AUTOMATED INSPECTION OF AN APPLE MOTH

Using Raspberry Pi and Python

HAMK
UNIVERSITY OF APPLIED SCIENCES

Bachelor’s thesis

Degree programme in Automation Engineering

Valkeakoski, spring 2014

El Motasim Gumaa, Mohamed Elfatih Taha

HAMK
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Valkeakoski
Degree programme in Automation Engineering
Option

Author	Gumaa, Taha	**Year** 2014
Subject of Bachelor's thesis	The automated inspection of an apple moth	

ABSTRACT

This thesis is dedicated to the implementation of a machine vision algorithm on a Raspberry Pi Microcomputer.

The commissioning client Teo Kanniainen conducted a research project on the subject of arranging low-cost apple moth inspection on his garden. The research resulted in production of a device collecting pictures of insects in order to process these images later. The contact person Markku Kippola discussed the issues of improved functionality and cost, so the overall aim of this thesis thus was to develop a reliable cheap alternative to Kanniainen's device.

The end product of the this project is an autonomous embedded system for inspection and reporting of apple moth, which functionalities can be further extended to perform other tasks, such as dispatching pesticide control. The designed system harnesses the powerful modularity of the Python scripting language and the OpenCV machine vision framework and utilises the widely used modular microcomputer, the 'Raspberry Pi'.

The reader of this thesis will gain understanding of how to program a Raspberry Pi microcomputer and will become familiar with algorithms used in machine vision that take advantage of both vision and machine learning capabilities. The thesis also provides a real life python code and demonstrates how to install and program machine vision applications with the OpenCV library.

This document describes the issues faced and the solutions found in this particular case.

Keywords Python, OpenCV, Haar-like features, Raspberry Pi, machine vision

Pages 26 p. + appendices 6 p.

ABBREVIATIONS AND TERMINOLOGY

GPIO - General Purpose input/output
Raspberry PI - A single-board computer
IDLE - Integrated development environment
DEPIAN - Operating system developed by The Debian Project.
OS - Operating system
HISTOGRAM - Graphical representation of the distribution of data
GRAYSCALE - Image in which the only colors are shades of gray
OPENCV - Open Source Computer Vision library
PYTHON - A general-purpose, high-level programming language
LIBRARY - A collection of implementations of behavior
NUMPY - A large library of high-level mathematical functions
SMTP - Simple Mail Transfer Protocol
RGB - Additive color model in which red, green, and blue
HSV - Cylindrical-coordinate representations of points in an RGB color model
THRESHOLD - Simplest method of image segmentation
WRAPPER - A function whose main purpose is to call a second function
COMPILER - A program that transforms a program into another language
CONTOUR - A line that joins points of equal elevation
MOMENT - A certain particular weighted average of the image pixels intensities

CONTENTS

1 INTRODUCTION

The main objective of this project was to make a new design for the device introduced in 2011 by the commissioner of this thesis Teo Kanniainen in his research project “Feature extracting and classification of forewings of three moth species based on digital images”.

Kanniainen used a prototype for image acquisition in his research. The prototype was to be placed in targeted fields to collect pictures of insects and to upload them onto a server, where they would be later analysed to recognize the targeted species. Figure 1 below shows the setup:

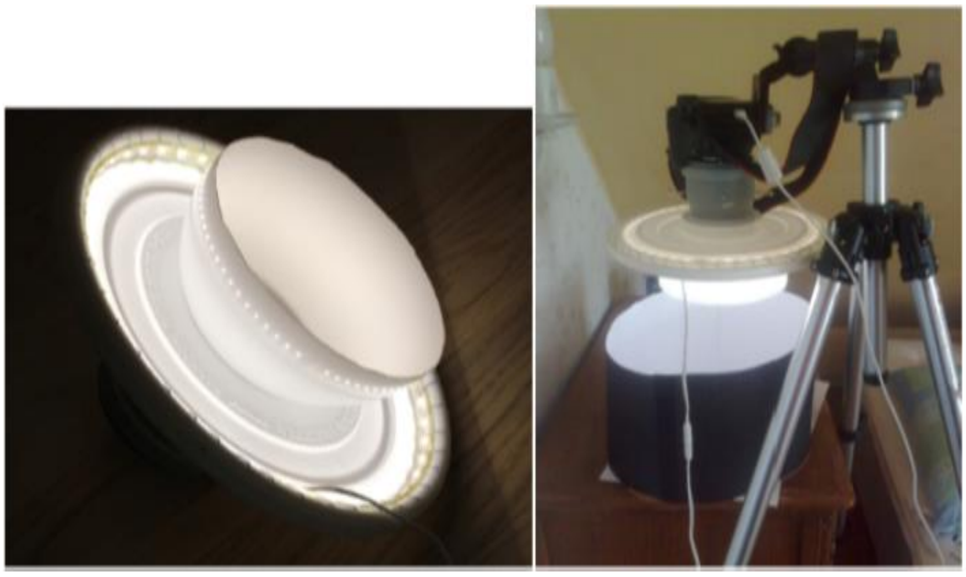


Figure 1 First prototype used by Kanniainen

The device needed to be optimized for better functionality and cost-efficiency, since it was using a Canon 550D SLR and a special lightening led as an image acquisition tools. Later Canon 550D was replaced with an android based camera phone and an IOIO board for controlling, as the phone’s camera needed to be triggered only in case that an object ”insect” has entered the capture zone. Sensing insects entering the capture zone was done by an infra-red sensor with a receiver and a transmitter aligned against each other on both edges of the gate, if an object passed between the transmitter and the receiver, the camera was activated and the captured images were uploaded onto a cloud server such as Dropbox to be analysed and categorized later on for research purposes. Figure 2 shows the concept:

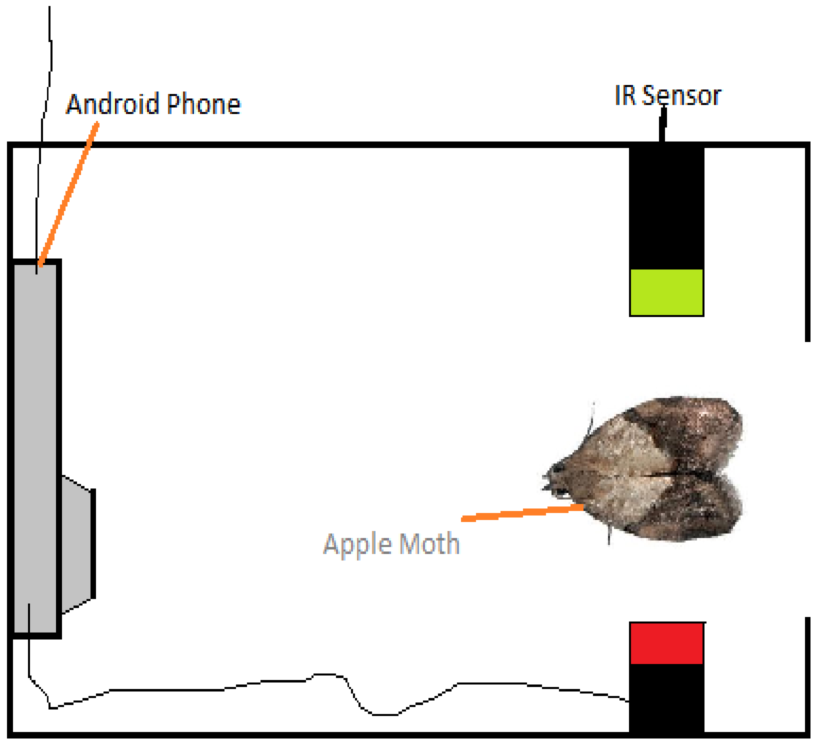


Figure 2 Motion detection was used to activate the camera in the second prototype

These prototypes had several problems such as a high cost, a difficulty to use and a lack of local processing. A combination of the above mentioned factors created the need to develop a fully automated device to carry on the required functions.

The negotiations with the commissioner of this project were focused on:

- Finding which features were needed on the design.
- Which device parts should be replaced, removed or redesigned.
- The budget of each prototype compared to the current design.
- Finding ways to improve the power supply issue of the device as it should be placed in a remote area for long periods of time .
- The connectivity of the device, and the possibility to remotely access it in order to monitor the operations or to modify the settings.
- Image quality and processing possibilities.

These points were taken into consideration, while comparing an array of viable designs and potential replacements. Mainly by comparing the advantages and drawbacks of each potential replacement or design it was possible to eliminate many designs either due to a high cost or not meeting the objective’s minimum requirements.

2 DESIGN

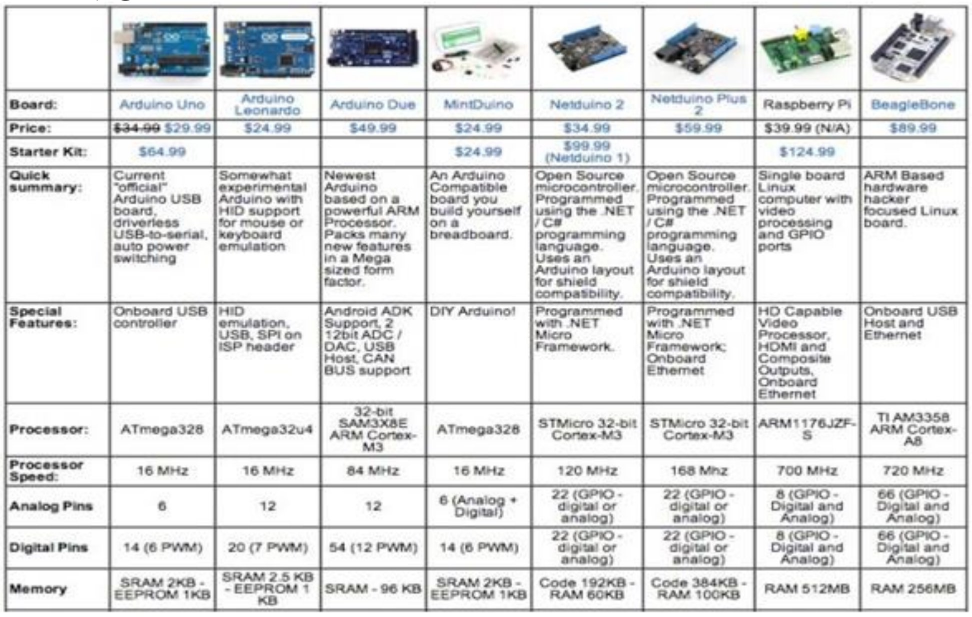
To simplify the design, it was divided into parts based on function:

- Controlling

- Image acquisition
- Motion sensing
- Image processing

2.1 Controlling

There are many microprocessors on the market. suitable for research purposes Taking a look at those will easily point out some of the leading microprocessors brands and products that should be considered, as seen below in (figure 3



Board:	Arduino Uno	Arduino Leonardo	Arduino Due	MintDuino	Netduino 2	Netduino Plus 2	Raspberry Pi	BeagleBone
Price:	~~$34.99~~ $29.99	$24.99	$49.99	$24.99	$34.99	$59.99	$39.99 (N/A)	$89.99
Starter Kit:	$64.99			$24.99	$99.99 (Netduino 1)		$124.99	
Quick summary:	Current "official" Arduino USB board, driverless USB-to-serial, auto power switching	Somewhat experimental Arduino with HID support for mouse or keyboard emulation	Newest Arduino based on a powerful ARM Processor. Packs many new features in a Mega sized form factor.	An Arduino Compatible board you build yourself on a breadboard.	Open Source microcontroller. Programmed using the .NET / C# programming language. Uses an Arduino layout for shield compatibility.	Open Source microcontroller. Programmed using the .NET / C# programming language. Uses an Arduino layout for shield compatibility.	Single board Linux computer with video processing and GPIO ports	ARM Based hardware hacker focused Linux board.
Special Features:	Onboard USB controller	HID emulation, USB, SPI on ISP header	Android ADK Support, 2 12bit ADC / DAC, USB Host, CAN BUS support	DIY Arduino!	Programmed with .NET Micro Framework.	Programmed with .NET Micro Framework; Onboard Ethernet	HD Capable Video Processor, HDMI and Composite Outputs, Onboard Ethernet	Onboard USB Host and Ethernet
Processor:	ATmega328	ATmega32u4	32-bit SAM3X8E ARM Cortex-M3	ATmega328	STMicro 32-bit Cortex-M3	STMicro 32-bit Cortex-M3	ARM1176JZF-S	TI AM3358 ARM Cortex-A8
Processor Speed:	16 MHz	16 MHz	84 MHz	16 MHz	120 MHz	168 Mhz	700 MHz	720 MHz
Analog Pins	6	12	12	6 (Analog + Digital)	22 (GPIO - digital or analog)	22 (GPIO - digital or analog)	8 (GPIO - Digital and Analog)	66 (GPIO - Digital and Analog)
Digital Pins	14 (6 PWM)	20 (7 PWM)	54 (12 PWM)	14 (6 PWM)	22 (GPIO - digital or analog)	22 (GPIO - digital or analog)	8 (GPIO - Digital and Analog)	66 (GPIO - Digital and Analog)
Memory	SRAM 2KB - EEPROM 1KB	SRAM 2.5 KB - EEPROM 1 KB	SRAM - 96 KB	SRAM 2KB - EEPROM 1KB	Code 192KB - RAM 60KB	Code 384KB - RAM 100KB	RAM 512MB	RAM 256MB

Figure 3 Microcontroller comparison, (Marker Shed 2013.)

In this case the ability to use a camera with the chosen microprocessor was essential, also connectivity, ease of programing and the cost of the board should be considered.

Considering the points above Raspberry Pi was a very good candidate for final selection. The Raspberry Pi board had advantages such as ease of programming and use, ability to use a camera module, and an intuitive user interface. Figure 4 shows the raspberry pi and its major components:

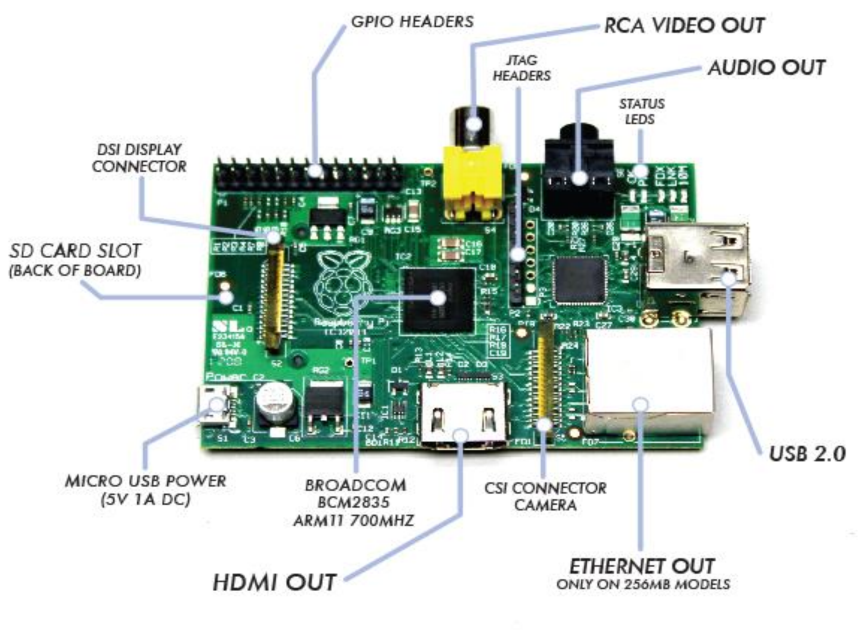

GPIO HEADERS
RCA VIDEO OUT
JTAG HEADERS
AUDIO OUT
STATUS LEDS
DSI DISPLAY CONNECTOR
SD CARD SLOT (BACK OF BOARD)
USB 2.0
MICRO USB POWER (5V 1A DC)
BROADCOM BCM2835 ARM11 700MHZ
CSI CONNECTOR CAMERA
HDMI OUT
ETHERNET OUT ONLY ON 256MB MODELS

Figure 4 Raspberry Pi board, (raspberrypi.org 2013.)

Raspberry Pi boards come in 3 models: A, B 256 and B 512 which have identical specifications except the following the ones compared in Table 1.

Table 1 Differences between Raspberry pi models,(raspberrypi.org 2014)

Model	A	B 256	B 512
USB PORTS	1	2	2
ETHERNET	No	Yes	Yes
RAM	256	256	512
PRICE	18	35	45

The factor affecting this project the most up to that point was the price of the device. As already mentioned the cost efficiency was one of the most critical points discussed with the thesis commissioner. Another noteworthy fact was that neither Ethernet nor the extra RAM were needed for this type of process.

There were a few Linux editions that were made specifically to operate on Raspberry Pi RASBIAN, the edition parallel to Debian Linux editions for PCs. The system is provided with Python IDE.

2.2 Image acquisition

Raspberry Pi has its own camera module which was developed especially for it. The camera module is connected to the Pi board through the CSI port via a 15 pin ribbon cable. The camera module is 5 MP and it supports

1080p/720p/640x480p high definition video. The module comes with a fixed focus photo sensor. Figure 5 below shows the camera

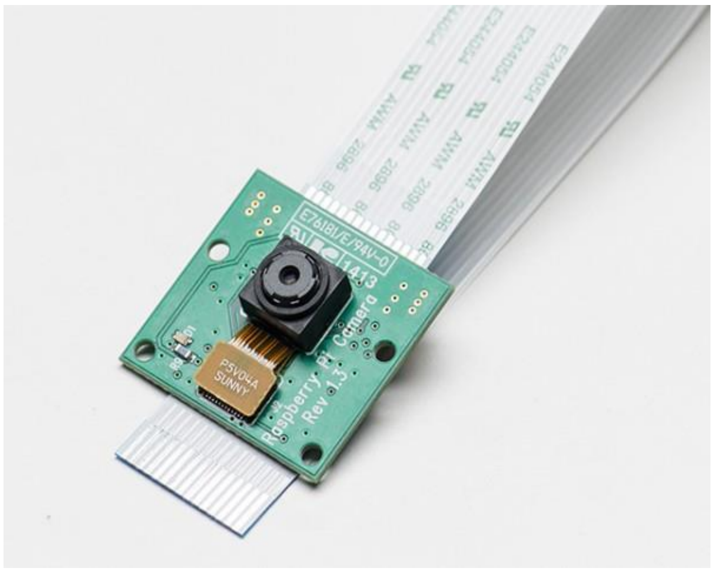


Figure 5 Raspberry Pi Camera module with a standard ribbon cable, (Pi Camera installation guide 2014.)

Fixed focus later on became an issue later on as the image processing required sharp images that were impossible to get of objects closer than 550mm away from the sensor. In order to get sharper pictures for closer objects a lens had to be installed in front of the sensor. Figure 6 shows images before and after correction.

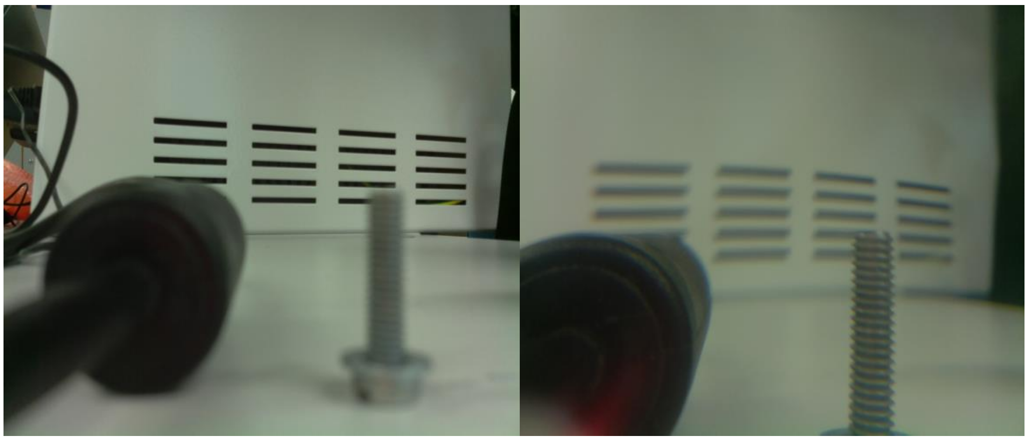


Figure 6 Sharpness comparison before (left) and after(right) using a lens, object at 40mm

2.3 Motion sensing

The same idea which was used in an earlier prototype was implemented to this project as it was efficient enough and reliable. The GPIO General Purpose Input/output port allowed easy installation of infrared sensing components such as a phototransistor and an IR diode, needed to design the sensing unit. The idea depends on the line-breaking method. Below is detailed map of GPIO in figure 7:

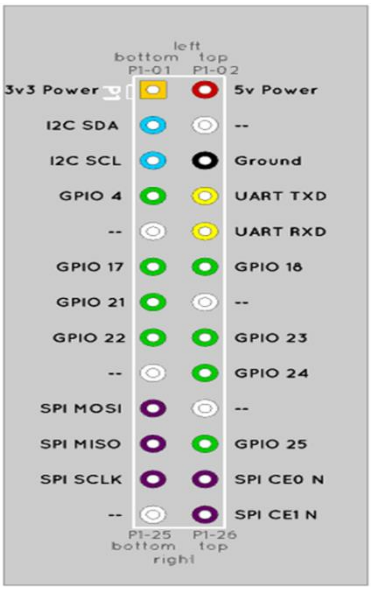


Figure 7 GPIO port, (Clough 2012,8)

Any object passing between the transmitters (IR led) and receiver (photo-transistor) should trigger the camera module. Figure 8 shows a schematic on how this was done

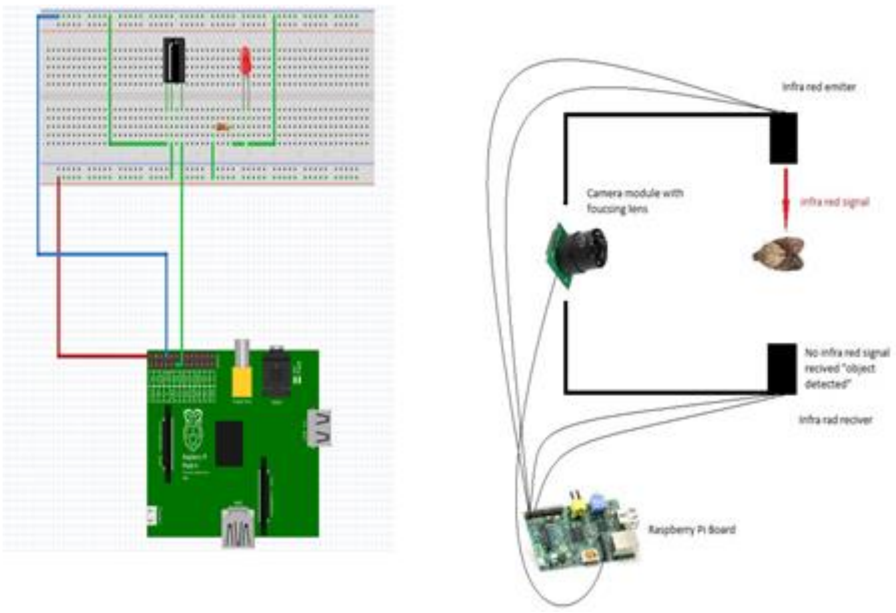

Infra red emiter
Camera module with foucsing lens
infra red signal
No infra red signal recived "object detected"
Infra rad reciver
Raspberry Pi Board

Figure 8 Infra-red receiver and transmitter connection to GPIO

2.4 Image processing

The system the client already had in place was not particularly sophisticated and it could not be described as automated and it certainly had no machine vision implemented.

Our final product was fully capable of achieving the same results and additionally it will only notify the client if the target bug was detected. The user will not have to visit the drop box website, because text message or email will be sent to him once something is detected.

3 USING OPENCV

Implementing machine vision application with tools such MATLAB or Lab VIEW turned out to be more complicated and costly than using OpenCV and due to:

- the cost of buying these products
- portability
- compatibility with common market products

While trying to figure out a way to make a machine vision software that works on the Raspberry Pi, Lab View was considered. However although Lab View has an easy graphical programming interface, there is one problem that makes it virtually impossible to work with other products. One reason for that is that Lab View requires runtime engine, meaning it cannot run as independent software on PC, let alone on an embedded device such as the Raspberry Pi.

Secondly, when programing for embedded devices with MATLAB , the code has to be ‘translated’ to a C script which then undertake further editing to be run on such platforms. For all these reason, choosing a library that allows to write “native code” that runs exactly as predicted the selection was made in favour of the OpenCV

Although virtually all documentation of OpenCV online was for code written on C++ or C, python was chosen for two main reasons:

- Ease of installation
- Python module structure

Code written in python is much easier to read and thus debug, and the variety of modules available makes programing more straightforward.
For comparison refer to the next 2 chapters for demonstration how installation of OpenCV is done on both C++ and python environment.

3.1 OpenCV installation on Code::Blocks, C++IDE

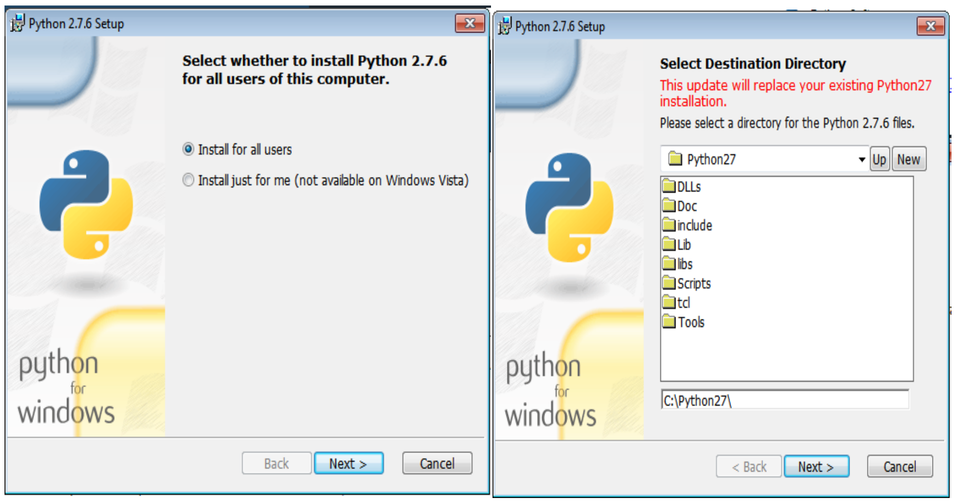
- Install Code:: Blocks IDE Form
- Install minGW

- Add minGW to system path
- Download and install OpenCV
- Download and install Cmake
- Compile your OpenCV using Cmake
- Add OpenCV bin to system path (similar to the previous step)
- Build your OpenCV binaries using cmake form command prompt
- Link code blocks and OpenCV library
- Edit the compiler and linker

As demonstrated OpenCV installation is unnecessarily complicated, especially considering that each of the steps described above includes smaller inside steps in itself which leaves room for errors in configuring the environment.

3.2 Installing OpenCV on Python

- go to: https://www.python.org/download/releases/2.7.6/
- select the ‘Windows X86-64 MSI Installer (2.7.6)
- http://sourceforge.net/projects/numpy/files/NumPy/1.7.1/numpy-1.7.1-win32-superpack-python2.7.exe/download
  All you need is to press next couple of times as shown in figure 9


Python 2.7.6 Setup

Select whether to install Python 2.7.6 for all users of this computer.

[✓ Install for all users]

Install just for me (not available on Windows Vista)

Python 2.7.6 Setup

Select Destination Directory

This update will replace your existing Python27 installation.

Please select a directory for the Python 2.7.6 files.

Python27

DLLs
Doc
include
Lib
libs
Scripts
tcl
Tools

C:\Python27\

Figure 9 Python setup screen

- download python numeric library, after installation it should be downloaded on the following directory: C: /Python27/
- go to: opencv/build/python/2.7 folder
- copy cv2.pyd to C:/Python27/lib/site-packeges
- open Python IDLE and type following codes in Python terminal:

```
import cv2
print cv2._version_
```

For further assistance on installation go to:
http://docs.opencv.org/trunk/doc/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html

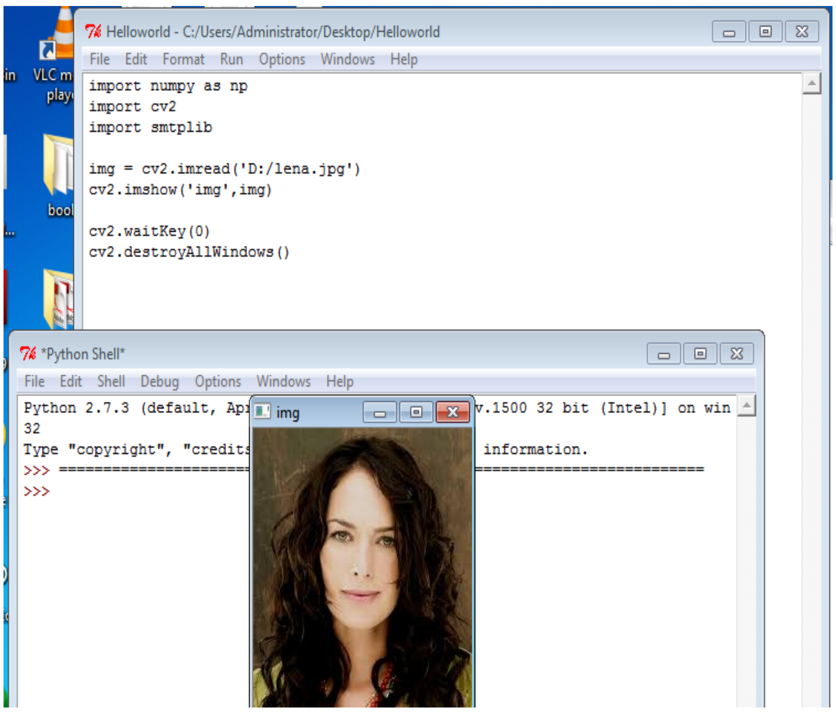
3.3 Testing OpenCV

A simple test program can be used to verify the installation validity. The program reads an image and displays it to the user. The following steps must be taken:

- Download the famous Lena machine vision picture
- Save it to your desktop
- Give it the name Lena.jpg
- On python IDLE
- Go to file and
- Make new window
- Paste the following code on it :

```
import numpy as np
import cv2

img = cv2.imread('D:/lena.jpg')
cv2.imshow('img',img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Afterwards the work is saved by again going to file and selecting “Save as”. File extension has to be set as “.py”, like hello world.py for example. This will help idle to compile the code; otherwise it will not be identified as Python script. If all the above mentioned steps were followed the results should be similar what you see in figure 10:


Helloworld - C:/Users/Administrator/Desktop/Helloworld
File Edit Format Run Options Windows Help
Python Shell
File Edit Shell Debug Options Windows Help
img

```
import numpy as np
import cv2
import smtplib

img = cv2.imread('D:/lena.jpg')
cv2.imshow('img',img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Python 2.7.3 (default, Ap                          v.1500 32 bit (Intel)] on win
32
Type "copyright", "credit                          information.
>>> =====================                          ==========================
>>>
```

Figure 10 Test program

3.4 OpenCV functions

In real life applications one would read video or streams or pictures straight from the camera. But for the purpose of this test program the picture is read from ‘file’ which is common practice to make sure the system is well set up.
To explain the software a little:

```
import numpy as np
import cv2
```

This part of the script tells python that in order to run the program successfully, the following libraries are needed: CV2 (OpenCV library) and Numpy (Python numeric library).

A library is a file that contains functions written in higher language with instructions on how to operate it on a machine level. For python to know what to do with the function “img = cv2.imread('D:/lena.jpg')”, it has to go to the cv2 library and fetch the machine level instruction for imread . The value between the two praises () is called parameter and it is how variable is transferred to the function for processing.

4 DETECTING THE APPLE MOTH

Detecting the moth proved more difficult than expected. Most of the ‘off the shelf ‘algorithms had two main principles:

- unified colour
- unified shape
-

The fact that a moth is a living organism brings to light the fact that it comes in many different sizes, shapes and colours. See figure 11 for comparison:

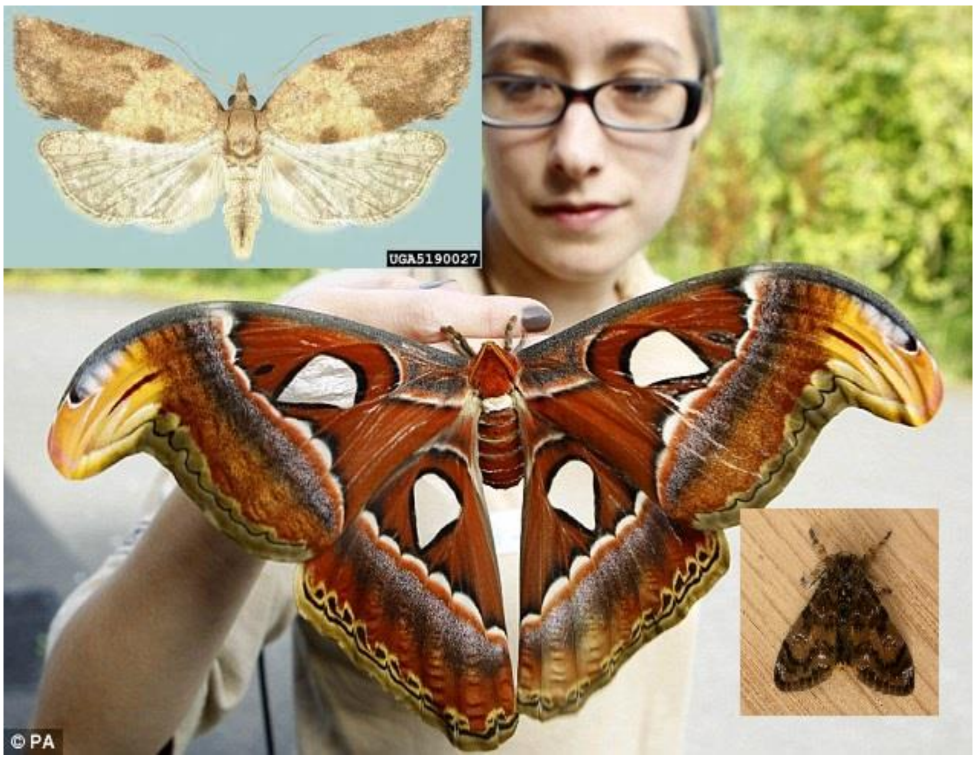


Figure 11 Sizes of moth

To overcome the challenge presented by multiple moth forms, a machine learning algorithm called “Haar-like features” was considered. The basic principle here is that the software is fed with a range of ‘Positive images’ ranging from 5000 to 9000 of the objects in question. Simultaneously an equivalent amount of ‘Negative images’ is fed into to software. Based on these two inputs, the software generates a set of descriptive vectors which are stored in an XML format, to be used later in the detection.

However due to the time constraint, and discouraging result obtained from a cascade trained with small ‘Positive data’ which was practically less than 12 clear images, and another hundred that was obtained from databases scattered on the internet a new method was put in place to achieve the detection. However the ‘Haar cascade’ will be used after a better set of ‘Positives’ is collected.

4.1 The Haar-like features detection

Haar like feature detection is based on the (Viola -Jones detection algorithm) that uses Haar wavelet in mathematics which is sequence of rescaled rectangles analysis similar to the Fourier analysis. The idea is to identify the rectangles by contrast, meaning pixel intensity, in the viola jones algorithm a target size is placed on top of subject and similarity is calculated based vectors length. Figure 12 below shows the scale and rotation invariance of the algorithm. (Bressers 2009).To further illustrates this concept se figure 12

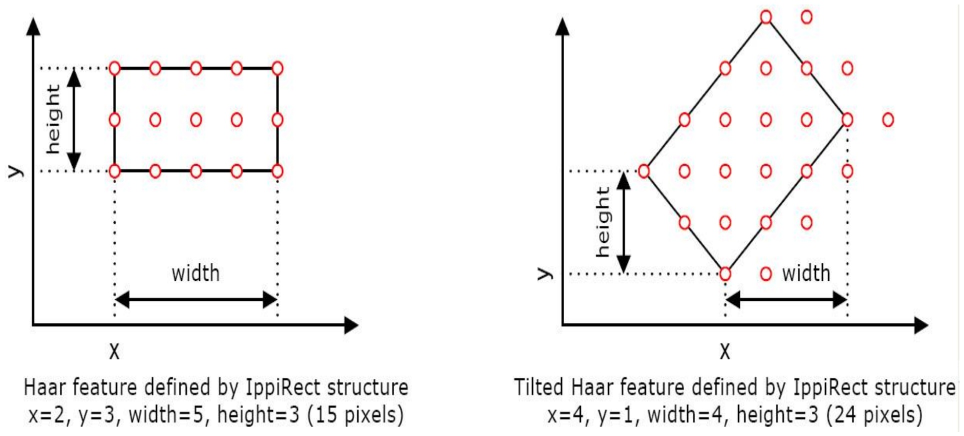

height
y
width
x
Haar feature defined by IppiRect structure
x=2, y=3, width=5, height=3 (15 pixels)
height
y
width
x
Tilted Haar feature defined by IppiRect structure
x=4, y=1, width=4, height=3 (24 pixels)

Figure 12 Haar feature, (OpenCV documentation 2013.)

From the figure above it is visibly clear that changes on the x plain are followed by corresponding shift on the y plain, which makes Haar method very efficient and rotation invariant, meaning that the object can’t take any rotational position without affecting its detecting ability.

$$\mathrm{sum} = I(C) + I(A) - I(B) - I(D).$$

Figure 13 Haar equation

But extracting the features of the moth is only first part of the task. The Haar framework thus needs a mechanism to extract, archive, retrieve and test. This is the machine learning part of it.

In the OpenCV this is done, by training the system using a "positive" set of images that contain the different varieties of rotation scaling and colouring of the object in question, along that a set of "negative" images is also introduced to the system. That way the computer can "learn" when and if an object is present. The numbers 1, 2 and 3 refers to the stages of training, with each stage the entire object is detected and passed on to the next stage for more refining, see figure 14 below:

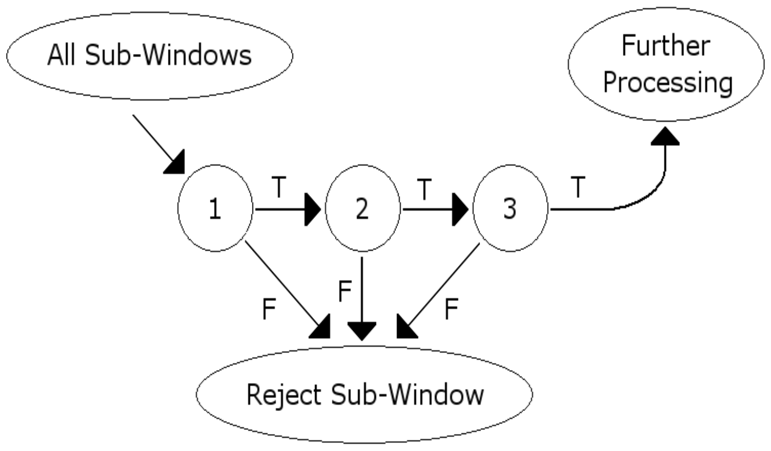


Figure 14 Cascade is taking control of the identification process, (Bressers 2009)

4.2 The Haar classifier

The haar classifier is pretty straight forward in OpenCV. The process begins with training the "Haar classifier". This process is basically teaching the computer a set of negative and positive images and will result in the generation of an XML file, containing the vectors of each positive and negative image that was fed to the system. Based on it the computer will compare each new input to its archive.

Steps of creating classifier in python:

- Create a text file of negative descriptions
- Create a text file of positive descriptions
- Use the create samples command to generate samples
- Train your cascade using the samples made in step 3

Figure 15 below contain command line on doing that.

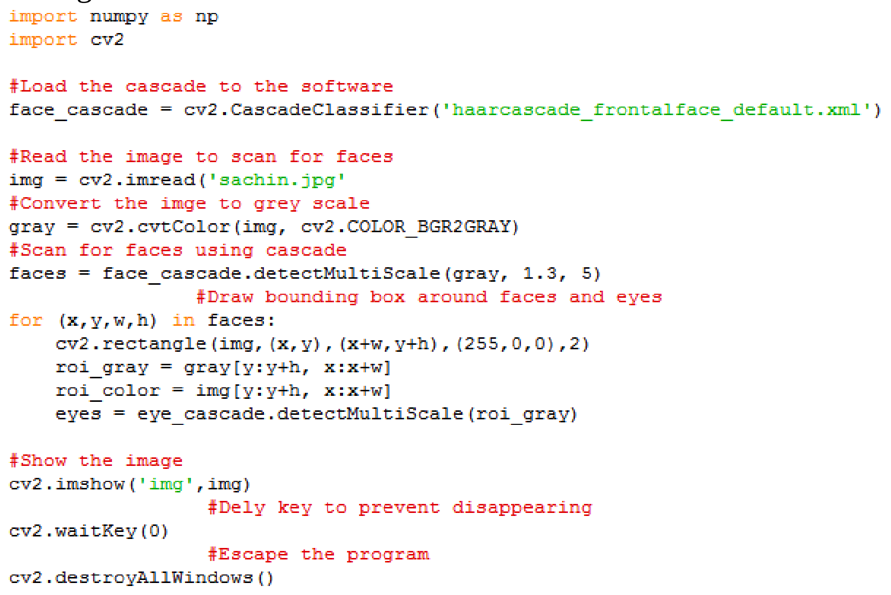
```
"images/positive 0.png"  1  120 160 40 40
"images/positive 1.png"  2  200 120 40 60  80 60 20
20

$ <opencv_createsamples> -vec <binary_description> -
info <positive_description> -bg
<negative_description>

$ <opencv_createsamples> -vec <binary_description> -
image <positive_image> -bg <negative_description>
```

Figure 15 Creating samples and training the cascade, (Howse 2013,123-131)

There is a waiting period of around two days. During this time the computer should be left powered. A notification message informing that the process is complete will be sent.

4.3 Loading the cascading

After the training of the system is complete, the resulting xml file has to be imported into the software. A program based on the face detection xml cascade that is provided by OpenCV. Below is code with comment in red. See figure 16.



```
import numpy as np
import cv2

#Load the cascade to the software
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

#Read the image to scan for faces
img = cv2.imread('sachin.jpg'
#Convert the imge to grey scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#Scan for faces using cascade
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
                #Draw bounding box around faces and eyes
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)

#Show the image
cv2.imshow('img',img)
                 #Dely key to prevent disappearing
cv2.waitKey(0)
                 #Escape the program
cv2.destroyAllWindows()
```

Figure 16 Haar cascade sample with face detection cascade

4.4 Problems with the “Haar-Cascade” method

- Requires a huge amount of data
- False positive is a common occurrence
- Laborious

Using an alternative measure to a moth specific algorithm, detection was achieved. The main features used were histogram of moth wings and a calculated pixel area approximation for the enclosing contour.

The idea was to capture the image and perform histogram enhancement and boosting in order to identify the moth. This approach has its own challenges but it is the most reliable method for this specific application.

The next chapters will deal with image processing, histogram calculation, histogram back projection and extraction, contour allocation, contour drawing, bounding boxes and bounding boxes thresholds.

5 PROCESS OF DETECTION

The goal of processing the image is to filter noise and boost desired features, namely the characteristics on which the detection will be based. The following steps will guide the reader through the image processing that was implemented in the program.

5.1 Greyscale conversion

The first image transformation to be done for reaching a detectible feature is greyscale conversion. It simply means converting the RGB colour span into a corresponding image with one colour (grey) with different intensity across the image. Figure 17 illustrate the conversion

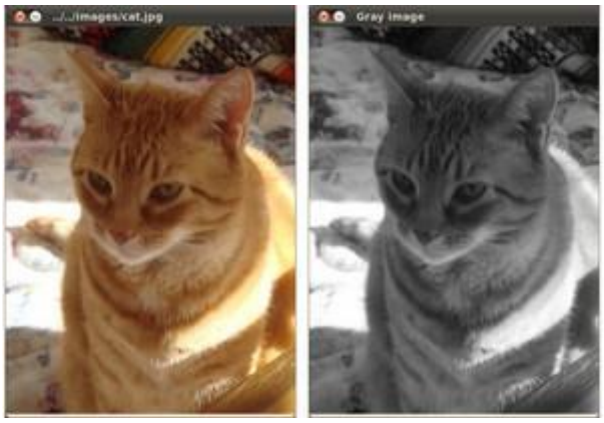


Figure 17 Grayscale conversion, (OpenCV documentation, 2014.)

Grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of grey from black to white. That is the reason that only one channel (colour) is chosen to represent the image when converting from RGB (which is an array of (255,255,255)). (OpenCV documentation 2014.)
To see split-channel RGB see figure 18:

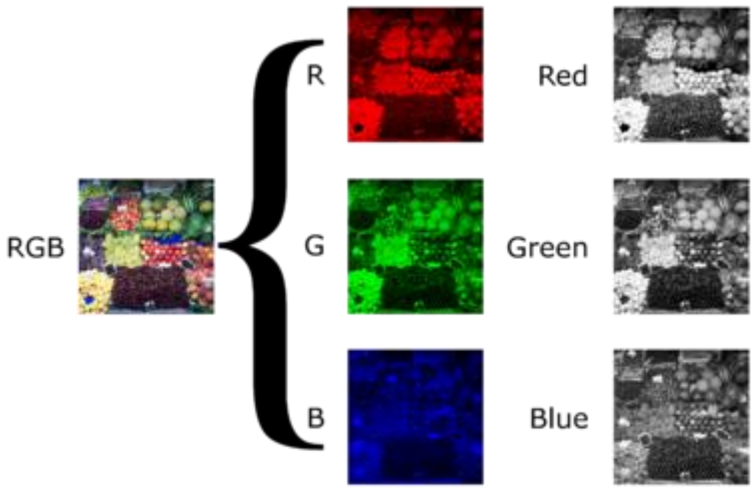


Figure 18 Gray scale conversion, (OpenCV documentation 2014.)

Grayscale conversion is performed in OpenCV using the following command:

```
cvtColor( image, gray_image, CV_BGR2GRAY );
```

Figure 19 Grayscale conversion command, (OpenCV documentation 2014.)

Implementation of the above function would thus look something like this:

```
namedWindow( imageName, CV_WINDOW_AUTOSIZE );
namedWindow( "Gray image", CV_WINDOW_AUTOSIZE );

imshow( imageName, image );
imshow( "Gray image", gray_image );
```

Figure 20 Grayscale conversion command implementation, (OpenCV documentation 2014.)

5.2 BGR To HSV conversion

OpenCV uses the BGR colour format as opposed to more commonly used RGB which results in reversed values, if the format is not considered. The focus on BGR to HSV is needed because histogram calculation at a later stage requires the image to be in the HSV domain.

$$h = \begin{cases} 0 & \text{if } \max = \min \\ (60^\circ \times \frac{g-b}{\max - \min} + 0^\circ) \bmod 360^\circ, & \text{if } \max = r \\ 60^\circ \times \frac{b-r}{\max - \min} + 120^\circ, & \text{if } \max = g \\ 60^\circ \times \frac{r-g}{\max - \min} + 240^\circ, & \text{if } \max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } \max = 0 \\ \frac{\max - \min}{\max} = 1 - \frac{\min}{\max}, & \text{otherwise} \end{cases}$$

$$v = \max$$

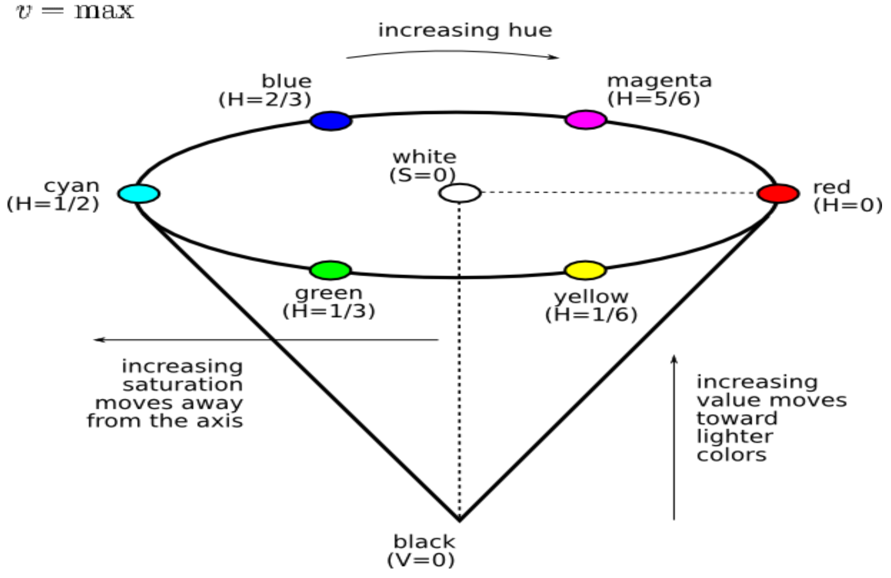


Figure 21 BGR to HSV formula and visualization (OpenCV documentation 2014.)

HSV stands for Hue, Saturation, and Volume, and the conversion command in OpenCV is as follows:

```
# Convert BGR to HSV
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Figure 22 BGR to HSV conversion command

Because BGR to HSV changes only the domain but not the value, a converted image would be exactly the same, only with new parameters.

5.3 Calculating histogram

An image histogram is a graphical representation of the number of pixels in an image as a function of their intensity. Figure 23 below shows Histogram distribution :

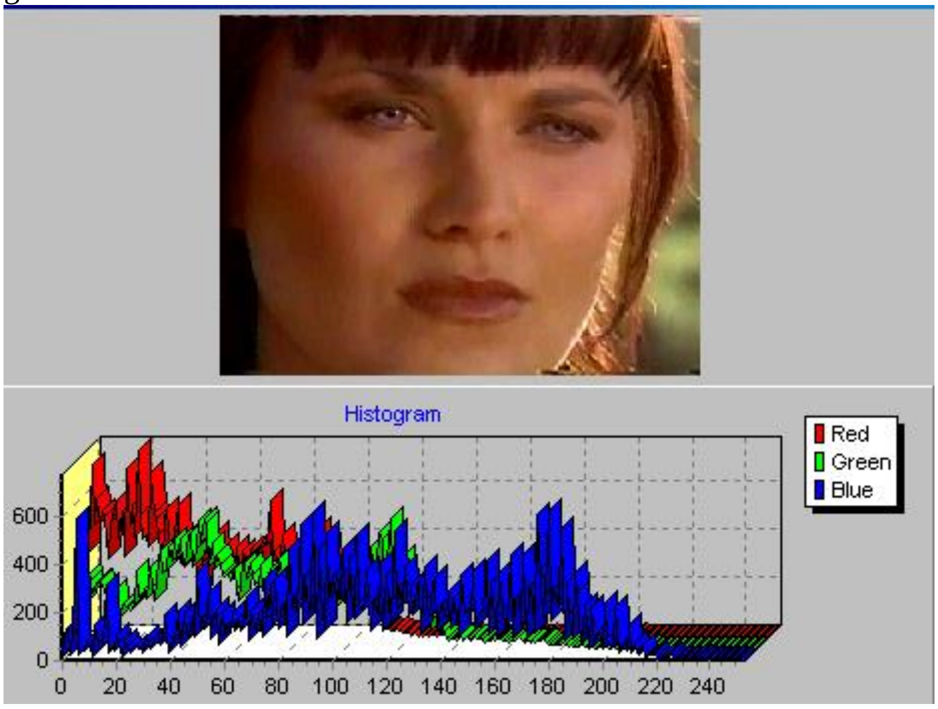


Figure 23 Histogram, (Class VLHistogram.TVLHistogram 2002.)

Performing Histogram in OpenCV is done through the following command:

```
roihist = cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )
```

Figure 24 Histogram command, (OpenCV documentation 2014.)

5.4 Histogram Equalization

Histogram equalization means averaging and redistributing intensity of tones. An image in grayscale would look like this after histogram equalization see figure 25:

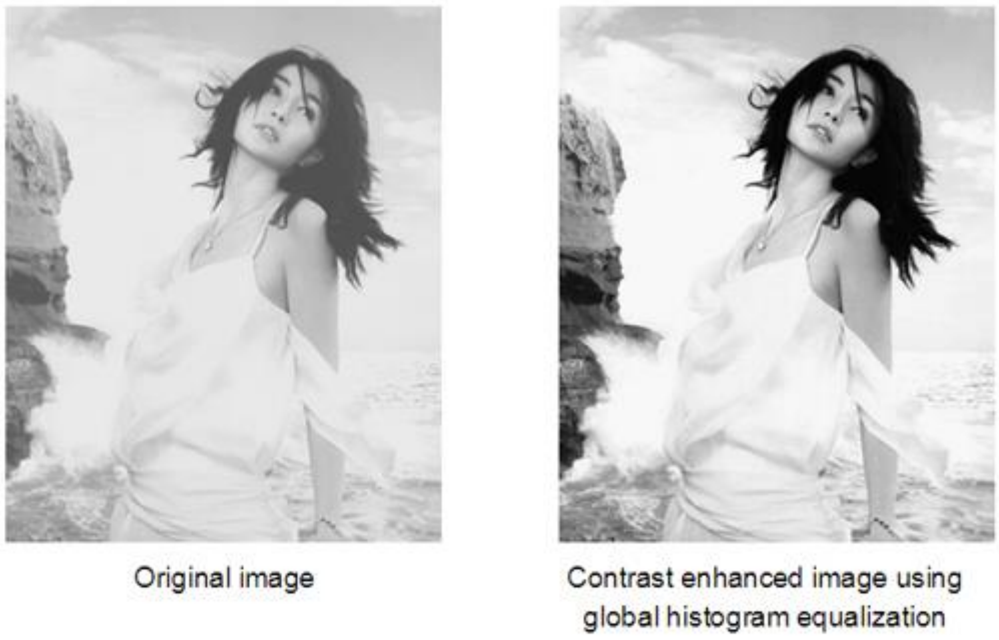
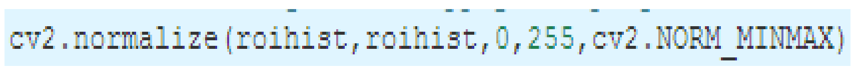


Figure 25 Histogram equalization, (OpenCV documentation 2014.)

To perform equalization in OpenCV the following command is used:

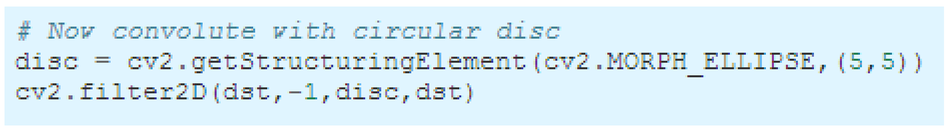


```
cv2.normalize(roihist,roihist,0,255,cv2.NORM_MINMAX)
```

Figure 26 Histogram equalization command, (OpenCV documentation 2014.)

5.5 Histogram convolution

Convolution is redistributing histogram on an image, for example a function. Consider this: g(x,y) = h(x,y) * f(x,y). It is essential for performing higher order processing like blurring and dilation. To perform convolution in OpenCV the following is used:



```
# Now convolute with circular disc
disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
cv2.filter2D(dst,-1,disc,dst)
```

Figure 27 Histogram equalization convolution, (OpenCV documentation 2014.)

5.6 Histogram back projection:

Back projection is, as the name implies, repainting a set of histogram values on gray scale or otherwise manipulated image, highlighting certain areas.

The area isolated is then used for the detection after being refined, look at figure 28 to get an idea:

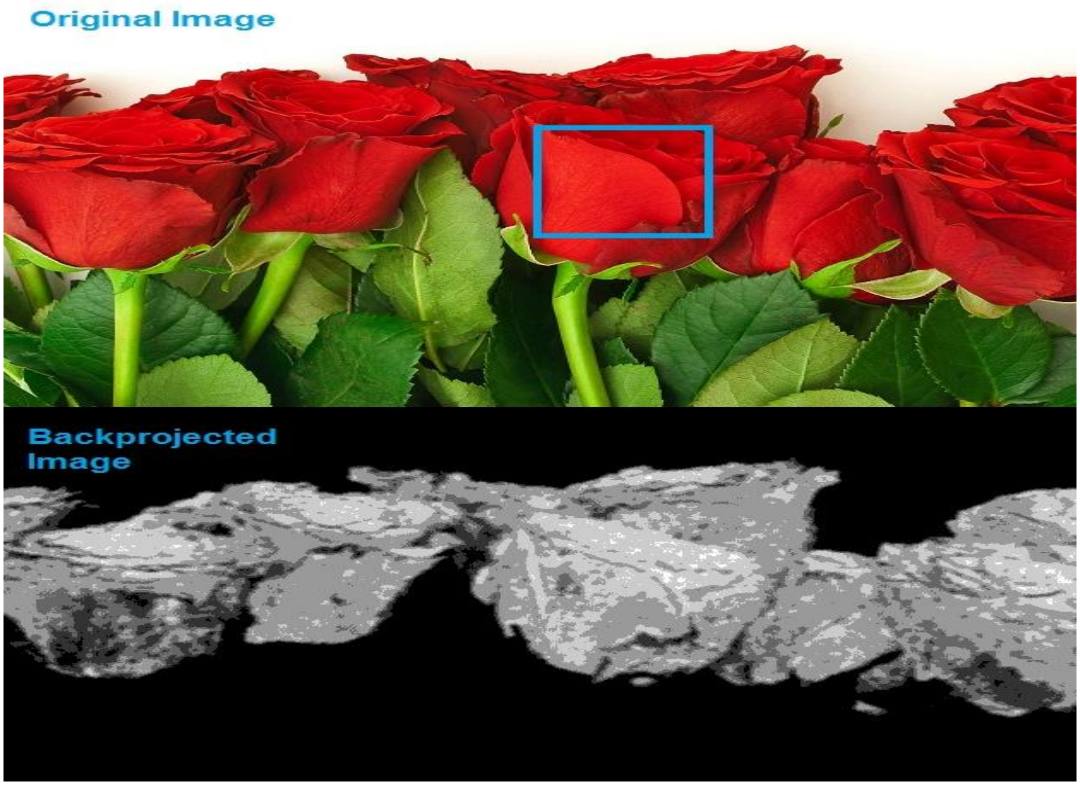
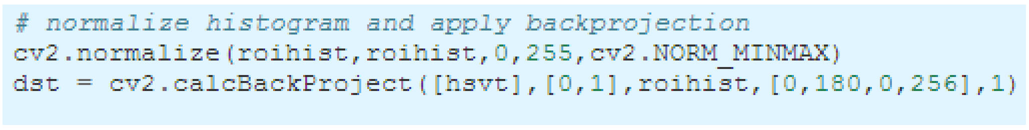


Figure 28 Back projection, (OpenCV documentation 2014.)

To perform back projection in OpenCV the following command is used:

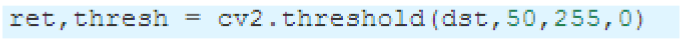


```
# normalize histogram and apply backprojection
cv2.normalize(roihist,roihist,0,255,cv2.NORM_MINMAX)
dst = cv2.calcBackProject([hsvt],[0,1],roihist,[0,180,0,256],1)
```

Figure 29 Back projection command, (OpenCV documentation 2014.)

5.7 Thresholding

Thresholding is adjusting the HSV or BGR colour domain value. This is done to enhance or to get rid of a particular colour.



```
ret,thresh = cv2.threshold(dst,50,255,0)
```

Figure 30 Thresholding command, (OpenCV documentation 2014.)

Numbers 50, 255, 0 represent the desired color in the HSV Domain. To see the effect of thresholding refer to figure 31:

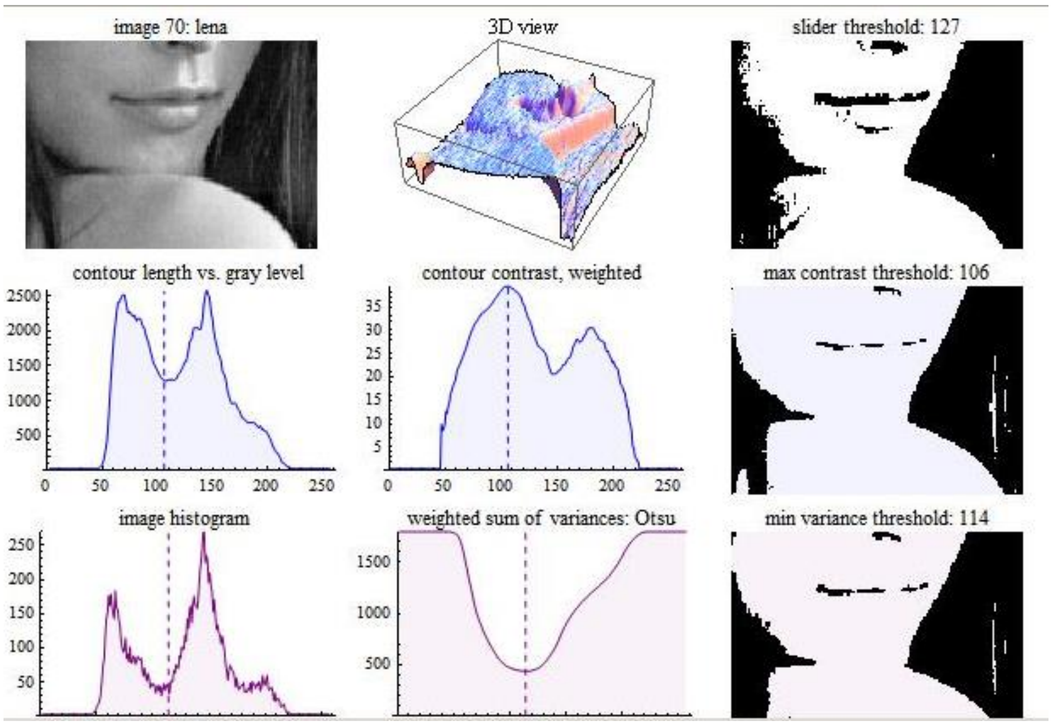


Figure 31 Thresholding, (OpenCV documentation 2014.)

6 MORPHOLOGICAL TRANSFORMATION

So far image processes that have been discussed related in one way or another to the colours and its domain. But another form of processing is equally needed and important - morphological processing. Morphological processing is based on the image shape and coordinates. A lot of material is available in the OpenCV documentation on this subject, but for the purposes of this thesis only some aspects of morphological processing will be mentioned.

6.1 Erosion

Erosion is used to eliminate foreground boundaries which result in shrinkage of objects within the image. It is normally used in cases when noise causes two objects to stick together forming a larger one. See figure 32 for effect of erosion

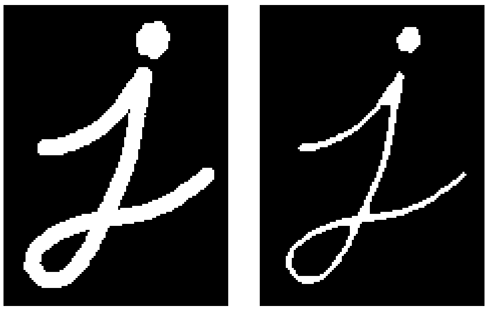


Figure 32 Applying erosion, (OpenCV tutorials 2014.)

To achieve results similar to the one in figure above, the following command is used:

```
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
```

Figure 33 Erosion command, (OpenCV tutorials 2014.)

6.2 Dilation

Dilation is the opposite of erosion and it is used in some cases to expand an object area in order to ease detection or further processing. . See figure 32 for effect of erosion:

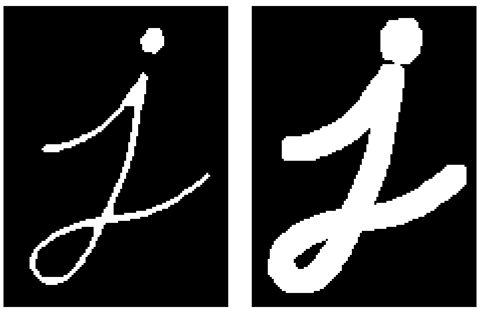


Figure 34 Dilation, (OpenCV tutorials 2014.)

7 CONTOURS

Contours is normally the final stage of object's detection. The OpenCV library draws contours around pixels of different intensities. Based of properties of these contours, such as length, area, edges, openness or closeness the detection of an object is done.

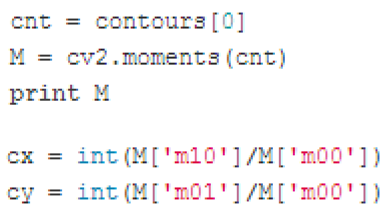
7.1 Finding contours

Images tend to have an abundance of details. As a result of this extracting contours from multi-channelled image will not achieve useable results. A better approach would be to thresh an image to binary state before attempting to find contours. See figure 35:

```
ret,thresh = cv2.threshold(imgray,127,255,0)
image, contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

Figure 35 Code for threshold and contour finding, (OpenCV documentation 2014.)

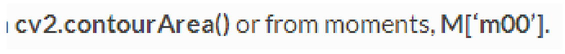
After contours are found (list)/Moments, a multitude of operations can be performed. See figure 36 it shows moments calculation:



```
cnt = contours[0]
M = cv2.moments(cnt)
print M

cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

Figure 36 Operations, (OpenCV documentation 2014.)

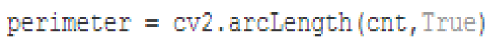
For example, determining the area for each contour:



cv2.contourArea() or from moments, M['m00'].

Figure 37 Contour area, (OpenCV documentation 2014.)

Or Arc length of a contour:

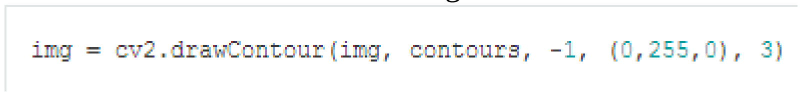


```
perimeter = cv2.arcLength(cnt,True)
```

Figure 38 Contour area length (OpenCV documentation 2014.)

7.2 Drawing Contours

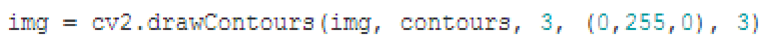
To draw all the contours in an image:



```
img = cv2.drawContour(img, contours, -1, (0,255,0), 3)
```

Figure 39 Drawing all contours command, (OpenCV documentation 2014.)

Or the forth one:



```
img = cv2.drawContours(img, contours, 3, (0,255,0), 3)
```

Figure 40 Drawing forth contour command, (OpenCV documentation 2014.)

7.3 Drawing bounding boxes

Drawing a box around a given closed contour requires 4 parameters: width, height and y coordinates

```
x,y,w,h = cv2.boundingRect(cnt)
img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

Figure 41 shows result of this:

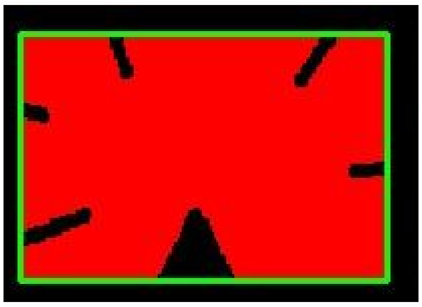


Figure 41 Bounding box

8 SOFTWARE STRUCTURE

The architecture and data flow is designed to separate functionality, in order to work around hardware limitations, mainly power and memory. For example, the camera module needed extra dependencies to work in OpenCV. Additionally, the SD card memory is limited to 4 GB, so storing images on regular bases will damage the device's memory. To avoid this captured image is assigned to a location, from which OpenCV retrieves it.

The resulting processed image is forwarded via email, while overwriting the previous one. This serves both as extra feature and helps to conserve memory. Figure 42 below shows this graphically

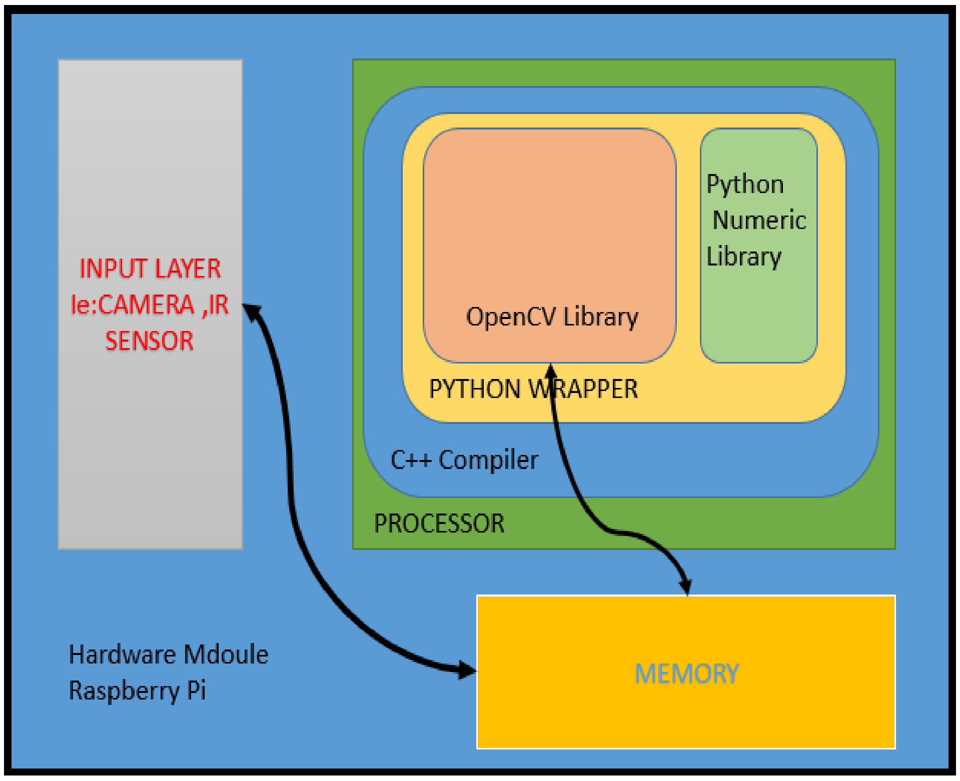


Figure 42 System Architecture

Such properties as power and connectivity would normally be included, but due to the nature of this document they were left out. Although it is noteworthy to point out that the completing the task assigned to us would have been vastly easier, if a constant supply of energy was available, but because the unit will be installed in remote locations that was not possible. Network coverage for 3G was also not tested.

8.1 Email Module

Email library for python to send an email notification to the user once the previous function for detecting was compiled.
This feature can be implemented in two ways:

- sending a simple test message (SMS)
- Sending an email with a picture attachment of the moth.

In the appendix 2 python code a detailed and commented code for achieving this is presented.

In order to send as SMS message .Taha@student.hamk.fi with a generic email address provided by GSM provider, for example 040123456@dna.fi.

8.2 Automatic boot

To make Raspberry run directly to the software several steps must be undertaken. First, write on the raspberry terminal:

```
'#!/bin/bash
/bin/login f root
```

Second, make the code executable:

```
chmod a+x /bin/autologin.sh
Edit /root/.bashrc
if [[ $(tty) == '/dev/tty1' ]]; then
/root/operation/mothdetector.py
Endif''
```

(Clough 2012, 7.)

This will root the device and give automatic login bypassing the login detail. Then execute the software when booting.

9 CONCLUSION

The project was successful in detecting the apple moth from images across the internet. Without controlled background, the accuracy stands at 72 % .This number is based on testing 25 images of the target apple moth and detection was achieved on 18 of them at the moment and will reach approximately 95 % when the casing with unified colour background is added. Adding the unified coloured background will dramatically increase the accuracy by preventing confusion between wing colour histogram and background colour.

To conclude:

- Reduction of cost from 600 euro (cost of DSLR camera and setup) to 50 euros (cost of raspberry and power backup) was achieved

-Automatic inspection of the moth was achieved, this replaced the manual process

-Automated notification was setup by creating email alert with image attachment

-well documented and commented code is kept for future modification

For full code check appendix 1 through 5 it contains software flowchart, python software along with command for adjusting camera and GPIO ,appendix 6 contains IR camera activation.

Figure 43 below shows current detection accuracy with background closer in properties to the moth .we can see it is still possible to accurately detect the insect

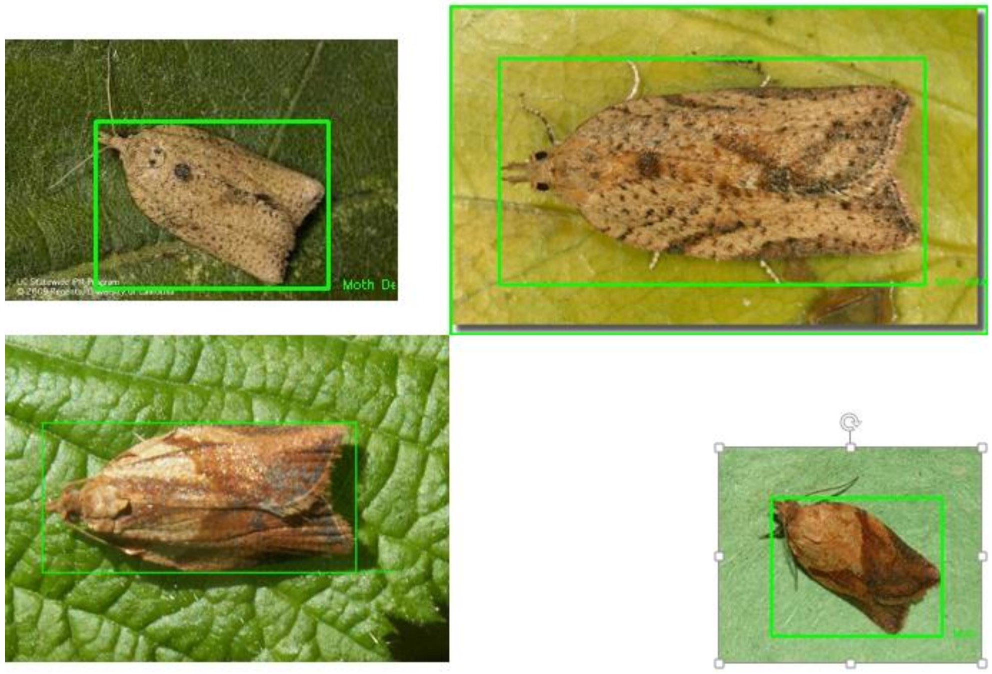


Figure 43 Results of detection

SOURCES

Bob Clough. 2012. Raspberry Pi as an Embedded Platform. Pdf file.
thinkl33t.co.uk)

Class VLHistogram.TVLHistogram. 2002.
http://www.mitov.com/wiki/index.php?title=Class_VLHistogram.TVLHistogram

David Beazley and Brian K Jones. 2013. Python Cookbook, 3rd edition.
ISBN 978-1-449-34037-7.

Daniel Lelis Baggio.2013. Mastering OpenCV with Practical Computer Vision.
ISBN 978-1-84951-782-9

Willem Bressers.2009.Real-time face verification using a low resolution camera
Bachelor Thesis, Universiteit van Amsterdam

Joseph Howse. 2013. OpenCV Computer Vision with Python
ISBN 978-1-78216-392-3

Kanniainen, T. 2011. Feature extracting and classification of forewings of
three moth species based on digital images. Licentiate's thesis, Tampere
University of Technology

Low level programming. 2014.
http://elinux.org/

OpenCV tutorials. 2014.
http://opencv-python-tutroals.readthedocs.org/

OpenCV documentation. 2014.
http://docs.opencv.org/

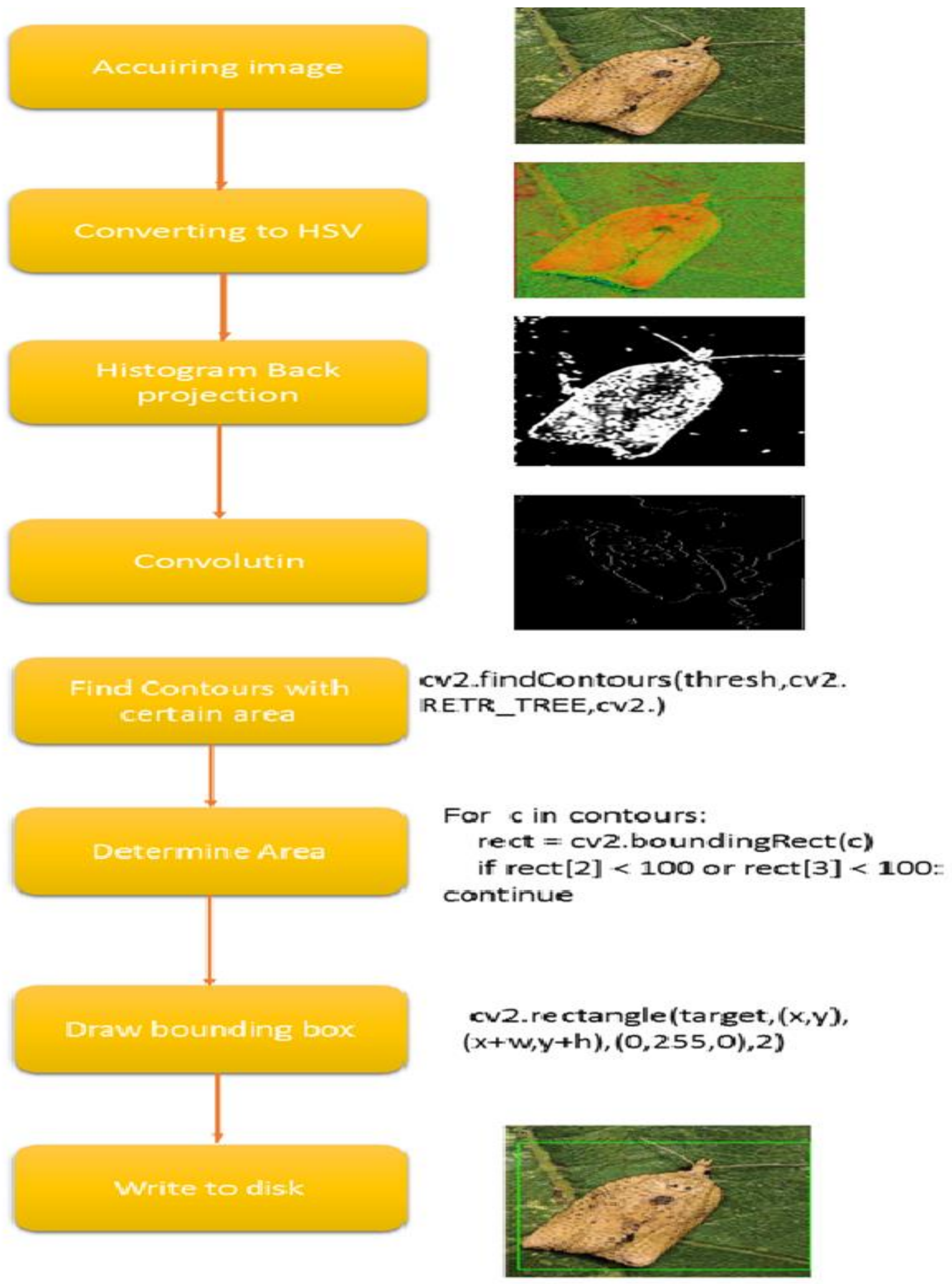
Pi Camera installation guide. 2014.
http://thepihut.com/pages/how-to-install-the-raspberry-pi-camera

Supaporn Spanurattana. 2012 Advanced Image Processing. Pdf file.
http://www.img.cs.titech.ac.jp/~akbari/pmwiki/uploads/Site/Supaporn-rep.pdf

GPIO programming, 2013.
http://www.raspberrypi.org/tag/gpio/

Appendix 1

PROGRAM FLOWCHART



```
cv2.findContours(thresh,cv2.
RETR_TREE,cv2.)
```

```
For  c in contours:
   rect = cv2.boundingRect(c)
   if rect[2] < 100 or rect[3] < 100:
continue
```

```
cv2.rectangle(target,(x,y),
(x+w,y+h),(0,255,0),2)
```

Appendix 2

PYTHON CODE # is followed by comment (python comment standard)

```
# importing libraries 1-opencv and 2-numiric library
import cv2
import numpy as np
#Read image of target histogram(wing cropped image)
roi = cv2.imread('C:/Users/Administrator/Desktop/Presentation/bugroi.jpg')
#Covert it to HSV
hsv = cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)
#Read the image of subject (saved by rapberry camera
target = cv2.imread('C:/Users/Administrator/Desktop/Presentation/Samples/bug10.jpg')
#Resize the image of subject
targetr = cv2.resize(target, (0,0), fx=0.2, fy=0.2)
#Convert Subject image to HSV
hsvt = cv2.cvtColor(target,cv2.COLOR_BGR2HSV)


# calculating object histogram
roihist = cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )

# normalize histogram and apply backprojection
cv2.normalize(roihist,roihist,0,255,cv2.NORM_MINMAX)
dst = cv2.calcBackProject([hsvt],[0,1],roihist,[0,180,0,256],1)

# Now convolute with circular disc
disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
cv2.filter2D(dst,-1,disc,dst)

# threshold and binary AND
ret,thresh = cv2.threshold(dst,40,200,0)
#Find Contours in subject
contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
#Calcualte Bounding box ccording to Contour area
for c in contours:
    rect = cv2.boundingRect(c)
    if rect[2] < 100 or rect[3] < 100: continue
    #Draw the bounding box
    print cv2.contourArea(c)
    x,y,w,h = cv2.boundingRect(c)
    cv2.rectangle(target,(x,y),(x+w,y+h),(0,255,0),2)
    #Print ''Moth detected ''
    cv2.putText(target,'Moth Detected',(x+w+10,y+h),0,0.3,(0,255,0))
    #Save the image
```

Appendix 2

PYTHON CODE # is followed by comment (python comment standard)

```
    #Save the image
    cv2.imwrite('C:/Users/Administrator/Desktop/Presentation/result.jpg',target)

#Import Email libraries (All of them are needed
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from smtplib import SMTP
import smtplib

#Determine the server to be used
SMTP_SERVER = 'smtp.gmail.com'
#Determine the port of server (usually 587)
SMTP_PORT = 587
#Determine the logging details of the client
sender = 'taha.elfatih@gmail.com'
password = "Googleearthworknew10"
#Write down the recipient email address .subject and message body
recipient = 'mohamed.taha@student.hamk.fi'
subject = 'attachement moth '
message = 'Images attached.'
#Directory of attachment image
directory = "/tmp/images/"
#Sending the email 'Class'
def main():
    msg = MIMEMultipart()
    msg['Subject'] = 'Python emaillib Test'
    msg['To'] = recipient
    msg['From'] = sender
    #Defining the image attachement directory
    img = part = MIMEApplication(open("D:/filters.jpg","rb").read())
    img.add_header('Content-Disposition', 'attachment', filename= 'filters.jpg')
    msg.attach(img)
    #Server logging protocol for logging
    part = MIMEText('text', "plain")
    part.set_payload(message)
    msg.attach(part)

    session = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)

    session.ehlo()
    session.starttls()
    session.ehlo
    session.login(sender, password)

    session.sendmail(sender, recipient, msg.as_string())
    session.quit()
if __name__ == '__main__':
    main()
```

Appendix 3

CAMERA SOFTWARE SETUP

Execute the following instructions on the command line to download and install the latest kernel, GPU firmware and applications. You will need an internet connection for this to work correctly.

```
sudo apt-get update
sudo apt-get upgrade
```

Now you need to enable camera support, using the `raspiconfig` program you will have used when you first set up your Raspberry Pi.

```
sudo raspi-config
```

Use the cursor keys to move to the camera option and select enable. On exiting `raspi-config` it will ask to reboot. The enable option will ensure that on reboot the correct GPU firmware will be running (with the camera driver and tuning), and the GPU memory split is sufficient to allow the camera to acquire enough memory to run correctly.

To test that the system is installed and working, try the following command:

```
raspistill -v -o test.jpg
```

The display should show a 5-second preview from the camera and then take a picture, saved to the file test.jpg, while displaying various informational messages.

Appendix 4

CAMERA CONTROL OPTIONS

`--sharpness, -sh` Set image sharpness (-100 to 100)
Set the sharpness of the image, 0 is the default.
`--contrast, -co` Set image contrast (-100 to 100)
Set the contrast of the image, 0 is the default
`--brightness, -br` Set image brightness (0 to 100)
Set the brightness of the image, 50 is the default. 0 is black, 100
is white.
`--saturation, -sa` Set image saturation (-100 to 100)
Set the colour saturation of the image. 0 is the default.
`--ISO, -ISO` Set capture ISO
Sets the ISO to be used for captures. Range is 100 to 800.
`--vstab, -vs` Turn on video stabilization
In video mode only, turn on video stabilization.
`--ev, -ev` Set EV compensation
Set the EV compensation of the image. Range is -10 to +10, default is

Appendix 5

CAMERA IR ACTIVATION   CODE   (#) Is followed by Comment

```
# Importing the needed modules for GPIO, Camera and time for waiting function

import RPi.GPIO as GPIO

import picamera

from time import sleep

# To set the input 17 to read from phototransistor

GPIO.setup(17, GPIO.IN)

# Definding the camera

camera = picamera.PiCamera()

while 1:

# Input from phototransistor is not one so there is something in between the      phototransistor and the IR Led

if GPIO.Input(17) =! 1 :

#  Wait 3 seconds till the insect is in front of camera

time.sleep (3)

# Capture image

camera.capture('image.jpg')

time.sleep (3)

# Capture another image

camera.capture('image2.jpg')

time.sleep (3)

# Capture another image

camera.capture('image3.jpg')
```