CPS Tello Stalking System Project Report

Abdelmoniem Abdelkhalek, Ahmed Eltayyeb, Klaudia Pazdzierz,
Paulina Pazdzierz, Mariia Kashirina, Tiago Giraldez

{ge87baq, ge94cos, ge48yep, ge48yog, ge48yex, ge92lur}@tum.de

Abstract

This project involves the development of a computer vision system for tracking and following the object based on its color using Tello drone. The system uses *OpenCV* for processing the images from the Tello's video stream and *djitellopy* library for controlling the movements of the drone. The goal of the project is to create a simple vision system that effectively tracks the given object.

1. Introduction

Drones are very important for many fields of our development as they can go to places that are hard for people to be reached. Lots of dangerous or time-consuming tasks can be replaced by drones, so tracking objects is a key ability to allow drones to search, rescue, watch and film the environment.

The main goals of this project are:

1. Connect to Tello drone and access its camera feed.
2. Process the life video from the drone to detect the objects based on their color.
3. Control the drone's movements in order to keep the object in the center of the camera view.

By using *OpenCV* library, the image is processed and the color chosen by configuring the HSV trackbars is detected. Then, the system will detect the contours with given thresholds and minimum area needed for it to become an object for the system. Then, the center coordinates are calculated, and depending on them, the system makes a decision about further drone movements. The system also has a grid which describes the boundaries of an object's position.

2. Hardware and Software Requirements

2.1. Hardware

The Tello drone is a small quadcopter that features a Vision Positioning System and an onboard camera. The Maximum Image Size is 2592x1936 and it records in 1280x720 30p Mode. Tello is powered by an Intel Movidius Myriad 2 VPU (Vision Processing Unit) and uses Wi-Fi (802.11n) for communication with controllers.

2.2. Software

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
Developers can use Python to interact with the Tello SDK which is provided by Ryze Technology and has the library *djitellopy* that offers various commands and interface for controlling the drone's flight.

```
from djitellopy import Tello
```

Listing 1: Importing djitellopy Library

OpenCV (Open Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms. It is used to change the color schemes, to create simple interfaces, perform edge and contour detection, etc.
NumPy is the fundamental package for scientific computing in Python.

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

Listing 2: Importing NumPy and OpenCV Libraries

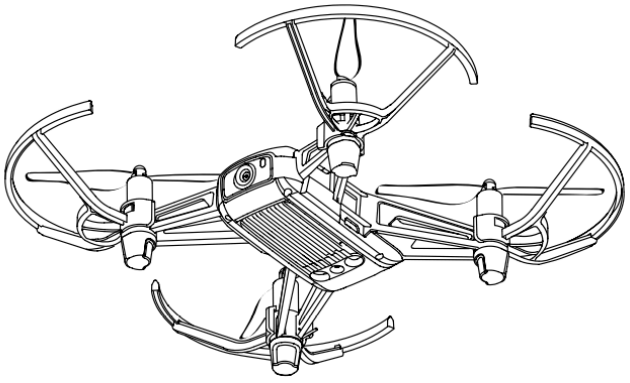

Figure 1: The Tello Drone [1]

3. Methodology and Implementation

The implementation and corresponding methodology will be shown part by part going through the following steps:

1. Connecting to the Tello drone.
2. HSV and Creating Trackbars.
3. Image Processing.
4. Contour Detection.
5. Drone Movement.

3.1 Connection with Tello Drone

Tello connects to the controller using WiFi and to send the signal after the connection, we need to initialize the drone itself in the program. The initial velocities and speed are set to 0. Fly is a boolean parameter which helps in debugging: when testing the program, we first need to configure color without any need for drone to move. Also, the stream is turned on and the camera starts working

```
fly = 1                      #0 - stream only, 1 - fly
tello = Tello()              #initializing the drone
tello.connect()              #connecting to the drone
tello.for_back_velocity = 0
tello.left_right_velocity = 0
tello.up_down_velocity = 0
tello.yaw_velocity = 0
tello.speed = 0
tello.streamon()
```

Listing 3: Initializing the Tello Drone

3.2 HSV and Creation of the Trackbars

HSV is a color model that describes colors in terms of Hue, Saturation and Value. It is very intuitive for human perception and is usually used in tasks that are connected with filtering and object segmentation. Hue is the basis of the color, it is usually measured in degrees. Saturation shows how intensive the color is, measured in percentages, where 0% is totally gray and 100% is the most saturated color. Value is also called Brightness and is also measured in percentages, where 0% is black (no light) and 100% is the most bright appearance of color.

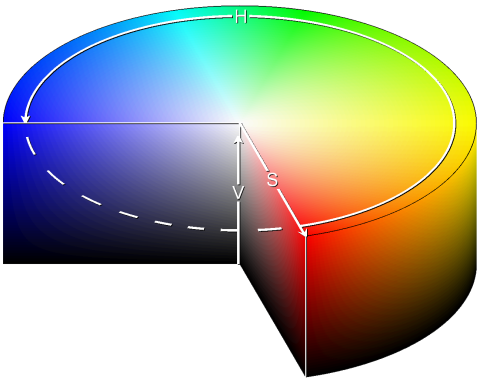

Figure 2: HSV Cylinder [2]

In OpenCV, the ranges for HSV parameters are different. Hue differs from 0 to 180 and Saturation with Value are in range of 256. We create a set of trackbars that will represent this ranges to help the drone detect the objects. Also, the trackbars for Contour Thresholds are implemented and their parameters will be explained later in this report. Code sample shows the creation of Saturation range with default parameters (167, 255) set to detect the specific orange object.

```
cv2.createTrackbar("Saturation Min","HSV parameters",167,255,empty)
cv2.createTrackbar("Saturation Max","HSV parameters",255,255,empty)
```

Listing 4: Creating the Trackbars using OpenCV

Now, having the controls, we can look in details how image processing is happening.

Figure 3: Trackbars Window

3.3 Image Processing

First, we need to get the frame from the stream and resize it to adjust with the screen size. Then, after reading the current trackbars parameters and converting the BGR (default Tello's color mode) to HSV, we create a mask. This mask identifies pixels within a specified color

range. When applying a mask to the image, we allow only the parts of the image that match the mask to be shown. This helps in isolating the object from the rest of the image. Then, the image is blurred with a special matrix to reduce noise and improve the accuracy of edge detection.

```
mask = cv2.inRange(imgHsv,lowerBound,upperBound)
    result = cv2.bitwise_and(img,img, mask = mask)
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    imgBlur = cv2.GaussianBlur(result, (7, 7), 1) #blur the masked image
    imgGray = cv2.cvtColor(imgBlur, cv2.COLOR_BGR2GRAY)
```

Listing 5: Creation of the Mask

Before detecting the contours, we will use Canny. It performs edge detection on the image, which is useful for detecting contours.

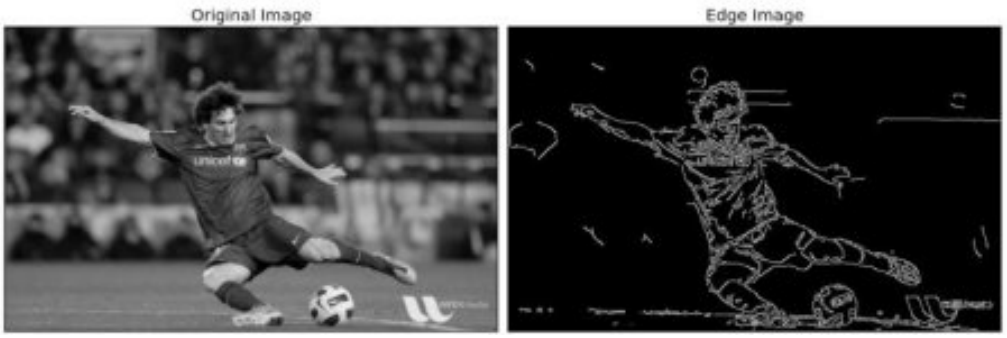

Figure 4: Canny Edge Detection Example [3]

It uses two thresholds (*t1* and *t2*) that will classify the edges into different groups depending on their gradient intensity. Pixels with gradient lower than *t1* are considered as non edges. It helps in reducing the number of false edges by ignoring lower gradient values that are likely to be noise. Pixels with gradient greater than *t2* are considered as strong edges and immediately taken as a part of a final edge. It ensures that only the most significant edges are detected initially. Pixels with gradient values between *t1*and *t2* are classified as weak edges. These may or may not be part of an edge. If they are connected to strong edges, they are considered part of the edge. Otherwise, they are discarded. By adjusting these thresholds, you can control the sensitivity of the Canny edge detector.

After identifying the edges, we use dilation to make the future contours more pronounced. It works by convolving an image with a kernel (a small matrix) and replaces each pixel value with the maximum value of its neighbors defined by the kernel. This process causes bright regions within the image to "grow" or "dilate". So, if before the edges are thin and broken, after the dilation they become thicker and more continuous.

```
imgCanny = cv2.Canny(imgGray, t1, t2)
kernel = np.ones((5, 5))
imgDil = cv2.dilate(imgCanny, kernel, iterations=1)
```

Listing 6: Edge Detection and Dilation

After this step, all manual work is done, all parameters are fixed and the final step for the algorithm is contour detection.

3.4. Contour Detection

The dilated image is fed as an input for the function which will retrieve only the outermost contours, ignoring any nested contours. This is useful if you are only interested in the outer boundary of objects. If there is no chain approximation, the program stores all the contour points. This can be useful for applications requiring precise contour shapes but may use more memory. If memory usage is a concern, *CHAIN_APPROX_SIMPLE* can be used instead to save memory by compressing horizontal, vertical, and diagonal segments into their endpoints. Now, after detecting all possible contours, the program will calculate their areas (a scalar value representing the number of pixels inside the contour) and will work only with those, which are bigger than the fixed value given by configuration of Area Trackbar.

```
contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    for contour in contours:
        area = cv2.contourArea(contour)
```

Listing 7: Contour Detection and Filtering

Contours are further highlighted on the screen both by their original shape and bounding rectangle.

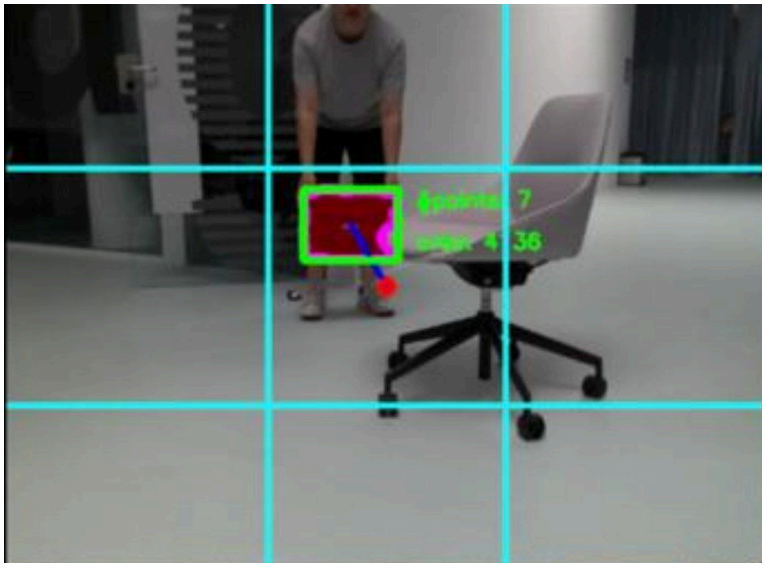

Figure 5: Contour Demonstration on the Frame

Finally, we calculate the center of the boundary rectangle and can send this value to the drone to perform the movements.

3.5. Movement Detection

In this project, the drone will perform yaw and up & down movements. We define an invisible dead zone and if the absolute value between the object's center and drone's camera center is greater than this parameter, we need to make decisions according to the drone's movement. In case of going out to the left, the drone *direction* signal will be generated as 1 andthen will activate the movement to the left.

```
if (cx < halfW - outVis):
                cv2.putText(imgContour, "LEFT", (30, 50), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 0, 0), 4)
                cv2.rectangle(imgContour, (0,halfH-outVis), (halfW-outVis,halfH+outVis),
(255, 0, 0), cv2.FILLED)
                direction = 1
```

Listing 8: Indication of the Left Movement

From the user's perspective, the grid is created and whenever the object's center crosses the lines of the center square, the respective neighbor rectangle is filled in with color and the text about the further movement of the drone appears. Also, each detected object is present with the number of points in its contour and area on the screen.

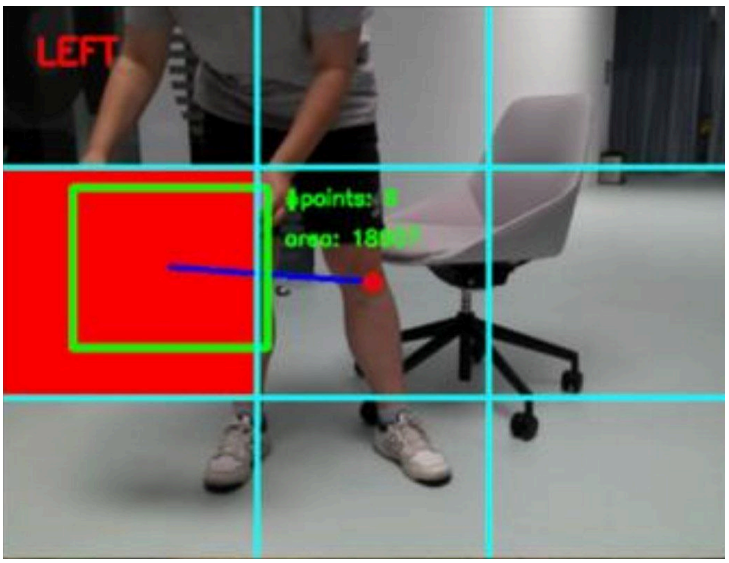

Figure 6: Movement to the Left Interface

The velocities are changed by 50 (or -50) in case of any movement and then sent to the drone. After the user finishes the test of the program, key q is pressed in order to land the drone and destroy all created windows.

```
if dir == 1:
       tello.yaw_velocity = -50
...
if tello.send_rc_control:
       tello.send_rc_control(tello.left_right_velocity,
tello.for_back_velocity, tello.up_down_velocity, tello.yaw_velocity)
```

Listing 9: Turning Left

Finally, the original, the “detected”, color mask and dilation images are stacked in order to show the user how the image processing and contour detection are performed and making the manual configuration of trackbars possible.

Conclusion and Results

The system successfully tracks and follows the object based on its color. The drone can move left, right, up, and down depending on the object's position in the frame. The trackbars allow for real-time adjustment of HSV and dilation values to accurately detect the object. For future work, we think about implementing forward and backward movements. It can be done by fixing concrete objects and their area. Then, when it is reduced, the drone needs to go forward and vice versa. The current problem is in a very sensitive colors configuration and that the program reacts very differently to any change of the environment light.

This project wasn't just about making the drone follow an object. It also helped us learn about important tools and techniques in image processing and computer vision. We saw how small adjustments to settings like the HSV values and Canny thresholds could make a big difference in how well the drone could track the object.

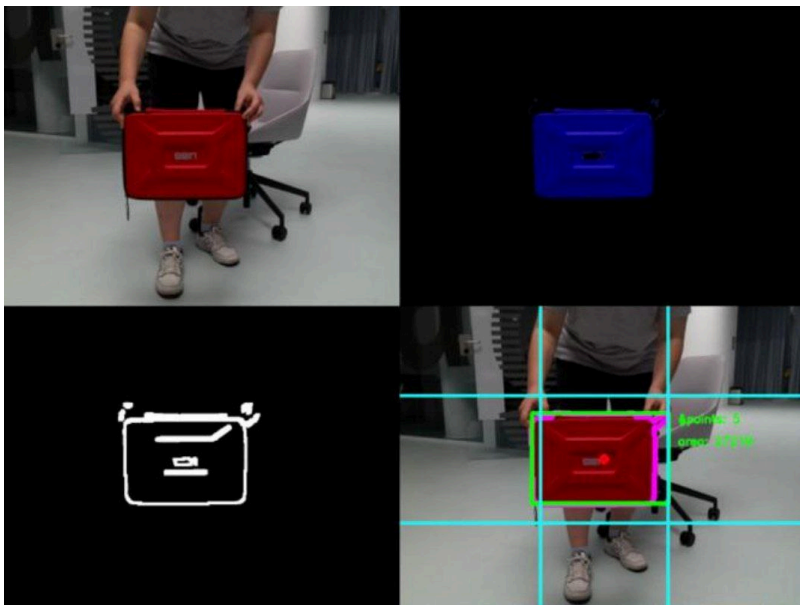

Figure 7: Final User Interface

References

1. Tello Guide
2. OpenCV Library
3. NumPy Basics.
4. Tello commands overview