Appendices

A. User Documentation

This page gives you an overview of all available documents which are created for this project.
The main libraries used in this project are:

- Bebop.py
- detect_color.py
- MergeImages.py
- Overlapping.py

B. Installation Instructions

The instruction for installing the necessary code and packages to run the codes are for Ubuntu 16.04.

B1 Installations necessary to run MergeImages.py

In order to run this code, the programming language Python 2.7 must be installed. Ubuntu comes with Python 2.7 pre installed. To update Python to 3.6 the following command can be executed in Terminal:

```
sudo apt-get install python3.6
```

Additional python packages is also necessary. These packages are: numpy, scikit and PIL.

To install numpy and scikit, the following command can be used in Terminal:

```
sudo apt-get install python-numpy python-scipy
```

To install PIL, the following command can be used in Terminal:

```
python3 -m pip install Pillow
```

B2 Installations necessary to run Overlapping.py

The same installation as in section B2 is necessary to run this code, as well as the following packages: matplotlib, scikit-image and scikit-learn.

To install matplotlib, type the following command in Terminal:

```
sudo apt-get install python3-matplotlib
```

To install scikit-images, type in the following command in Terminal:

```
sudo apt-get install python-skimage
```

To install scikit-learn, type in the following command in Terminal:

```
pip install -U scikit-learn
```

B3 Installations necessary to run detect_color.py

The same installation as in section B2 is necessary to run this code, as well as the following packages:time,cv2,numpy and Image from PIL.

time packages is already present in Python's standard library

To install cv2, type in the following command in Terminal:

```
pip install opencv
```

To install numpy, type in the following command in Terminal:

```
sudo apt-get install python-numpy
```

To install PIL, type in the following command in Terminal:

```
sudo apt-get install python-PIL
```

C. <u>All the Code (including data necessary to run the code)</u>

C1 prot_1.py

```
#!/usr/bin/python

''' Program that connects to a parrot bebop drone V.1, making it fly autonomously to perform areal footage of an area. The drone flies to the marker autnomously, takes areal pictures of it and returns to another selected marker. '''

#Install the required libraries
''' Common libraries '''
import sys
import cv2
import os
```

```
import time
import signal
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import detect_color as detect

'''Bebop libraries '''
from bebop import Bebop
from commands import movePCMDCmd
from capdet import detectTwoColors, loadColors
from bebop import *
from apyros.metalog import MetaLog, disableAsserts
from apyros.manual import myKbhit, ManualControlException


import time

# Create global variables
drone = Bebop()

''' Function that provides an angle for the pitch according to tilt angle of the camera '''
def obtainSpeed(AngleCamera):

    if AngleCamera == -30:
    return 20,0,False
    if AngleCamera == -42:
    return 10,-1, False
    if AngleCamera == -54:
    return 6,-2, True
    if AngleCamera == -66:
    return 5,-2, True
    return 20, 0, False


''' Function that provides an angle for the pitch according to tilt angle of the camera '''
def obtainSpeed2(AngleCamera):

    if AngleCamera == -30:
    return 27,0,False
    if AngleCamera == -42:
    return 9,-3, False
    if AngleCamera == -54:
    return 5,-3, True
    if AngleCamera == -66:
    return 5,-3, True
```

```
        return 20, 0, False


''' Function that creates a folder called "Photos" '''
def createFolder():

        # Create the name of the folder's path
        file_path = "/home/bebopdev/kata/Photos/"
        directory = os.path.dirname(file_path)

        # If it does not exist, create it
        if not os.path.exists(directory):
        os.makedirs(directory)

''' Function that allows the drone to take 8 frames of a video '''
def testCamera():

        # Local variables
        i = 0

        try:
        # source of the video
        cap = cv2.VideoCapture('./bebop.sdp')

        # Look for 20 frames of video
        while (i < 20):
            # obtain the frame
            ret,img = cap.read()

            if ret:
                                                        # show the frame in the screen
                                                        cv2.imshow('img', img)
                                                        cv2.namedWindow('img',
cv2.WINDOW_NORMAL)
                                                        cv2.waitKey(1)
            # update the state of the drone
            drone.update()
            i += 1
        except (TypeError) as e:
        pass



''' Function that allows the drone to take 8 frames of a video '''
def takePict():
```

```
        # Local variables
        i = 0

        try:
        # source of the video
        cap = cv2.VideoCapture('./bebop.sdp')

        # Look for 20 frames of video
        while (i < 20):
            # obtain the frame
            ret,img = cap.read()

            if ret:
                                                        # save images from the frame 11
to above
                                                        if (i > 11):
                                                        # Save the image
                                                        imageName = "Photo_"+ str(i)
+".jpg"
                                                        cv2.imwrite(imageName, img)

                                                        # obtain the coordinates of the
amrker in the image

distancex,x,distancey=detect.coordenates(imageName)
                                                        distancex2 = abs(distancex)

                                                        # Check if the marker is in the
centre of X in the image
                                                        if distancex2 < 80:
                                                        # Print status messages
                                                        print 'Object Centred'
                                                        print 'x: ', distancex, 'y :',
distancey

                                                        # Move the image to the folder
                                                        current_path = "Photo_"+ str(i)
+".jpg"
                                                        final_path = "Photos/" +
current_path
                                                        os.rename(current_path,
final_path)
                                                        # Say that the marker is centred
and return the coordinates
                                                        i = 20
                                                        return True, distancex, distancey
```

```
            # update the state of the drone
            drone.update()
            i += 1
        # Say that the marker is NOT centred and return the coordinates
        return False, distancex, distancey
        except (TypeError) as e:
        pass

''' Function that helps to find a target '''
def findTarget(angleCamera):

        # Local variables
        centred =  False
        YawAngle = 15


        while not centred :
        # Obtain the images
        centred, xvalue, yvalue = takePict()

        if not centred :
            # modify the angle to Yaw
            if xvalue < 0:
                                                        # Print status messages
                                                        print "Rotating left"

                                                        YawAngle = -16
                                                        RollAngle = -7
            else :
                                                        # Print status messages
                                                        print "Rotating right"

                                                        YawAngle = 16
                                                        RollAngle = 7
            if yvalue > 0:
                                                        pitch = 11
            else:
                                                        pitch = 5
            # Make the drone Yaw
            # This compensate the movement to the back of the drone
            drone.moveBebop(0, 1, 0)
            drone.wait(1)

            # Rotate(Yaw)
            drone.moveBebop(0, pitch-3, YawAngle)
            drone.wait(1)
```

```
            drone.moveBebop(RollAngle,pitch-1,0)
            drone.wait(1)

            # This compensate the movement to the back of the drone
            drone.moveBebop(0, 1, 0)


        # Check the position in Y to tilt the camera
        if yvalue < -30:
        if angleCamera > -80:
            angleCamera = angleCamera - 12

        # Print status messages
        print "Angle Camera :", angleCamera

        return angleCamera, xvalue,yvalue

''' function to arrive to target'''
def arrive():
        # Local varaibles
        angleCamera = -30
        there = False

        while not there:
        # Find and centre the target
        NewangeleCamera,x,y = findTarget(angleCamera)
        angleCamera = NewangeleCamera

        # Tilt the camera
        drone.moveCamera(tilt=angleCamera, pan=0)

        # Print status messages
        print "Moving forward"

        # This compensate the movement to the back of the drone
        drone.moveBebop(0, 1, 0)
        drone.wait(1)

        # Obtain the angle for the roll movement
        speed,counter,there =obtainSpeed(angleCamera)

        # Print status messages
        print 'Pitch :', speed, ' Counter: ',counter

        if x > 0:
            Roll = 7
```

```
        else:
            Roll = -7

        # Make the pitch movement
        drone.moveBebop(Roll, speed, 0)
        drone.wait(1)
        drone.moveBebop(0, counter, 0)
        drone.wait(1)

        # This compensate the movement to the back of the drone
        drone.moveBebop(0, 1, 0)

        # Emergency break, in case of moving to much forward
        if angleCamera >= -42 and y < -120 :
            print ' Emergency stop '
            there = True
            # Stop the bebop
            drone.moveBebop(0, -5, 0)
            drone.wait(1)

''' Function that allows the drone to take 8 frames of a video '''
def UAVPict(Position):

        # Local variables
        i = 0

        try:
        # source of the video
        cap = cv2.VideoCapture('./bebop.sdp')

        # Look for 20 frames of video
        while (i < 20):
            # obtain the frame
            ret,img = cap.read()

            if ret:
                                                        # save images from the frame 11
to above
                                                        if (i > 11):
                                                        # Save the image
                                                        imageName = "UAV_"+ Position
+ str(i - 11) +".jpg"
                                                        cv2.imwrite(imageName, img)

                                                        # Print status messages
                                                        print 'Photo taken : ', Position
```

```
                                                    # Move the image to the folder
                                                    current_path = imageName
                                                    final_path = "Photos/" +
current_path

                                                    os.rename(current_path,
final_path)

            # update the state of the drone
            drone.update()
            i += 1
        except (TypeError) as e:
        pass


''' Function that makes the drone hover over the target and take pictures '''
def AerealFootage():

        # Adjust the camera
        drone.moveCamera(tilt= -90, pan=0)

        # Hover
        drone.moveBebop(-1, 0, 0)
        drone.moveBebop(1, 0, 0)
        drone.moveBebop(0, 1, 0)

        # Take the pictures centre
        UAVPict("First")

        # Setting Yaw, Roll and Pitch values
        YawAngle = 30
        RollAngle = 9
        pitch = 9

        # Rotate
        drone.moveBebop(0, 1, 0)
        drone.wait(1)
        drone.moveBebop(0, 4, YawAngle)
        drone.wait(2)
        drone.moveBebop(RollAngle,pitch,0)
        drone.wait(1)
        drone.moveBebop(0, 1, 0)

        # Take the pictures centre
        UAVPict("Second")
```

```
    # Setting Yaw, Roll and Pitch values
    YawAngle = 30
    RollAngle = 9
    pitch = 12

    # Rotate
    drone.moveBebop(0, 1, 0)
    drone.wait(1)
    drone.moveBebop(0, pitch, YawAngle)
    drone.wait(2)
    drone.moveBebop(RollAngle,pitch+1,0)
    drone.wait(2)

    # Take the pictures centre
    UAVPict("Third")


''' Function that place the drone in a better position to the take the UAV photos '''
def accomodate():

    # take the pictures
    drone.moveBebop(0, 1, 0)
    centred, xvalue, yvalue = takePict()

    if not centred :

    # modify the Yaw value
    if xvalue < 0:
        # Print status messages
        print "Rotating left"

        YawAngle = -15
        RollAngle = -7
    else :
        # Print status messages
        print "Rotating right"

        YawAngle = 15
        RollAngle = 7

    # Obtain the pitch value
    if yvalue > 0:
        pitch = 9
    else:
        pitch = 5
```

```
        # Rotate
        drone.moveBebop(0, pitch, YawAngle)
        drone.wait(1)
        drone.moveBebop(RollAngle,pitch,0)
        drone.wait(1)

        # Make the drone fly to 1.5 m of altitude
        drone.hover()
        drone.flyToAltitude(1.5)
        drone.flyToAltitude(1.5)

        # Make the drone Yaw
        # This compensate the movement to the back of the drone
        drone.moveBebop(0, 1, 0)

''' function to arrive to the second target'''
def arrive2():

        # Local varaibles
        angleCamera = -30
        there = False

        while not there:
        # Find and centre the target
        NewangeleCamera,x,y = findTarget(angleCamera)
        angleCamera = NewangeleCamera

        # Tilt the camera
        drone.moveCamera(tilt=angleCamera, pan=0)

        # Print status messages
        print "Moving forward"

        # This compensate the movement to the back of the drone
        drone.moveBebop(-1, 1, 0)
        drone.wait(1)

        # Obtain the angle for the roll movement
        speed,counter,there =obtainSpeed2(angleCamera)

        # Print status messages
        print 'Pitch :', speed, ' Counter: ',counter

        # Obtain some roll values for compensation
        if x > 0:
            Roll = 7
```

```
        else:
            Roll = -7

        # Make the pitch movement
        drone.moveBebop(Roll, speed, 0)
        drone.wait(1)
        drone.moveBebop(-1, counter, 0)
        drone.wait(1)

        # This compensate the movement to the back of the drone
        drone.moveBebop(-1, 1, 0)


''' Make the drone return the initial position (with a marker) '''
def comeBack():

        # Tilt the camera to -30
        drone.moveCamera(tilt= -30, pan=0)

        # Rotate the drone one more time
        YawAngle = 25
        RollAngle = -17
        pitch = 15

        # Rotate
        drone.moveBebop(0, 1, 0)
        drone.wait(1)
        drone.moveBebop(-5, 10, YawAngle)
        drone.wait(3)
        drone.moveBebop(RollAngle,pitch,0)
        drone.wait(2)

        # Make it fly to 1.8 meters
        drone.flyToAltitude(1.8)
        drone.moveBebop(-5, 10, 0)
        drone.wait(2)

        # Make it come back
        drone.moveBebop(-2, 5, 0)
        drone.wait(2)
        arrive2()


'''Main function '''
def main():
```

```
    # We create the folder for the images
    createFolder()

    # Prepare the camera

    # Establish the signal of the camera with the computer
    signal.signal(signal.SIGINT, signal_handler)

    # adjust the camera for the video
    drone.moveCamera(tilt= -30, pan=0)

    # Start the video
    drone.videoEnable()
    testCamera()

    # take off
    drone.takeoff()
    drone.flyToAltitude(1.8)
    drone.moveBebop(0, 5, 0)
    drone.flyToAltitude(1.8)
    drone.moveBebop(0, 7, 0)
    drone.wait(2)

    # look for the target and arrive there
    arrive()

    # Accomodate the drone before taking picures
    accomodate()

    # Take the Areal pictures
    AerealFootage()

    # Return
    comeBack()

    # finish all the operations
    drone.land()
    sys.exit(0)


''' Detects when the prgram has been interrupted'''
def signal_handler(signal, frame):
    print('You pressed Ctrl+C!')
    drone.land()
    sys.exit(0)
```

```
if __name__ == "__main__":
        print "Battery:", drone.battery
        if (drone.battery > 2) :
        main()
        else :
        print "Battery too low"
```

C2 MergeImages.py

```
# Merging images horizontally.

# This code is based upon the source code: min2bro, 12 July 2017 . [Internet]. Available:
https://kanoki.org/2017/07/12/merge-images-with-python/. [Accessed: 14 March 2018]


import numpy as np
import cv2
import sys
import os
import operator


from PIL import Image
from PIL import ImageDraw



# Load the images
DroneImages = ['UAV_first3.jpg', 'UAV_second3.jpg', 'UAV_third2.jpg']
DroneImg = [ Image.open(i) for i in DroneImages ]

# Resize the images to match the size of the smallest image
MinDroneImageShape = sorted( [(np.sum(i.size), i.size ) for i in DroneImg])[0][1]

# Merge the images horizontally
DroneImageMerge = np.hstack( (np.asarray(
i.resize(MinDroneImageShape,Image.ANTIALIAS) ) for i in DroneImg) )

# Creating an image memory
DroneImageMerge = Image.fromarray(DroneImageMerge)

# Save the image
DroneImageMerge.save( 'MergeImages.jpg' )

# Plot the image
DroneImageMerge.show()
```

C3 Overlapping.py

```
# Stitching images together

# This code is based upon the source code: Shawn Gomez, 19 June 2014. [Internet]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4081273/. [Accessed: 15 March 2018]

# To run the part of the code that finds the matching key points, comment out everything after plt.show(). To run the merging of image1 and image2, comment out plt.show() and everything after
##############################################################################
# Repeat the process and merge 'Overlapping.jpg' with the third photo.
##############################################################################


from skimage import data
from skimage import transform as tf
from skimage.feature import (ORB, match_descriptors, corner_harris, corner_peaks, plot_matches)
from skimage.io import imread
from skimage.measure import ransac
from skimage.transform import ProjectiveTransform
from skimage.color import rgb2gray
from skimage.io import imsave, show
from skimage.color import gray2rgb
from skimage.exposure import rescale_intensity
from skimage.transform import warp
from skimage.transform import SimilarityTransform
import matplotlib.pyplot as plt
import numpy as np
import cv2

from PIL import Image

import matplotlib


# Read the images
DroneImage1 = imread('UAV_first3.jpg')
DroneImage2 = imread('UAV_second3.jpg')


# Convert the image to gray scale
DroneImage1= rgb2gray(DroneImage1)
DroneImage2= rgb2gray(DroneImage2)
```

```
# Define the number of key points and the threshold
orb = ORB(n_keypoints=1000, fast_threshold=0.5 )

# Find the key points in each picture
orb.detect_and_extract(DroneImage1)
keypointsDroneImage1 = orb.keypoints
descriptorsDroneImage1 = orb.descriptors

orb.detect_and_extract(DroneImage2)
keypointsDroneImage2 = orb.keypoints
descriptorsDroneImage2 = orb.descriptors


# Find the matching key point between the two images
matchesImage12 = match_descriptors(descriptorsDroneImage1, descriptorsDroneImage2,
cross_check=True)

# Plot the images with key points
fig, ax = plt.subplots(nrows=2, ncols=1)
plt.gray()


plot_matches(ax[0], DroneImage1, DroneImage2, keypointsDroneImage1,
keypointsDroneImage2, matchesImage12)
ax[0].axis('off')
ax[0].set_title("Left vs. Center")

plot_matches(ax[1],  DroneImage1, DroneImage2, keypointsDroneImage1,
keypointsDroneImage2, matchesImage12)
ax[1].axis('off')
ax[1].set_title("Left vs. Center")


plt.show()


src = keypointsDroneImage2[matchesImage12[:, 1]][:, ::-1 ]
dst = keypointsDroneImage1[matchesImage12[:, 0]][:, ::-1 ]

# Estimate the transformation model
model_robust, inliers = \
    ransac((src, dst), ProjectiveTransform,
           min_samples=4, residual_threshold=2)

# Determine the shape of the merged image
```

```
r, c = DroneImage2.shape[:2]
corners1 = np.array([[0, 0], [0, r], [c, 0], [c, r]])

DeformCorners = model_robust(corners1)

# Merge the arrays vertically
CombCorners = np.vstack((DeformCorners, corners1))

MinCorner = np.min(CombCorners, axis = 0)
MaxCorner = np.max(CombCorners, axis = 0)

output_shape = (MaxCorner - MinCorner)
output_shape = np.ceil(output_shape[::-1])

offset = SimilarityTransform(translation=-MinCorner)

# Deform the image to the same size as the output_shape

DeformDroneImage1 = warp(DroneImage1, offset.inverse, output_shape=output_shape,
cval=-1)
DeformDroneImage2 = warp(DroneImage2, (model_robust + offset).inverse,

output_shape=output_shape, cval=-1)

# Add alpha to the find the average number of images

MaskDroneImage1 = (DeformDroneImage1 != -1 )
DeformDroneImage1[~MaskDroneImage1] = 0
AlphaDroneImage1 = np.dstack((gray2rgb(DeformDroneImage1), MaskDroneImage1))

MaskDroneImage2 = (DeformDroneImage2 != -1 )
DeformDroneImage2[~MaskDroneImage2] = 0
AlphaDroneImage2 = np.dstack((gray2rgb(DeformDroneImage2), MaskDroneImage2))

# Merge the images together
Merged12 = (AlphaDroneImage1 + AlphaDroneImage2)
Alpha = Merged12[..., 3]
Merged12 /= np.maximum(Alpha, 1)[..., np.newaxis]

# Save the image
matplotlib.image.imsave('Overlapping.jpg', Merged12)



##########################################################################
# Repeat the process and merge 'Overlapping.jpg' with the third photo.
```

```
##########################################################################

# Read the images
DroneImage3 = imread('Overlapping.jpg')
DroneImage4 = imread('UAV_third2.jpg')


# Convert the image to gray scale
DroneImage3= rgb2gray(DroneImage3)
DroneImage4= rgb2gray(DroneImage4)

# Define the number of key points and the threshold
orb = ORB(n_keypoints=1000, fast_threshold=0.5 )

# Find the key points in each picture
orb.detect_and_extract(DroneImage3)
keypointsDroneImage3 = orb.keypoints
descriptorsDroneImage3 = orb.descriptors

orb.detect_and_extract(DroneImage4)
keypointsDroneImage4 = orb.keypoints
descriptorsDroneImage4 = orb.descriptors


# Find the matching key point between the two images
matchesImage34 = match_descriptors(descriptorsDroneImage3, descriptorsDroneImage4,
cross_check=True)

# Plot the images with key points
fig, ax = plt.subplots(nrows=2, ncols=1)
plt.gray()


plot_matches(ax[0], DroneImage3, DroneImage4, keypointsDroneImage3,
keypointsDroneImage4, matchesImage34)
ax[0].axis('off')
ax[0].set_title("Left vs. Center")

plot_matches(ax[1],  DroneImage3, DroneImage4, keypointsDroneImage3,
keypointsDroneImage4, matchesImage34)
ax[1].axis('off')
ax[1].set_title("Left vs. Center")


#plt.show()
```

```
src = keypointsDroneImage4[matchesImage34[:, 1]][:, ::-1 ]
dst = keypointsDroneImage3[matchesImage34[:, 0]][:, ::-1 ]

# Estimate the transformation model
model_robust, inliers = \
    ransac((src, dst), ProjectiveTransform,
           min_samples=4, residual_threshold=2)

# Determine the shape of the merged image
r, c = DroneImage4.shape[:2]
corners3 = np.array([[0, 0], [0, r], [c, 0], [c, r]])

DeformCorners3 = model_robust(corners3)

# Merge the arrays vertically
CombCorners3 = np.vstack((DeformCorners3, corners3))

MinCorner3 = np.min(CombCorners3, axis = 0)
MaxCorner3 = np.max(CombCorners3, axis = 0)

output_shape = (MaxCorner3 - MinCorner3)
output_shape = np.ceil(output_shape[::-1])

offset3 = SimilarityTransform(translation=-MinCorner3)

# Deform the image to the same size as the output_shape

DeformDroneImage3 = warp(DroneImage3, offset.inverse, output_shape=output_shape,
cval=-1)
DeformDroneImage4 = warp(DroneImage4, (model_robust + offset).inverse,

output_shape=output_shape, cval=-1)

# Add alpha to the find the average number of images

MaskDroneImage3 = (DeformDroneImage3 != -1 )
DeformDroneImage3[~MaskDroneImage3] = 0
AlphaDroneImage3 = np.dstack((gray2rgb(DeformDroneImage3), MaskDroneImage3))

MaskDroneImage4 = (DeformDroneImage4 != -1 )
DeformDroneImage4[~MaskDroneImage4] = 0
AlphaDroneImage4 = np.dstack((gray2rgb(DeformDroneImage4), MaskDroneImage4))

# Merge the images together
```

```
Merged34 = (AlphaDroneImage3 + AlphaDroneImage4)
Alpha = Merged34[..., 3]
Merged34 /= np.maximum(Alpha, 1)[..., np.newaxis]

# Save the image
matplotlib.image.imsave('Overlapping1.jpg', Merged34)
```

C4 detect_color.py

```
# import the necessary packages
import time
import cv2
import numpy as np
from PIL import Image

''' Program that receives an image and detects the coordantes of an specifiec color object '''

def coordenates(name):
    # Read the image
    image = cv2.imread(name)

    # Blur the image
    blur = cv2.blur(image, (3,3))

    # Select the lower and upper boundaries
    lower = np.array([76,31,4],dtype="uint8")
    upper = np.array([210,90,70], dtype="uint8")

    # Apply the threhold to the blurred image
    thresh = cv2.inRange(blur, lower, upper)
    thresh2 = thresh.copy()

    # find contours in the threshold image
    image, contours,hierarchy =
cv2.findContours(thresh,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    # finding contour with maximum area and store it as best_cnt
    max_area = 0
    best_cnt = 1
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > max_area:
                                                    max_area = area
                                                    best_cnt = cnt

    # finding centroids of best_cnt and draw a circle there
    M = cv2.moments(best_cnt)
```

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

# Modify the values to show them
im = Image.open(name)
width, height = im.size #size of our image
distance= (cx-width/2)

#print(distance)# if positive the object is on the right side of the image
b=((height/2) - cy)

if((height/2) >= cy):
x=1 # object in the upper part of the image
else:
x=0 #object in the lower part of the image

return distance,x,b
```

C5 What_was_found.py

```
# import the necessary packages
import time
import cv2
import numpy as np
from PIL import Image

''' Program that receives an image and detects the coordantes of an specifiec color object,
printing the dots of the center of the image and the detected target '''

def coordenates(name):
    # Read the image
    image = cv2.imread(name)

    # Blur the image
    blur = cv2.blur(image, (3,3))

    # Select the lower and upper boundaries
    lower = np.array([76,31,4],dtype="uint8")
    upper = np.array([210,90,70], dtype="uint8")

    # Apply the threhold to the blurred image
    thresh = cv2.inRange(blur, lower, upper)
    thresh2 = thresh.copy()

    # find contours in the threshold image
    image, contours,hierarchy =
cv2.findContours(thresh,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    # finding contour with maximum area and store it as best_cnt
    max_area = 0
    best_cnt = 1
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > max_area:
                                                    max_area = area
                                                    best_cnt = cnt

    # finding centroids of best_cnt and draw a circle there
    M = cv2.moments(best_cnt)
    cx,cy = int(M['m10']/M['m00']), int(M['m01']/M['m00'])

    # Modify the values to show them
    im = Image.open(name)
    width, height = im.size #size of our image
```

```
        distance= (cx-width/2)

        #print(distance)# if positive the object is on the right side of the image
        b=((height/2) - cy)

        if((height/2) >= cy):
        x=1 # object in the upper part of the image
        else:
        x=0 #object in the lower part of the image

        # Print the obtained values
        print cx,cy
        print distance, b

        # put the circles in the image
        cv2.circle(blur,(cx,cy),10,(0,0,255),-1)
        cv2.circle(blur,(width/2,height/2),10,(0,0,0),-1)

        # show the frame
        cv2.imshow("Frame", blur)
        key = cv2.waitKey(0) & 0xFF

def main():
        coordenates("Photo_13.jpg")
        ''' Show the results graphically'''



if __name__ == "__main__":
        main()
```

C6 saveme.py

```
#!/usr/bin/python

import sys
import cv2
import os
import time
import signal

from bebop import Bebop
from commands import movePCMDCmd
from capdet import detectTwoColors, loadColors
from bebop import *
# this will be in new separate repository as common library fo robotika Python-powered robots
from apyros.metalog import MetaLog, disableAsserts
from apyros.manual import myKbhit, ManualControlException

import time


drone = Bebop()
drone.land()
sys.exit(0)
```