mu;
    cv::Point2f mc;

    /* ROI visual feedback */
    cv::Scalar const redDot = cv::Scalar(1,0,255);
    cv::Scalar const ROI_box = cv::Scalar(255,0,0);

    /* Used to determine contour that contains the seed pixel */
    int contains_seed = 0;

    /* Used for filtering noie after seed pixel has been selected */
    cv::Mat mask;
    cv::Mat masked_input;

    /* Perform morphological operations on the masked input frame */
    void morph(int op);


    /*  Simple distance estimation based off of focal length calibration:
        https://stackoverflow.com/questions/6714069/finding-distance-from-camera-to-object-of-known-size
     */
    float calibration_card_heigth       = 21;      // cm
    float calibration_card_width        = 29.7;    // cm
    int   calibration_card_pixel_height = 306;     //reference at 50cm from laptop camera:     306
    int   calibration_card_pixel_width  = 446;     //reference at 50cm from laptop camera:     446
    float calibration_distance          = 50;      // cm

    /*  Using focal-length calculation and similarity of triangles to estimate distance.
```

```
        Confounding variables:
        -> rotated rectangle (aliasing will cause edge lengths to cover fewer pixels compared to not being rotated)
        -> rotation of target perpendicular to camera viewing axis will cause perceived change in aspect-ratio that will be incorrect
    float focal_length       = float(calibration_card_pixel_width) * calibration_distance / calibration_card_width;

    float distance_estimate = 0.0;//calibration_card_width * focal_length / float(current_card_pixel_width);

    /* for implementing running avg for to filter out sporadic changes */
    float     avg_distance    = 0.0;
    int       avg_count       = 0;
    const int count_limit     = 20;  // average over predetermined number of frames.


public:

    /* Class constructor */
    ContourRectangles(cv::Mat *infrm);

    /* Fits rectangle onto target contour and does some basic tracking using a ROI. HSV must be calibrated first!*/
    void FindRectangles();
};
```

D. GaussianBlurTrackbar

```
#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>

using namespace cv;
using namespace std;

#include "GaussianBlurTrackbar.hpp"

/* Gathering I/O frames and display window */
GaussianBlurTrackbar::GaussianBlurTrackbar(Mat *infrm, String glugg_nafn)
{
    this->_input_frame  = infrm;
    this->window_name   = glugg_nafn;

    // Need to setup window so trackbars can be attached to it.
    namedWindow(this->window_name);


    createTrackbar("Gaussian_Blur", this->window_name, &this->_gauss_blur_qty, this->_max_gauss, GaussianBlurTrackbar::onGausTrack, th
}

/* Redirection necessary to setup trackbar */
void GaussianBlurTrackbar::onGausTrack(int val, void* ptr)
{
    GaussianBlurTrackbar* tmp = (GaussianBlurTrackbar*)(ptr);
    tmp->gauss_blur_callback(val);
}



/* Trackbar callback */
void GaussianBlurTrackbar::gauss_blur_callback(int val)
{
     if (val % 2 == 0)
     {
       val = val + 1;
     }
     this->_gauss_blur_qty = val;
}


/* Processing methods */
void GaussianBlurTrackbar::gauss_blur(bool show_img)
{
    GaussianBlur(*this->_input_frame, *this->_input_frame, Size(this->_gauss_blur_qty, this->_gauss_blur_qty),0,0);

    if(show_img)
    {
        imshow(this->window_name, *this->_input_frame);
    }
}
```

```
/*-------------------------------------------------------------------------------------*/
/*  Attaches Gaussian-blur trackbar onto a window that shows the blurred input Matrix
    (trackbar adjusts size of filter kernel) */
/*-------------------------------------------------------------------------------------*/
#ifndef __GAUSSIANBLURTRACKBAR__
#define __GAUSSIANBLURTRACKBAR__

#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <iostream>


class GaussianBlurTrackbar
{
    private:

        /* Input Matrix address and display window window */
        cv::Mat *_input_frame;
        cv::String window_name;

        int _gauss_blur_qty = 1, _max_gauss = 100;  // Trackbar parameters

        /*  Gaussian-blur trackbar callback method
            and corresponding static redirection method
            needed to get trackbar working from within a class.
        */
        static void onGausTrack(int val, void *ptr);
        void gauss_blur_callback(int val);

    public:

        /* Class constructor */
        GaussianBlurTrackbar(cv::Mat *infrm, const cv::String glugg_nafn);

        /*  Method called from main loop so that blurring
            (of kernel-size according to trackbar value)
            is performed on every captured camera frame.
        */
        void gauss_blur(bool show_img = true);
};


#endif
```

E. HsvThresholdTrackbar

```
#include "HsvThresholdTrackbar.hpp"

using namespace cv;
using namespace std;


/* Gathering I/O frames and display window */
HsvThresholdTrackbar::HsvThresholdTrackbar(Mat *infrm, String glugg_nafn):
GaussianBlurTrackbar(infrm, glugg_nafn),
ClickForPixelData("Click␣me!")
{
    this->_input_frame  = infrm;
    this->window_name   = glugg_nafn;

    // Need to setup window so trackbars can be attached to it.
    namedWindow(this->window_name);

    // Trackbars to set thresholds for HSV values
    createTrackbar("Low␣H",  this->window_name, &low_H,     max_value_H,    onLow_H_track,  this);
    createTrackbar("High␣H", this->window_name, &high_H,    max_value_H,    onHigh_H_track, this);
    createTrackbar("Low␣S",  this->window_name, &low_S,     max_value,      onLow_S_track,  this);
    createTrackbar("High␣S", this->window_name, &high_S,    max_value,      onHigh_S_track, this);
    createTrackbar("Low␣V",  this->window_name, &low_V,     max_value,      onLow_V_track,  this);
    createTrackbar("High␣V", this->window_name, &high_V,    max_value,      onHigh_V_track, this);

    // HSV presets
    createTrackbar("RED:␣0,␣GRN:␣1,␣BLU:2", this->window_name, &preset_count, preset_max, onPreset_track, this);

}
```

```
/*---- Use inherited get_pixel_HSV to adjust slidebars  ----*/
void HsvThresholdTrackbar::adjust_slideBars()
{
    if(this->get_seed_pixel_hsv(false) == 0 && this->capture_hsv == 1)
    {

        /* hard-coded HSV ranges around the seed-pixel HSV values */
        int H_step = 50;
        int S_step = 50;
        int V_step = 50;

        /* Use the HSV values to adjust the low/ high HSV trackbar values */
        this->low_H  = ( (this->H - H_step) > 0) ? this->H - H_step : 0;
        this->low_S  = ( (this->S - S_step) > 0) ? this->S - S_step : 0;
        this->low_V  = ( (this->V - V_step) > 0) ? this->V - V_step : 0;

        this->high_H = ( (this->H + H_step) < this->max_value_H) ? this->H + H_step : this->max_value_H;
        this->high_S = ( (this->S + S_step) < this->max_value)   ? this->S + S_step : this->max_value;
        this->high_V = ( (this->V + V_step) < this->max_value)   ? this->V + V_step : this->max_value;


        /* Consider that color RED is at either extreme of the Hue-spectrum */
        if(this->low_H == 0 || this->high_H == this->max_value_H)
        {
          this->low_H  = 0;
          this->high_H = this->max_value_H;
        }

        /* Update trackbar values */
        setTrackbarPos("Low␣H",  this->window_name, this->low_H);
        setTrackbarPos("High␣H", this->window_name, this->high_H);
        setTrackbarPos("Low␣S",  this->window_name, this->low_S);
        setTrackbarPos("High␣S", this->window_name, this->high_S);
        setTrackbarPos("Low␣V",  this->window_name, this->low_V);
        setTrackbarPos("High␣V", this->window_name, this->high_V);
    }
}

/*---- Apply HSV thresholding to input image ----*/
void HsvThresholdTrackbar::run_HSV_thresh()
{

    /*SHOULD APPLY GAUSSIAN BLUR FIRST!*/
    this->gauss_blur(false);

    /* Grab a copy of the blurred frame for getting "Averaged HSV values at selected pixel location" */
    this->FrameToCheck(*this->_input_frame);

    Mat frame_HSV;


    //Convert from BGR to HSV colorspace
    cvtColor(*this->_input_frame, frame_HSV, COLOR_BGR2HSV);


    // Detect the object based on HSV Range Values
    inRange(frame_HSV, Scalar(this->low_H, this->low_S, this->low_V), Scalar(this->high_H, this->high_S, this->high_V), *this->_input_

    imshow(this->window_name, *this->_input_frame);


    /* Update the slidebars if mouse has been recently clicked */
    adjust_slideBars();

}



/*---- Slidebar callback functions ----*/
void HsvThresholdTrackbar::low_H_callback(int val)
{
    this->low_H = min(this->high_H-1, val);
    setTrackbarPos("Low␣H", this->window_name, this->low_H);
}
void HsvThresholdTrackbar::high_H_callback(int val)
{
    this->high_H = max(val, this->low_H+1);
    setTrackbarPos("High␣H", this->window_name, this->high_H);
}
void HsvThresholdTrackbar::low_S_callback(int val)
{
    this->low_S = min(this->high_S-1, val);
    setTrackbarPos("Low␣S", this->window_name, this->low_S);
}
void HsvThresholdTrackbar::high_S_callback(int val)
```

```
{
    this->high_S = max(val, this->low_S+1);
    setTrackbarPos("High S", this->window_name, this->high_S);
}
void HsvThresholdTrackbar::low_V_callback(int val)
{
    this->low_V = min(this->high_V-1, val);
    setTrackbarPos("Low V", this->window_name, this->low_V);
}
void HsvThresholdTrackbar::high_V_callback(int val)
{
    this->high_V = max(val, this->low_V+1);
    setTrackbarPos("High V", this->window_name, this->high_V);
}



/*---- Redirections necessary to setup Thresholding trackbars ----*/
void HsvThresholdTrackbar::onLow_H_track(int val, void* ptr)
{
    HsvThresholdTrackbar* tmp = (HsvThresholdTrackbar*)(ptr);
    tmp->low_H_callback(val);
}
void HsvThresholdTrackbar::onHigh_H_track(int val, void* ptr)
{
    HsvThresholdTrackbar* tmp = (HsvThresholdTrackbar*)(ptr);
    tmp->high_H_callback(val);
}
void HsvThresholdTrackbar::onLow_S_track(int val, void* ptr)
{
    HsvThresholdTrackbar* tmp = (HsvThresholdTrackbar*)(ptr);
    tmp->low_S_callback(val);
}
void HsvThresholdTrackbar::onHigh_S_track(int val, void* ptr)
{
    HsvThresholdTrackbar* tmp = (HsvThresholdTrackbar*)(ptr);
    tmp->high_S_callback(val);
}
void HsvThresholdTrackbar::onLow_V_track(int val, void* ptr)
{
    HsvThresholdTrackbar* tmp = (HsvThresholdTrackbar*)(ptr);
    tmp->low_V_callback(val);
}
void HsvThresholdTrackbar::onHigh_V_track(int val, void* ptr)
{
    HsvThresholdTrackbar* tmp = (HsvThresholdTrackbar*)(ptr);
    tmp->high_V_callback(val);
}



/*---- HSV preset functions ----*/
void HsvThresholdTrackbar::onPreset_track(int val, void *ptr)
{
    HsvThresholdTrackbar* tmp = (HsvThresholdTrackbar*)(ptr);
    tmp->preset_HSV_callback(val);
}
void HsvThresholdTrackbar::preset_HSV_callback(int val)
{

    switch (val)
    {
        case RED:

            this->low_H   = this->red[0];
            this->high_H  = this->red[1];
            this->low_S   = this->red[2];
            this->high_S  = this->red[3];
            this->low_V   = this->red[4];
            this->high_V  = this->red[5];


            break;

        case GRN:

            this->low_H   = this->grn[0];
            this->high_H  = this->grn[1];
            this->low_S   = this->grn[2];
            this->high_S  = this->grn[3];
            this->low_V   = this->grn[4];
```

```
            this->high_V  = this->grn[5];


            break;

        case BLU:

            this->low_H   = this->blu[0];
            this->high_H  = this->blu[1];
            this->low_S   = this->blu[2];
            this->high_S  = this->blu[3];
            this->low_V   = this->blu[4];
            this->high_V  = this->blu[5];


            break;

        default:
            break;
    }

    setTrackbarPos("High␣H",    this->window_name, this->high_H);
    setTrackbarPos("Low␣H",     this->window_name, this->low_H);
    setTrackbarPos("High␣S",    this->window_name, this->high_S);
    setTrackbarPos("Low␣S",     this->window_name, this->low_S);
    setTrackbarPos("High␣V",    this->window_name, this->high_V);
    setTrackbarPos("Low␣V",     this->window_name, this->low_V);
}



/*-------------------------------------------------------------------------------------------*/
/*  Attaches HSV thresholding trackbars onto window showing the binarized input image.
    Inherits Gaussian blur trackbar and uses it to perform gaussian blur on input image,
    prior to HSV thresholding. Inherits ClickForPixelData in order to access selected
    pixel HSV in blurred image, based on selected pixel coordinate from input image */
/*-------------------------------------------------------------------------------------------*/
#ifndef __HSVTHRESHOLDTRACKBAR__
#define __HSVTHRESHOLDTRACKBAR__

#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <math.h>
#include <iostream>

#include "GaussianBlurTrackbar.hpp"
#include "ClickForPixelData.hpp"

class HsvThresholdTrackbar: public GaussianBlurTrackbar, public ClickForPixelData
{
    private:

        /*-------------------------------------------------*/
        /* Thresholding using inRange() in HSV colorspace */
        /*-------------------------------------------------*/
        //reference here: https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html


        cv::String window_name;

        /* Redirection methods to get trackbars to function from within class */
        static void onLow_H_track(int val, void *ptr);
        static void onHigh_H_track(int val, void *ptr);
        static void onLow_S_track(int val, void *ptr);
        static void onHigh_S_track(int val, void *ptr);
        static void onLow_V_track(int val, void *ptr);
        static void onHigh_V_track(int val, void *ptr);


        /* Callback functions called whenever a trackbar is adjusted */
        void low_H_callback(int val);
        void high_H_callback(int val);
        void low_S_callback(int val);
        void high_S_callback(int val);
        void low_V_callback(int val);
        void high_V_callback(int val);

    protected:

        /* This Mat address needs to be shared with the ContourRectangles class */
        cv::Mat *_input_frame;
```

```
        /* HSV presets */
        void preset_HSV_callback(int val);                  // Callback fcn
        static void onPreset_track(int val, void *ptr);     // Redirection fcn
        #define RED 0
        #define GRN 1
        #define BLU 2
        int preset_count = 0;
        const int preset_max = BLU;

        int red[6] = {0, max_value_H, 120, max_value, 0, max_value};    // Preset HSV values for color Red
        int grn[6] = {27, 152, 44, max_value, 0, max_value};            // Preset HSV values for color Green
        int blu[6] = {80, 130, 52, max_value, 0, max_value};            // Preset HSV values for color Blue

        void adjust_slideBars(); // Adjust slidebars according to the last known HSV parameters from the ClickForPixelData Base class

    public:
        HsvThresholdTrackbar(cv::Mat *infrm, const cv::String glugg_nafn);
        void run_HSV_thresh();
};


#endif
```