Intelligent UAV Technology for Visual Inspection

Submitted as part of the requirement for:

CE903/CE913 Group Project

Group 10 members:

Alba Mendoza

Vilde Olsen

Lorenzo Mirante

Enrique Zaragoza

Horacio Ruiz

Supervisors:

Manoj P Thakur

Dongbing Gu

Date:

March 22, 2018.

Abstract: This document shows the design and development process of an autonomous UAV system used for visual inspection. The results and evaluations obtained for the system are presented as well. The code make the drone capable of flying to an specific location, indicated by a marker through the use of color recognition, and take pictures of the location. Afterwards, the images are merged together giving the user a bigger view of the area around and of the marker.

Keywords: UAV, drone, visual inspection, object recognition, autonomous flying, merge images, target location, Bebop.

Table of Contents

1. Introduction

1.1. Problem Description

Robotic systems can be considered one of the most important industries nowadays. Many economic areas, like agriculture, manufacture and health have been benefited from intelligent systems. Adaptability in this systems is key to solve tasks efficiently. Recent development of autonomous robots have open a gate of opportunities by facilitating processes where the human resource is difficult to implement. An example of this autonomous systems are the Unmanned Aerial Vehicles (UAV); this emerging technology represents an opportunity to efficiently tackle tasks that requires agility and speed.

The lightweight UAV systems can be implemented on increasingly important tasks for visual recognition by inspecting hard-to-reach areas. This could provide an overview footage in critical situations or only for data collection. This project aim is to design an autonomous UAV system where an on-board camera is used to provide information about the surface conditions of a certain area. The visual localization was made through the camera and provided the system with the information to strategically follow a path to reach a marker placed in the interest area. Additionally, an online image processor was built to work on the pictures taken by the drone.

1.2. Tools

An off-the-shelf drone known as Parrot Bebop 1 was used to design the system. The drone has one build-in camera that helped the system to recognize the area by finding a previous placed marker. The system work through the implementation of a repository called KATARINA and python language [6]. Some python libraries that are used implicitly in the codes are: os, cv2, signal, numpy, and matplotlib. In order to evaluate the system, multiple tests were conducted where the drone had to search for the marker and safely fly to the area to take pictures and then return to the starting point.

1.3. Methodology

As this project implies a software development with multiple feedback and testing stages implied; it was necessary to follow a structure where changes during the working process were allowed; therefore, this work was planned under the structure of the Agile Scrum Process methodology.

1.4. Objective

This document presents a description of process and results obtained in the elaboration of the autonomous UAV system. Other important aspects such as limitations of the project and features of the designed system are further discussed in this paper.

2. System Design

This section will discuss the requirements followed to develop the intelligent UAV system. The system architecture will be described, enlisting the different components. The hardware used, and the software created as well as their interaction are explained.

2.1. Methodology

The Agile Scrum Process was the methodology chosen for the project management structure due to the benefit that it brings for software development. The Agile Process mainly encourages the communication and collaboration between the teammates and the customer (supervisor); allowing to easily track continuous tasks such as feedback, tests, planning, improvements [12]. The ongoing changes in prioritization in this method contributes to have versatile project. The Agile Process structure is suitable for projects of software developing with short working period.

Scrum is the chosen Agile Process method for this project. This gives the framework for developing, delivering and sustaining complex products [12]. Following the structure of this method for the Scrum Team; the product owner was the supervisor of the project; the project group was the Development Team and the Scrum Master was Vilde M. G. Olsen. The Scrum methodology provide the project with multiple events like meetings between the product owner, Scrum Master and Development Team. The recurrent events used for this project were the meetings. The meetings were used for sprint planning, sprint review and retrospective. Even though a sprint is a working time-frame usually of a month, the duration had to be trim to fit the working period. A detailed description of the followed sprints through the project development is further discussed.

2.2. Requirements

The requirements necessary for the development of the project were exhaustively defined in the "Intelligent UAV Technology for Visual Inspection Software Requirements Specification" document (SRS document). It describes the scope, specifies the software and hardware requirements, and defines the different types of interfaces needed. For further information on these, the SRS document should be consulted [3].

The requirements for the system can be broadly summarized as follows: The drone should be able to perform autonomous flying, finding the marker, and obtaining digital images of target area. It may optionally perform object avoidance. The system should also be able of merging digital images.

2.3. Sprint 1 - Project Start

The first sprint consisted on the identification of the specific system requirements and to decide what type of tests that will be performed to fulfill the system requirements. The next table (see table 1) shows the parts of the sprints, the task holders, status and comments.

Task	Task Holder	Status
Requirement Specification	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 08.02.18
Project Plan	Vilde	Done by 08.02.18
Test Specification	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 08.02.18
Sprint Retrospective	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 08.02.18

Meeting		

Table 1. Sprint 1.

2.4.Sprint 2 - Autonomous flying

The second sprint (table 2), is focused in make the drone fly autonomously and deal with the connection problem that the drone might have.

Task	Task Holder	Status
Sprint Planning Meeting	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 09.02.18
Take off, landing, taking pictures manually	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 14.02.18
Autonomous flying	Enrique, Horacio	Done by 22.02.18
Connection Problem	Lorenzo, Alba, Vilde	Done by 22.02.18
Testing	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 22.02.18
Sprint Retrospective Meeting	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 2.02.18

Table 2. Sprint 2.

2.4.1. Implementation

2.4.1.1. Take off and landing

The main requirement to cover in the first sprint was to make the drone take off and land in a safely way. These actions were aimed to be commanded through the computer. Since the programming language used for this project was Python; for the creation of the code that enables the autonomous flying, python 2.7 was used. This allowed the implementation of functions from open source libraries available for the Bebop 1.

The Parrot Bebop Drone V.1 was the main component for this sprint. The camera and the 2.4 GHz band connection were key for starting the implementation with the computer. The PC used Ubuntu 17.10 as the main Operative System (OS) using the terminal as the main tool to run the code for testing.

The steps followed for writing the code for the Taking off/Landing function are as follows. Firstly, in order to use the functions available from the open source Katarina library, an object had to be created. The object `drone` was created and assigned as an object of `Bebop`.

```
drone = Bebop()
```

To make the drone take off, the function with the same name was used.

```
drone. takeoff()
```

Once in the air, different altitudes that the drone could achieve were tested. For that reason, the functions flyToAltitude() and the function wait() to make intervals between the different altitudes.

```
drone.flyToAltitude(1)
drone.wait(2)
drone.flyToAltitude(1.5)
drone.wait(2)
drone.flyToAltitude(2)
drone.wait(2)
drone.flyToAltitude(1)
drone.wait(2)
```

And finally, to make the drone land, the function called `land()` was used.

```
drone.land()
```

From the library Katarina, the main function used is the one given in `bebop.py`, named `bebop.` This file contains the main instructions to control the drone used by the user, including different update status variables of the drone.

This file uses `commands.py` to perform the different actions of the drone. `Commands.py` uses some commands from `ARDrone3` [13], converting them to packages. At the same time, `commands.py` uses `nvdata.py` which sends, converts and creates the packages, and logs files, and send them to the drone, which is also based in `ARDrone2` [13].

2.4.1.2. Autonomous flying

The autonomous flying routine is the basic function for the system. The motion of the drone is set by three different values known as Principal Axis or Rotation Axis. The pitch, roll and yaw are the values responsible for the lateral axis, longitudinal axis and the vertical axis respectively. A routine was created to control these values in the drone. The function was created inside the file `bebop.py`, and it was named `moveBebop` which receive the attributes for the roll, the pitch and the yaw. The function was created as follows:

```
def moveBebop (self, aRoll, aPitch, aYaw):
        self.update( cmd=movePCMDCmd( active=True,
         roll=aRoll, pitch=aPitch, yaw=aYaw, gaz=0 ))
```

The values that are set in the function is the percentage of the full capacity for each value, which means that the variable `aPitch` takes the value of 10, it means that it is the 10% of the full speed that the drone is capable to provide to move forward.

For this reason, if it was wanted by the user to move to drone forward, the `aPitch` value had to be modified, while the others should be kept at zero. Nevertheless, this does not mean that the function does not allow to make different movements at the same time, but as an individual test the following function was used to make the drone fly forward (Pitch) at its 10% of capacity for 5 seconds.

```
drone.moveBebop(0, 10, 0)
drone.wait(5)
```

It also should be mention, that in order for this function to work, the drone had to take off before.

The longitudinal axis (roll) is the responsible for rotating the drone from nose to tail. To roll the drone to the right for 5 seconds at 15% of its full potential, the next instructions were used.

```
drone.moveBebop(15, 0, 0)
drone.wait(5)
```

Finally, the rotation in the vertical axis makes the drone move the nose side to side. To do a similar test for the rotation of the drone in its own axis (Yaw) with 15% of its full capacity, the code used was the presented as follows:

```
drone.moveBebop(0, 0, 15)
drone.wait(5)
```

It must be said that two observations were done. The first one is that before landing, it was better to make the drone hover to control the unsteadily movement while landing. To do so, a function, with the same name, present in bebop was used.

```
drone.hover()
```

Or, the previous function that was created could be used, setting all the values to zero.

```
drone.moveBebop(0, 0, 0)
```

The second observation is that because the movements generated by the code does not generate any type of automatic compensation as the official app does; manually this type of compensations were done to make the drone more stable.

For example, after moving forward, even if the drone was set to hover, the momentum generated by the last function makes it continue to move forward for some moments. As an example, if the drone had a movement of pitch positive, a smaller negative movement should be used to keep the drone more stable after finishing the positive pitch movement. This is better explained in the next lines of code.

```
drone.moveBebop(0, 10, 0)
drone.wait(5)
drone.moveBebop(0, -3, 0)
drone.wait(1)
drone.hover()
```

Something to be mentioned is that it was also observed that this drone presented different behaviours according to the room or environment the drone was flying in (even when used with the official app). In the room that the drone was tested, it presented a constant backward movement (Pitch). To make the drone hover more stable, the function `moveBebop` was used, to compensate, as follows.

```
drone.moveBebop(0, 1, 0)
drone.wait(1)
```

2.4.1.3. Taking pictures

To take pictures, the were two options used: the function inside bebop for taking pictures called `takePicture()`, or use individual frames of video transmission from the drone.

First, the function takePicture() was tested, but two characteristics of this function that made it not useful for the project.

The first one is that the image obtained from the camera has a format not useful for the type analysis that was wanted to be made.

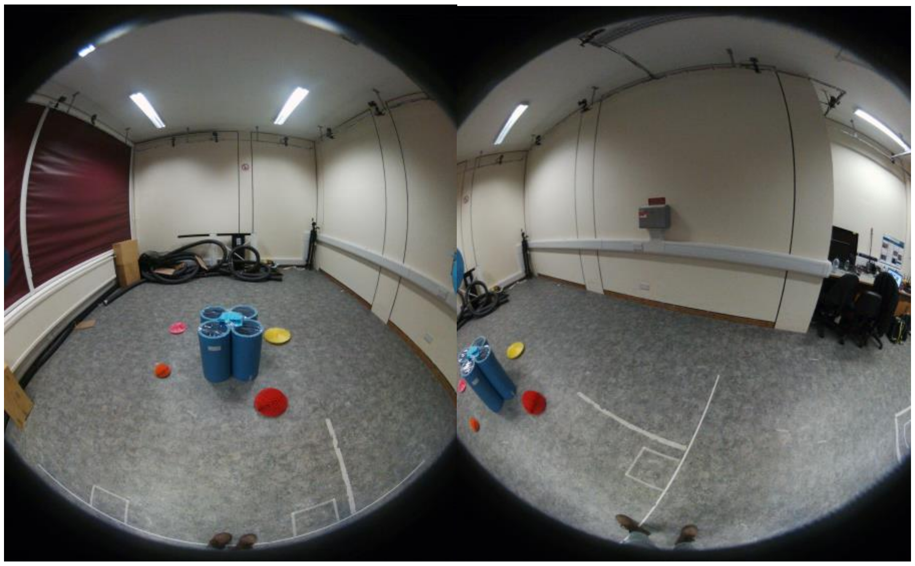


Figure 1A, 1B. Drone pictures.

The image was not processed, and a semi-distorted photo was obtained, mostly because of the type of camera lens that the drone has; which is a fisheye wide lens, see Figure 1 a and b.

The computation of the distance between the drone's position and the marker became a difficult task since the unwanted movement in the drone affects the position of the marker in the picture. Even though the marker still is in the same place, the drone's point of view changed frequently. This was solved by using the tilt of the camera as a reference point for the distance between the two objects.

The second reason for not using that function is that it stored the photos taken into the drone, which made it more difficult to be able to obtain the photo. Either the drone had to return and be connected to the computer with a physical cable or a different program had to be used to connect to the internal storage of the drone; using its IP address.

For that reason, the option to use individual frames from video transmission was used. The quality of it is not as high as the photos, but it was enough to do image analysis and allow the drone to fly to the marker.

To enable video, different functions had to be used:

Firstly, the video signal had to be established.

```
signal.signal(signal.SIGINT, signal_handler)
```

Then the camera needed to be adjusted to a tilt and a pan angle, and the video to be enabled.

```
drone.moveCamera(tilt=tiltvalue,                                  pan=0)
drone.videoEnable()
```

Then, a variable called cap received the frame taken from the drone, and lastly it was processed.

```
cap = cv2.VideoCapture('./bebop.sdp')
ret, img = cap.read()
```

Further, if the frame is wanted to be shown on the screen, using commands of CV2 can be used. At this moment, `img` contains the frame that could be used for image analyzing.

```
cv2.imshow('img',                                                 img)
cv2.namedWindow('img', cv2.WINDOW_NORMAL)
cv2.waitKey(1)
```

The image obtained from the camera usually have some noise in them, but it is not as significant to make the marker not visible, such as the one that is shown in Figure 2.

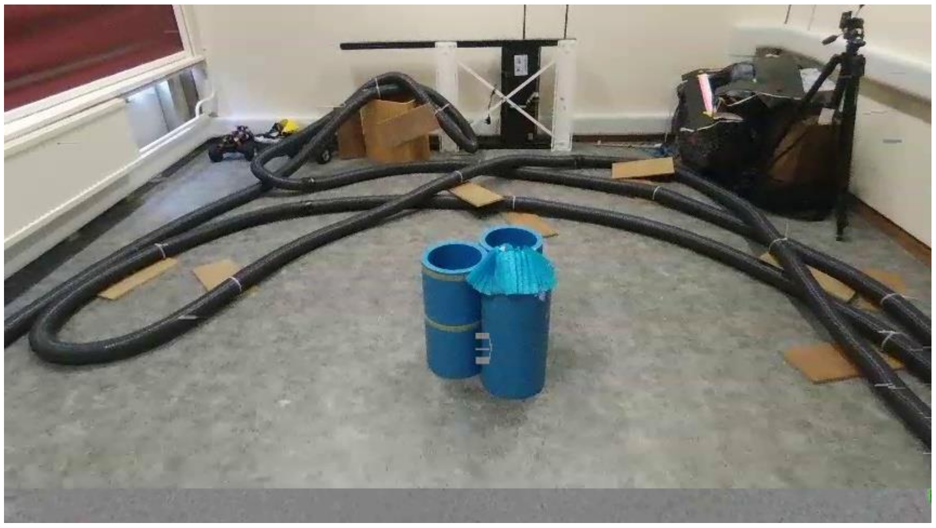


Figure 2. High resolution picture.

Nevertheless, some frames with a lot of noise could be obtained, such as the following:

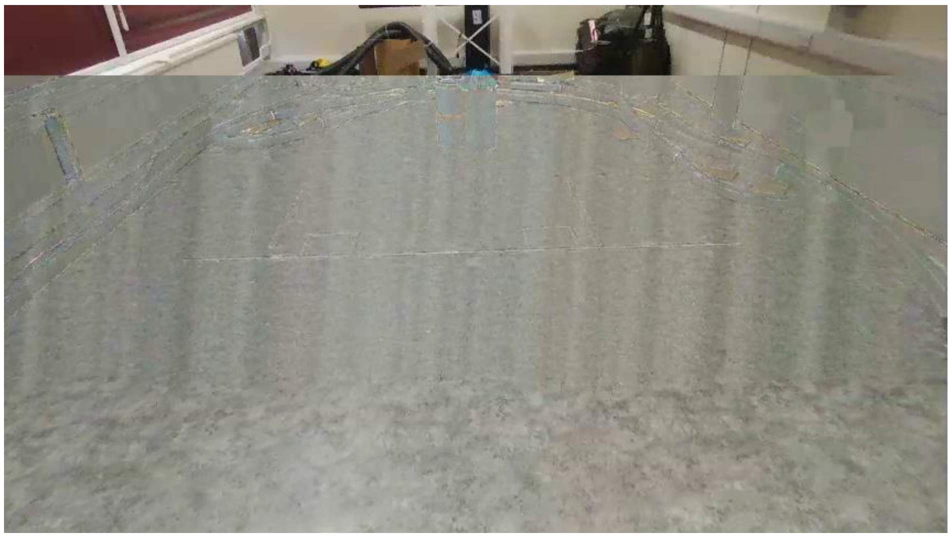


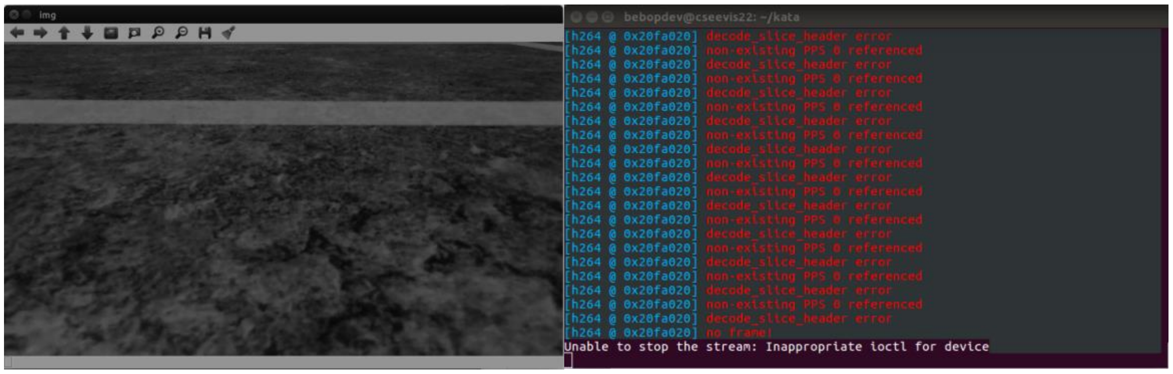
Figure 3. Low resolution picture.

For that reason, at least 20 frames were obtained each time, where it was usually frame 11 and above that had less amount of noise; they were used for image analysis.

2.4.1.4. Connection Problem

Due to the function of the `nvdata.py` the drone is controlled using packages. Thus, when there is a poor connection some packages might be lost. Usually the code and the system works with no problems when some packages are lost, but, when a large amount of them are not received properly the system starts to malfunction. For instance, when this problem occurs the program crashes making the drone to either do the last command it was performing or disconnects and hovers by itself.

For this reason, the program called `saveme.py` was created. The main function of this algorithm is to reconnect the drone and make it land using the previous mentioned functions. An example of a disconnection scenario is illustrated in Figure 4.



```
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] non-existing PPS 0 referenced
[h264 @ 0x20fa020] decode_slice_header error
[h264 @ 0x20fa020] no frame!
Unable to stop the stream: Inappropriate ioctl for device
```

Figure 4. Error picture.

2.4.2. <u>Testing</u>

2.4.2.1. Test T.1

As it was planned in the SRS document [3], the first test was addressed to evaluate the basic tasks in the system: taking off, landing, moving and the connection problems. Multiples tests were made in this sprint even though it was stated that only one test was needed. However at the end of this sprint, a complete test was conducted to evaluate the acceptance criteria in the first test from the SRS document (see Table 3) [3].

Test ID	**T.1**
Requirements Tested	**3.2.1.3.2; 3.2.2.3.1; 3.5.1**
Test Description	**Autonomous take off, hover and land.**
Resources	**Drone, computer**
Test steps	**1. Connect to the drone** **2. Take off vertically** **3. Flying three meters in any direction** **4. Land vertically**
Acceptance Criteria	**· The drone takes off vertically.** **· The drone stops rising vertically when it has reached 1.5 meters.** **· Move horizontally.** **· Hove at the same location** **· Land the drone safely and vertically.**

Table 3. Test 1.

After discussing the obtained results in sprint 2, it can be said that the drone was capable to take off vertically at a certain height. However, the battery performance affect the altitude of the drone although the same height was set. The horizontal move could be achieved through the created function, as well as the hover feature. Safely landing was also achieved as the drone can land whether by itself, or with a security stop.

2.5.Sprint 3 - Image processing

The third sprint (table 3.) include the work made for the object recognition function where the drone has to localize a marker.

Task	Task Holder	Status
Sprint Planning Meeting	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 22.02.18
Object Recognition	Enrique, Alba, Lorenzo	Done by 01.03.18
Image Stitching	Vilde, Horacio	Done by 21.03.18
Object recognition test	Lorenzo, Alba, Horacio	Done by 07.03.18
Image Stitching test	Enrique, Horacio, Vilde	Done by 21.03.18
Sprint Retrospective Meeting	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 21.03.18

Table 3. Sprint 3.

2.5.1. <u>Implementation</u>

2.5.1.1. Object Recognition

A crucial part of this project is the drone being able to locate the target in the environment. To perform this task, the camera, located in the frontal part of the drone, is used to capture video while the drone is flying. Frames of this video are taken as images to be analyzed to find the target. Below are the different approaches to locate the target in the image, and the approach that gave the best result are described.

The first code used for this task was collected from [4]. This uses the library presented in GitHub repository, in which some images obtained from the drone were used for testing.

This code reads the image, applies a 3x3 mask to it, and then converts the image from RGB to HSV. Further, a threshold was used to detect only a certain range of HSV values of an specified colour. This threshold would ensure that the algorithm was able to ignore other colours that was not the one specified.

After some testing, the code proved to work on specific cases, but when in use mid-flight, it did not provide an accurate target recognition. It sometimes detected a different object as a target or provide the message that the target was not present in the frame when it was.

The weaknesses that this code had were due to a bad conversion from RGB to HSV; therefore, it was decided to use a different code or approach to achieve the desired target detection.

The second algorithm tested is presented in `detect_color.py`, which was based on [5]. This code works similarly to the previous code, applying a 3x3 mask and using range values of a color to detect the target. Using these modifications, the noise inside the image was reduce and the target was found. The coordinates of the target were given as a return value, using the width and height of the image to create a cardinal axis, and then measured from the center of it.

This code proved to be more efficient in the target detection, mostly due to not converting RGB to HSV. Nevertheless, even though the detection of the target was quickly done, when the marker was not present in the image, most of the times the algorithm tried to find the most similar color of the one specified to be the target and return the coordinates of it; which could translate as a high sensitivity of it.

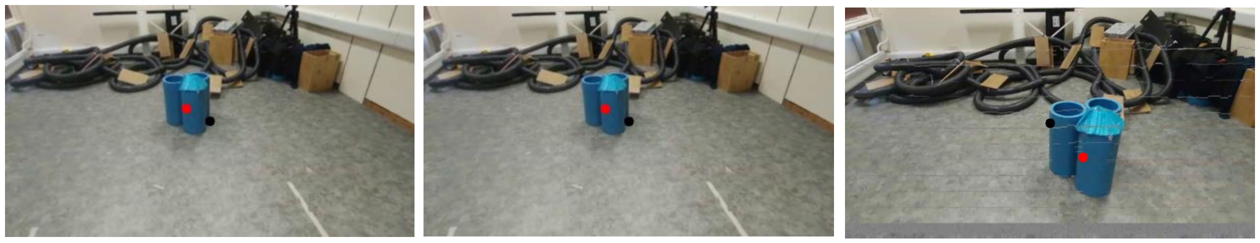


Figure 5. Object recognition

The purpose of this code was to provide coordinates of the target to the drone. This way the drone could choose to which side to rotate in order to center the target in the image. Centering the target in the image was made using a specified range in the code.

To have a visual representation of the centre of the cardinal axis generated and the location of the detected target, a third code was created. This code, called `What_was_found.py` uses the code provided in `detect_color.py` and modifies the image. The modifications it does to the image are: print a black dot at the center of the image, and print a red dot as the detected target. The creation of this code was made in order to have a visual representation of the results of the identification of the target, without modifying the color detection. In addition, `detect_color.py` does not display the image because that will take up processing power and time, resulting in a bad performance of the done.

Some examples of the results of `What_was_found.py` using sample images are the presented in Figure 5.

2.5.1.2. Image Merging

As specified in the requirements the drone will take more than two pictures of the market. The drone takes 24 pictures of the market. The reason for the large number of photos is because of the quality of the pictures. After multiple tests, it was realised that the quality of the image would vary from image to image. To assure that the images that would be merged together would have the best quality, it was necessary to take 24 pictures. The pictures would be placed into three categories; First, Second and Third. The “First” category contains eight images of the left side of the target, the “Second” category contains eight images of the center of the target, and the “third” category contains eight image of the right side of the target. The drone would position itself over the target, take eight pictures, turn 45° to the right, take another eight photos, turn another 45°, then take the last eight photos. One photo with the best quality from each category where chosen to be merged together.

To merge the images into one image, Python was used. The first algorithm that were used to perform this task is found in Appendix C2, named `MergeImages.py`. The source for this code was found at [1]. The code is a simple version of merging images, by merging them horizontally without stitching them together. The code uses Python Image Library to process the images. The images are loaded into the code. The images are then used in the function `sorted` to resize the images to match the smallest image of them.

```
MinDroneImageShape = sorted( [(np.sum(i.size), i.size ) for i in DroneImg])[0][1]
```

The resized images are then horizontally merged using hstack a numpy function.

```
DroneImageMerge = np.hstack( (np.asarray( i.resize(MinDroneImageShape,Image.ANTIALIAS) ) for i in DroneImg) )
```

`DroneImageMerge` is further used in Image.fromarray to create an image memory.

```
DroneImageMerge = Image.fromarray(DroneImageMerge)
```

This image memory represents the final merged photo and is illustrated in Figure 6.

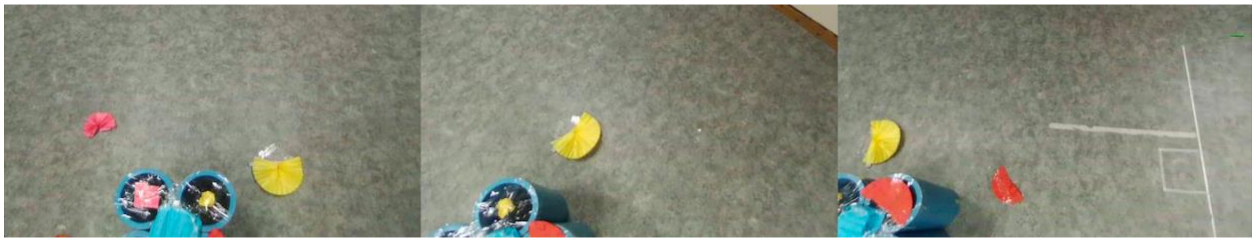


Figure 6. Horizontally merged pictures.

It was further wished that the three images would be stitched together, overlapping the parts of the images that matched. To do this a more complicated algorithm had to be written. The source for this algorithm can be found in [2]. The algorithm performs a stitching of images using scikit which is a tool for data mining and data analysis. The algorithm can be found in Appendix C3, named Overlapping.py. First, the necessary packages are imported from scikit. The first and second image are stitch together. Once that processes is done, the stitched image is matched and stitched with the right image, see Figure 7A, B and C for the images.

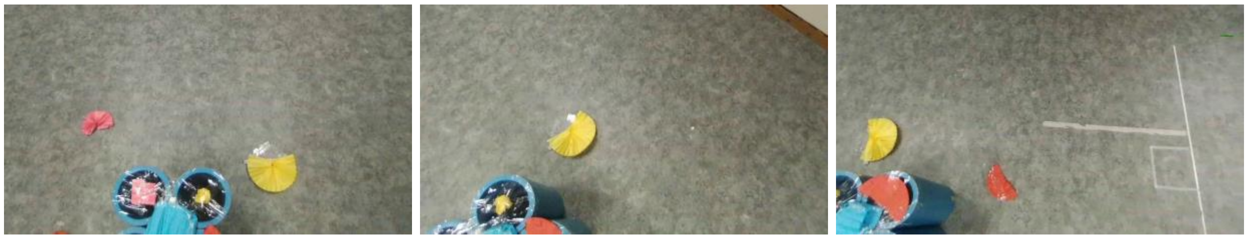


Figure 7A, 7B, 7C. Input pictures for Overlapping.py.

The first process of the algorithm is to read and grayscale the first and second image. The algorithm further use ORB (Oriented FAST and rotated BRIEF) to define the number of key point that are to be find in the images, find the key points in each photo and match them to each other.

```
orb = ORB(n_keypoints=1000, fast_threshold=0.5 )

orb.detect_and_extract(DroneImage1)
keypointsDroneImage1 = orb.keypoints
descriptorsDroneImage1 = orb.descriptors

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

```
        orb.detect_and_extract(image2)
        keypoints2 = orb.keypoints
        descriptors2 = orb.descriptors

        matchesImage12=match_descriptors(descriptorsDroneImage
    1, descriptorsDroneImage2, cross_check=True)
```

The images with their respected key point and matched key points are illustrated in Figure 8.

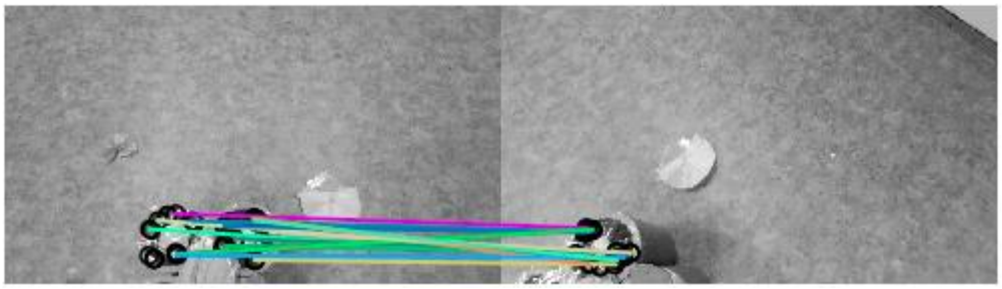


Figure 8: Matched key points from the first and second image.

The next step in the algorithm is to estimate the transformation model. This is done by using the RANSAC (RANdom SAmple Consensus) function. The function choose the model that most resembles with the matches[2].

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

model_robust, inliers = \ransac((src, dst), ProjectiveTransform,
min_samples=4, residual_threshold=2)
```

Further, the panorama picture is created. The first step of this process is to determine the shape of the stitched image. To do this, the shape of second image is found, where the output is the number of rows and columns in the image.

```
r, c = DroneImage2.shape[:2]
```

The rows and the columns are used to create an array of the corner of the final image. The corners are then used to deform the image corners to correspond to the new corners. The `corners1` and the `DeformCorners` acts as images. They are therefore further transferred into the function `np.vstack`. The function merges the two images vertically. The merged image is then used to find the output shape of the final image by subtracting the minimum of the merged image array of the maximum of the merged image array.

```
corners1 = np.array([[0, 0], [0, r], [c, 0], [c, r]])

DeformCorners = model_robust(corners1)

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])
```

The output shapes are then used to deform the images.

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

Further, an alpha is added to the deformed images to inform the average number of images in the stitched image.

```
Merged12 /= np.maximum(Alpha, 1)[..., np.newaxis]
```

Lastly, to save the stitched image, the matplot library is used.

```
matplotlib.image.imsave('Overlapping.jpg', Merged12)
```

The stitched image is illustrated in Figure 9. This process is further repeated to stitch the already stitched image with the right image.

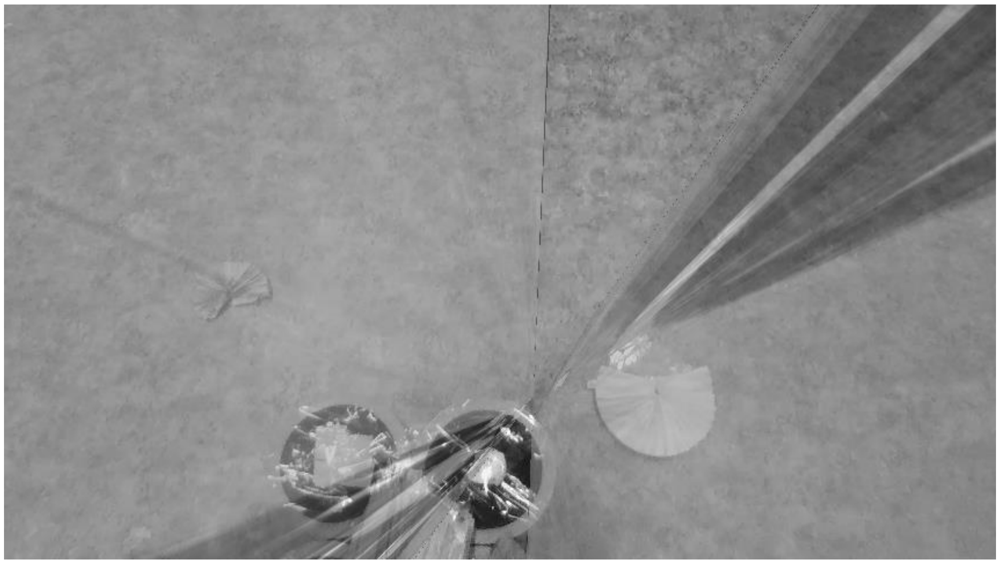


Figure 9: Result of stitching the first and the second image together.

2.5.2. <u>Testing</u>

2.5.2.1. Test T.2

This table is taken from the SRS document. It describes a test overview, test steps and acceptance criteria for finding the marker function. The main objective of this test was to evaluate the image recognition function. Before running the function for test on the hovering drone, it was tested with drone placed in the ground. The code was used several times to try to find different colors. It was find that the color blue was the most effective target to identify. In the following table (Table 4), the object recognition on the flying drone is described.

Test ID	**T.2**
Requirements Tested	**3.3.2; 3.2.3.3.1; 3.2.3.3.2;**
Test Description	**Obtain images from the camera and process them to locate the marker.**
Resources	**Camera, drone, computer**
Test steps	**1. Send the start instruction** **2. Take off vertically** **3. Put the marker close to the drone** **4. See the messages shown in the terminal** **5. Move the marker to different locations** **6. Repeat until the tester is satisfied**
Acceptance Criteria	· **The drone locates the correct marker**

Table 4. Sprint 4.

After trying with markers with different colors, it was seen that the function worked better with a blue marker. The steps were followed as it was written in the SRS document [3]. Nevertheless, since the room does not have enough space; the drone starting position was changed instead of the location of the marker. The results were satisfactory as the drone correctly detect the color. The problem relied that sometimes the color was detected in some other place in the room (i.e. a shadow) messing the functionality of the drone.

2.5.2.2. Image Stitching Test

As we can see in Figure 9, the quality of the image and the stitching of the two images together is not good. The reason for this is the input images. The drone is not able to take images horizontally because of its instability. Therefor, when taking the pictures the drone will turn when taking the second and third photo, leaving the photos rotated compared to the first photo. Beneath is an example of how the algorithm would have performed had the images used been taken perfectly horizontally with a slightly shift. Figure 10a and 10b illustrates the two images that are being stitched together. The images are one image split into two with a slightly overlapping part.

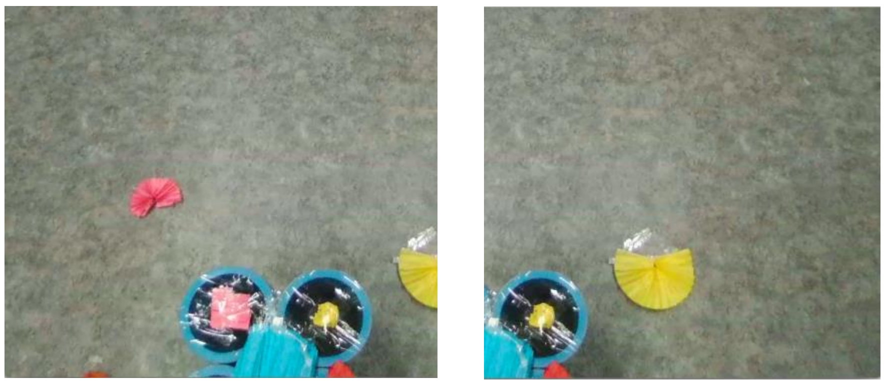


Figure 10a and 10b: Images used to test the algorithm performance

The result of the merging of the two images are illustrated in Figure 11. The images are perfectly stitched together, proving that the code is working.

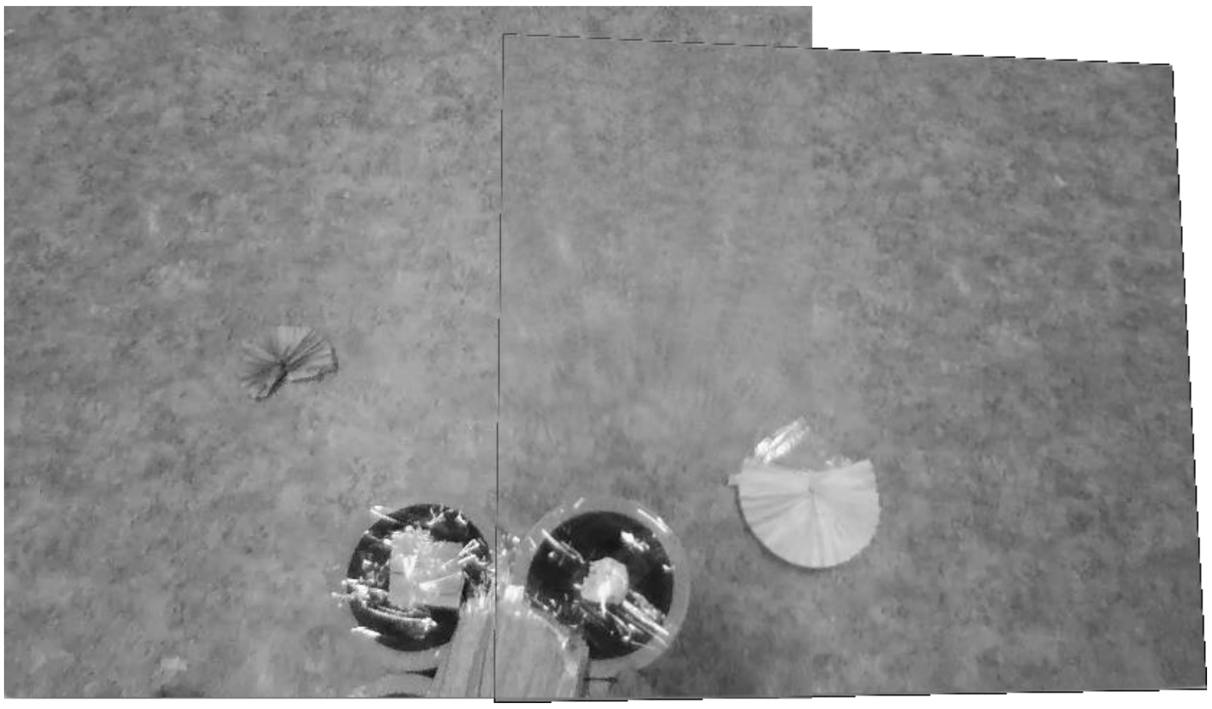


Figure 11: The result of merging the two example images together.

Because of the bad quality of the stitched image in Figure 11, it was not possible to stitch it with the third photo. The algorithm could not find any features in the stitched image of the first and second photo. Do to this complication, it was not possible to merge all the three images together. In addition, the merged image of the first and second photo (Figure 11) is of very bad quality and does not provide the user of the system with accurate information of the area around and at the marker. To provided the user with the information needed, it would be advised to use ImageMerge.py to merge the three images together.

2.6.Sprint 4 - Combination of all functions

This additional sprint was to combine all the function as one system. There were multiple tests where the system parameters were changed for better response to the different scenarios. The implementation and results are presented in the following sections.

Task	Task Holder	Status
Sprint Planning Meeting	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 21.03.18
Drone testing	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 22.03.18
Sprint Review Meeting	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 22.03.18
Sprint Retrospective Meeting	Enrique, Horacio, Alba, Lorenzo, Vilde	Done by 22.03.18

Table 5. Sprint 5.

2.7. Features of design

The architecture of the system is structured from different functions. These functions uses resources for different components of the hardware and software. The main components used for the main system are the camera, the drone's propellers, battery and software resources. The main function can briefly be described as follows.

After running the code, the system starts by creating a folder to store the images and set the camera by enabling the signal. Here the camera component and the Wi-Fi connections are being used. An initial photo is taken to check the connectivity. Further, the drone will take off and fly to an altitude of 1.8 meters. While the drone is taking off a couple of functions are applied to handle the movement that the drone slightly has.

Once the drone is in the air, it hovers for two seconds before the object recognitions function takes control to find the marker. Once the marker is identified, the drone position is allocated to face towards the marker and it begins moving forward.

When the drone has reached the marker, it descents to 1.5 m to take the pictures of the area while it is rotating. After the pictures of the area were taken the drone is ready to go return to the starting point by using the same routine to search for the marker, as the starting point has a similar marker. After retrieving the pictures, a different function is used to merged the photos to have a better look of the area.

3. Implementation

3.1. General structure

The code is divided in different sections. Each one has an specific purpose and together support the UAV autonomous system. The sections can be listed as following.

- The algorithm creates a folder to save the future taken photos.
- The camera is initialize to enable video streaming.
- The drone takes off and flies to a specified height.
- The drone localize the target and flies towards its position.
- Once the drone is on the target, it accommodates itself so the target is centered in the photos.
- The drone starts to rotate while it takes three pictures of the area.
- The drone fly back by using the same function to find the second marker.

- Once the drone has reached the return marker, it lands.

3.2. Detailed functions

To create the folder, the code uses the function called `createFolder()`.

This function first creates the path for the folder, which is called “Photos”.

```
file_path = "/home/bebopdev/kata/Photos/"
directory = os.path.dirname(file_path)
```

Then, it checks if it exists, so it can create it.

```
if not os.path.exists(directory):
    os.makedirs(directory)
```

To enable the video, the previous functions tested were used.

```
signal.signal(signal.SIGINT, signal_handler)
drone.moveCamera(tilt= -30, pan=0)
drone.videoEnable()
testCamera()
```

In this case, the function `testCamera()` only to activate the video streaming before the drone deploys.

To takeoff, and fly to a certain altitude, the functions that were tested before were used. The drone was deployed to a height of 1.8 meters.

```
drone.takeoff()
drone.flyToAltitude(1.8)
drone.moveBebop(0, 5, 0)
drone.flyToAltitude(1.8)
drone.moveBebop(0, 7, 0)
drone.wait(2)
```

At the same time, the compensations that were mentioned before are also present here. In this case, most of the times the drone took off, it started drifting backwards, a small positive pitch value was given.

To make the drone find the target and fly to it, the function `arrive()` was used. The structure of this function is as follows:

- A Boolean variable, called `there`, was established as false.

```
there = False
```

- The initial value of the tilt value of the camera is set as -30.

```
angleCamera = -30
```

- A loop that repeats until `there` is not true, was created.
- Inside the loop:
    - The function `findTarget()` is called, which return the value of x and y of the target, as well as it modifies the tilt value of the camera.

```
 NewangeleCamera,x,y = findTarget(angleCamera)
angleCamera = NewangeleCamera
```

    - The new value of tilt is applied to the camera.

```
 drone.moveCamera(tilt=angleCamera, pan=0)
```

    - According to the tilt value of the camera, the speed that drone must have is obtained. This is done using the function `obtainSpeed()`.

```
speed,contrarest,there =obtainSpeed(angleCamera)
```

    - After that, the drone moves with the pitch value obtained for 1 second and the loop starts again.

```
drone.moveBebop(Roll, speed, 0)
drone.wait(1)
drone.moveBebop(0, contrarest, 0)
drone.wait(1)
```

❑ The function `findtarget()` mainly makes the drone rotate(Yaw) until the target is centered the frame it obtains, returning the x and y values of the target, as well as a Boolean value to define if the target was centered after 20 indentations :

- ❑ First the function `takePict()` is called, which returns a boolean value specifying if the target is centered (According to range of value specified) and the x and y value of the target.

```
centred, xvalue, yvalue = takePict()
```

- ❑ If the target is not centered, the x value is taken into consideration.
- ❑ If the x value is higher than 0, the drone starts to make a positive yaw movement, else, a negative yaw movement is done. At the same time, the values to compensate the unwanted movement are chosen.

```
            if xvalue < 0:
                YawAngle = -16
                RollAngle = -7
            else :
                YawAngle = 16
                RollAngle = 7
```

- ❑ At the same time, the y value indicates the pitch value that the drone has to make, which is done to compensate the instability of its hover state.

```
          if yvalue > 0:
                pitch = 11
            else:
                pitch = 5
```

- ❑ The movements set to the drone.

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

- ❑ Once the target is finally centered, the new angle for the tilt of the camera is obtained according to the y value.

```
    if yvalue < -30:
        if angleCamera > -80:
            angleCamera = angleCamera - 12
```

- ❑ The function `takePict()` does a loop of 20 iterations, obtaining frames of the video streaming.
  - ❑ 20 frames are taken, and they are analysed from the 11th because that is usually when the level of noise is lower in the images.
  - ❑ The frame is analysed using the previous code in `detect_colour.py`, obtaining the x and y value of the target.

```
distancex,x,distancey=detect.coordenates(imageName)
```

  - ❑ The function obtains the absolute value of the x value, checks if it is lower than 80 to indicate if the target is centered.

```
 distancex2 = abs(distancex)
     if distancex2 < 80:
                  #The target is centred
```

  - ❑ If the the target is centered, the loop stops and return the boolean values, as well as the two coordinates.

```
return True, distancex, distancey
```

  - ❑ If the 20th iteration ends, it means that it was never centered, and the return values are given.

```
return False, distancex, distancey
```

After the function `arrive()` is finished, the drone should be right on top of the target; nevertheless, due to the uncertainty of its movements, some accommodations previous to takes photos can be done. For that reason, the function `accomodate()` is done. The function works as follows:

- The function previously mentioned, takePict(), was used.

```
centred, xvalue, yvalue = takePict()
```

- If the target is not centred, the x value is taken in consideration. If the target is greater than zero a positive yaw movement is done, otherwise a negative is given.

```
if xvalue < 0:
            YawAngle = -15
            RollAngle = -7
```

```
        else :
            YawAngle = 15
            RollAngle = 7
```

- At the same time, the Y value is taken in consideration and a pitch value given

```
if yvalue > 0:
            pitch = 9
        else:
            pitch = 5
```

- Then the drone moves according to it.

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

- Finally, the drone descends to 1.5 meters

```
drone.hover()
drone.flyToAltitude(1.5)
```

Once the drone has accommodated, the aerial pictures are taken. for this reason the function `AerialFootage()` is used. The function works as follows:

- Firstly, the camera is tilted to values of -90.

```
drone.moveCamera(tilt= -90, pan=0)
```

- The drone hovers to gain some stability

```
    drone.moveBebop(-1, 0, 0)
    drone.moveBebop(1, 0, 0)
    drone.moveBebop(0, 1, 0)
```

- The function `UAVPict()` is called, and the first set of photos are taken.
- Then, values for roll, pitch and yaw are given, so the drone rotates on its own axis with 45 degrees approximately.

```
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)
```

- Then, the second set of pictures are taken using `UAVPict()`.
- Finally, the drone is set to rotate another 45 degrees.

```
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)
```

- And the third set of photos are taken using `UAVPict()`.

❑ The function `UAVPict()` works really similarly as `takePict()`, but instead, it saves the images in the folder created by `createFolder()` with the name it receives as parameter. In the same way as `takePict()`, it uses 20 frames from which the last 8 are saved.

Finally, once the photos are taken, the drone has to return to its original position. To do this, the function `comeBack()` is used. These functions basically prepares the drone to use a similar function to `arrive()`, because its original position is delimited by another marker. The function works as follows:

- Tilt the camera to a value of -30

```
drone.moveCamera(tilt= -30, pan=0)
```

- Give values to yaw, pitch and roll, so the drone rotates on its own axis

```
 YawAngle = 25
    RollAngle = -17
    pitch = 15
    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 the drone fly to the altitude of 1.8 m, as well as do the movements to compensate the unwanted drifting,

```
drone.flyToAltitude(1.8)
drone.moveBebop(-5, 10, 0)
drone.wait(2)
drone.moveBebop(-2, 5, 0)
drone.wait(2)
```

- And, finally when the drone is in a stable position, make it return. To do so, the function `arrive2()` is called. This functions works in the exact same way as `arrive()` but it uses different values of yaw and pitch because the compensation for the movement of the drone is different; mostly because of the area it is in and the state of its battery.

4. Results and evaluation

Looking at the project plan in the SRS documentation already presented for this project, it is possible to understand, that the divide and conquer approach has been used. For this reason it was possible to develop different parts of the main system and test them independently, separated from each other. Correspondingly the test phase was divided in different parts.

Each one of these parts is focussed on one particular task that at the start only contemplate small actions from the drone and then arrive to sum all of these in the final task.

As it was mentioned before, multiple tests were made to test the system. Whether an individual function or the complete system, the evaluation was based on the response of the drone facing different starting positions. The testing phase has as main objective to ensure that the functions were working well together. The autonomous flying had to have smooth transitions when the camera was looking for a target.

The used markers were blue cylinders to use the color and shape contrast from the room. Furthermore, different color markers made of paper were placed around the cylinders to work as point of reference for the image processing tests. As illustrated in Figure 12A and 12B below, different scenarios were set to test the UAV system.

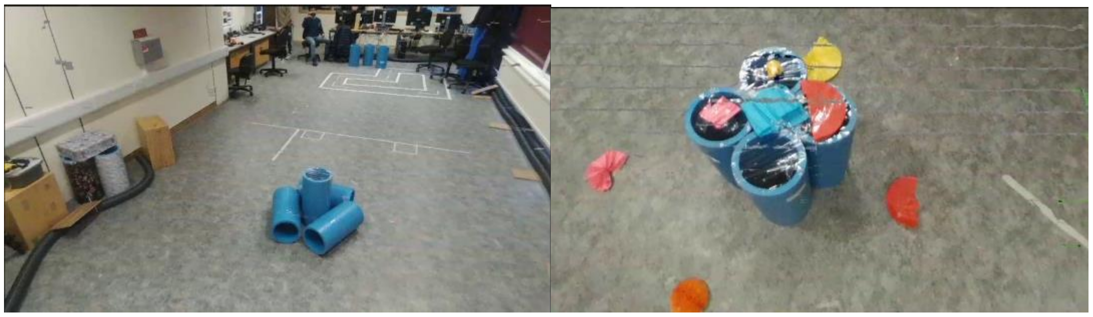


Figure 12A and B. Scenarios used to test the system.

It was noticed that the drone finds it complicated to hover above (or near) to the cylinders as the air expelled from the drone propellers when it entered the holes, it pushed the UAV and making the system unstable. To solve this, tapes was used to cover the cylinder holes.

It is important to denote that the test results went different because of the hardware conditions. The used Bebop has notable damages as it was used in previous projects. Additionally, the batteries were used and the power supply was different for each test.

Another problem for the test was being able to know the depth of the object in the image. “We know that images from multiple cameras can in principle be processed to determine the locations of objects in 3D space because that is what the human visual system does.”[Computer Vision (CE316 and CE866):Vision in a 3D World, Adrian F.

Clark]. As known there must be used at least two cameras to be able to have a 3D scheme of the objects in the image and understand the distance of the object from where the images are taken, but in this case only one camera is available.

To overcome this problem different information has been extracted from the image. As already mentioned in the chapter about the target recognition, one of the most important features that has been calculated is the distance of the object from the center of the image. Using the width and the height of the image it is possible to determine the distance of the object from the center of the image. This information resulted to be really useful to create a system that can modify the drone movements, and tilt its camera, "to keep it flying" in the direction of the target.

4.1. Testing strategy

The test phase has been executed in different steps during the development of the project. The testing has been divided to evaluate the different routines of the system, and that each performs the intended task. With this specific purpose the system was divided into Autonomous flying and Drone's Target Recognition. The testing strategies are presented below:

4.1.1. Black box testing

- Autonomous flying: This routine contains Drone's target recognition routine, but they are considered separately.

Input - Command line on terminal

Output - Aerial Photos, the drone takes off and lands by itself.

- Drone's target recognition: This routine is contained on Autonomous flying routine, but are tested separately.

Input - Video Feed from the drone

Output - Image with target localized and centered.

- Image merging:

Input - Three photos

Output - One merged image

4.1.2. <u>White Box Testing</u>

For the white box testing, path testing was implemented. The program flow graph for each routine is detailed below, which present the control decisions that can be taken in each different code.

- Autonomous flying: Originally this routine was going to be tested by T1 in SRS document [3]. However, once the development was in process it was realized this task would contain other routines and could be tested further in depth. As a consequence the test evolved and its steps increased. This is illustrated in the autonomous flying flow graph in figure 13.

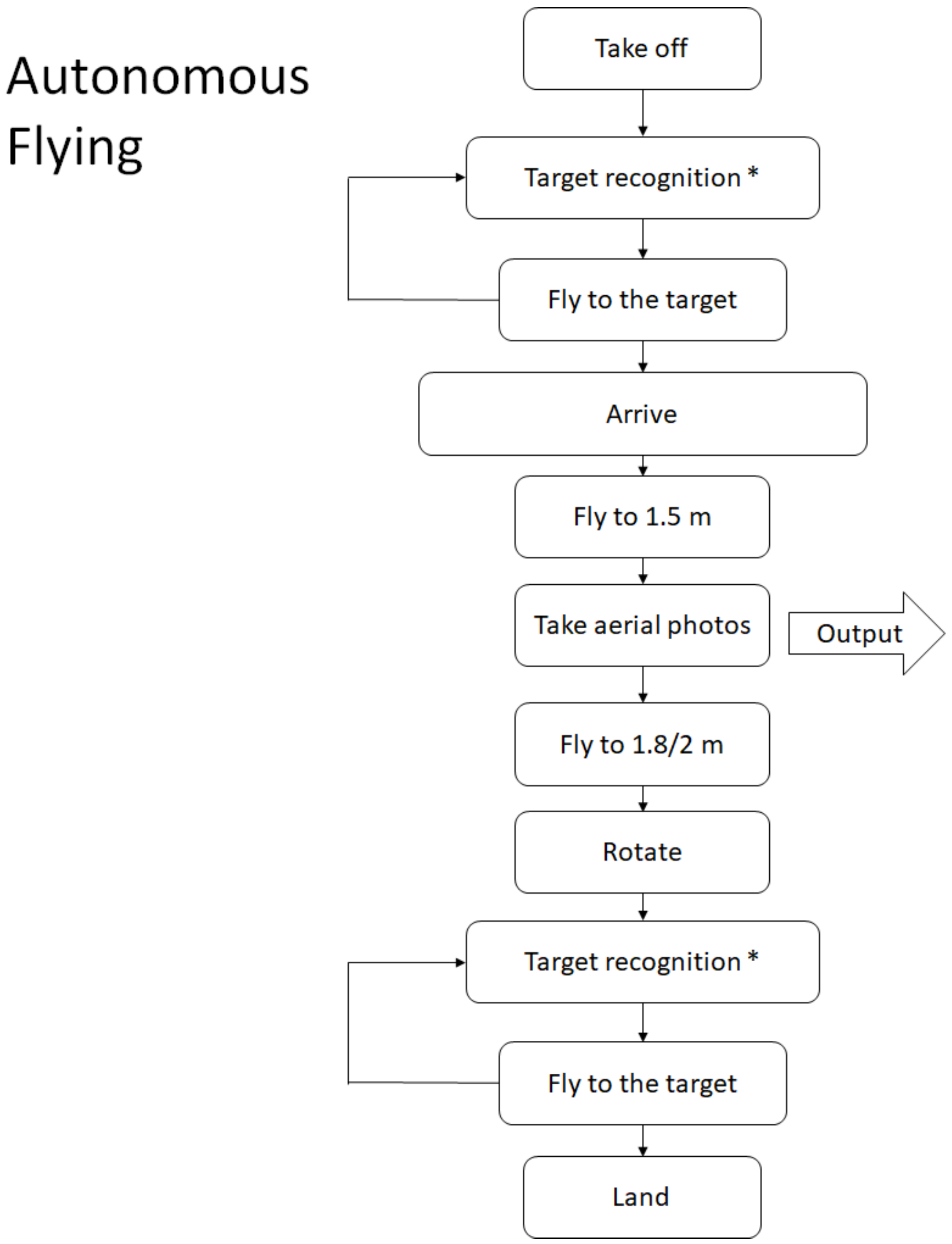


Figure 13. Autonomous flying flowchart.

- Drone's target recognition: This routine was tested by T2 from SRS document [3]. It tested the drone's ability to correctly locate the target. The test returned the information shown in the drone's target recognition flow graph in figure 14.

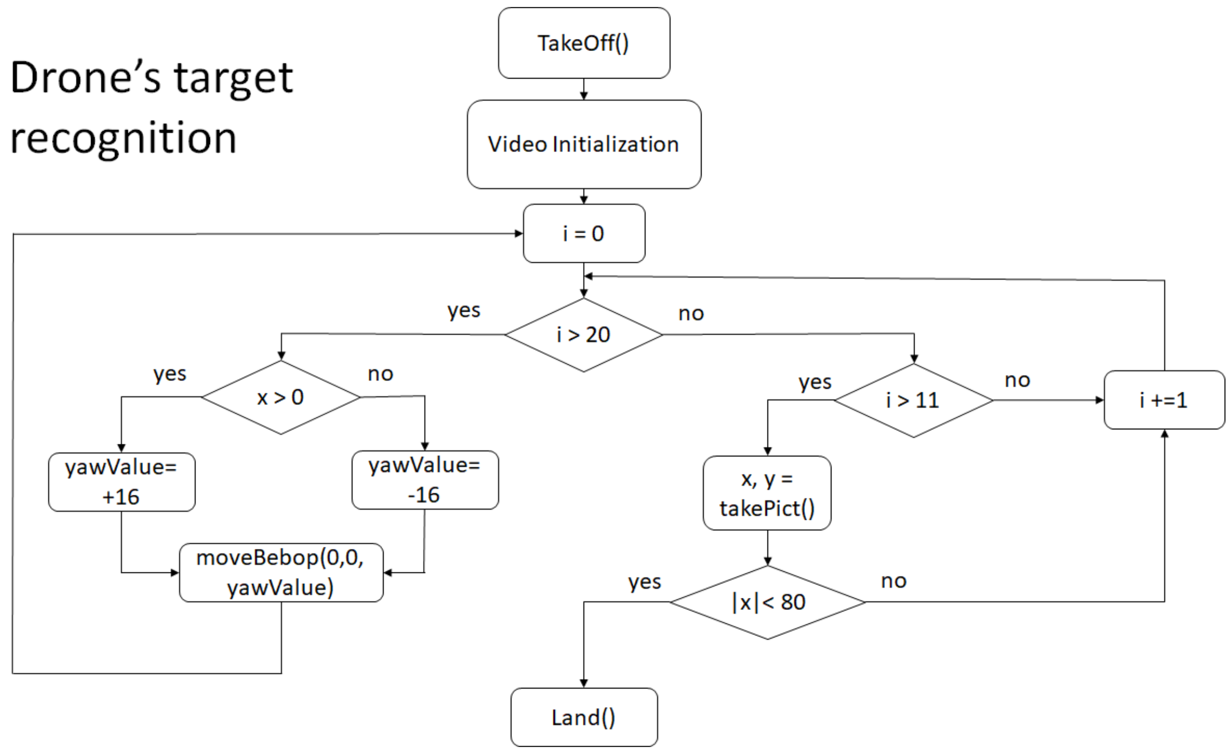


Figure 14. target recognition flowchart.

- Image merging: Originally the SRS document [3] contemplated T3 for obstacle avoidance, yet this objective was optional and the image merging routine was prioritized. As a consequence, a new test was designed to evaluate this task routine. As described in the black box testing, the input is three photos, and the expected output is one fully merged image. The path test, with the steps of this test, is presented in the image merging flow graph in figure 15.

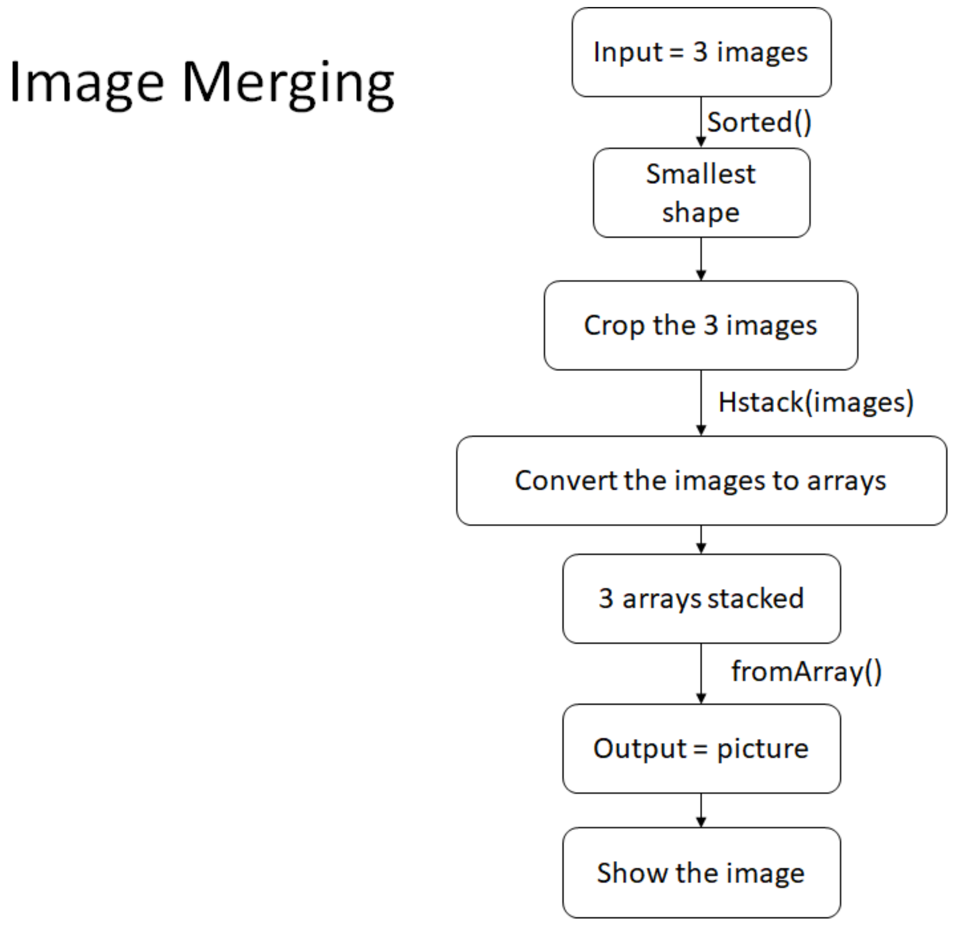


Figure 15. Image merging flowchart.

4.2. <u>Testing Results</u>

Below are presented some of the testing results obtained.

- Autonomous flying: The following image shows the drone mid-flight.

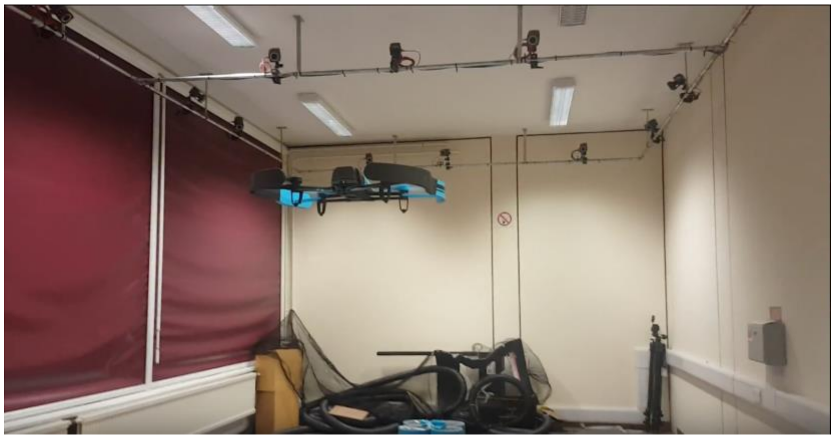


Figure 16. Flying drone.

The appendices contain a video that shows the drone performing this test.

- Drone's target recognition:

The image below is showing both points, the red one shows the routine achieved to find the target and the black is the marked center to orient the target as centered as possible.

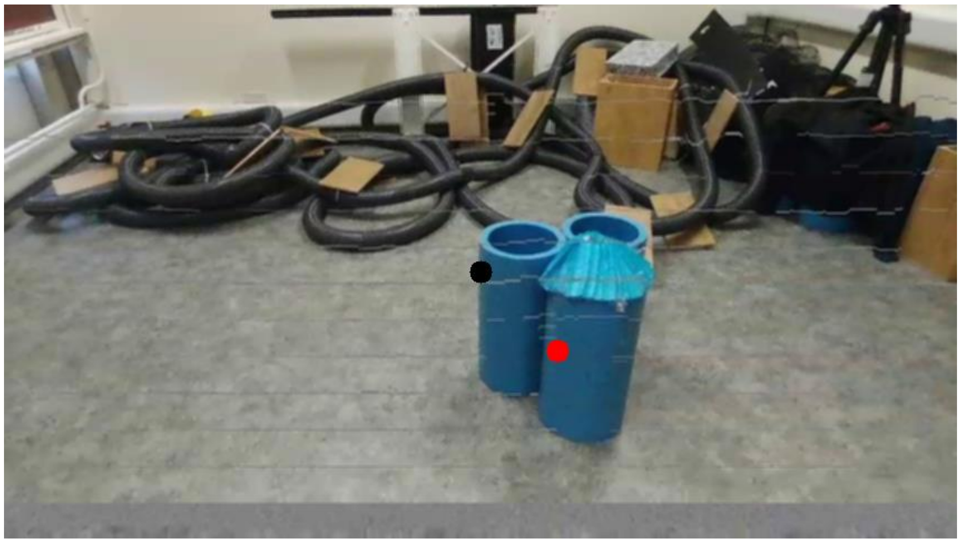
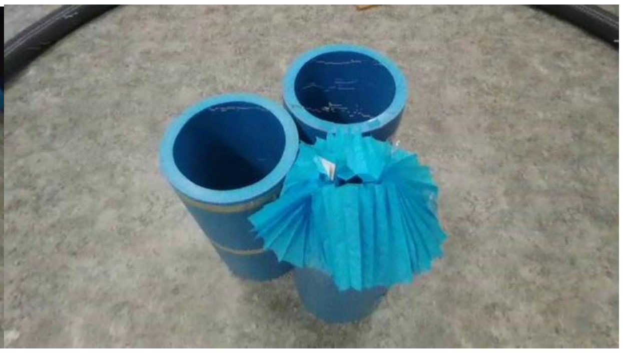


Figure 17. Target's image with marks its center and on the center of the image.

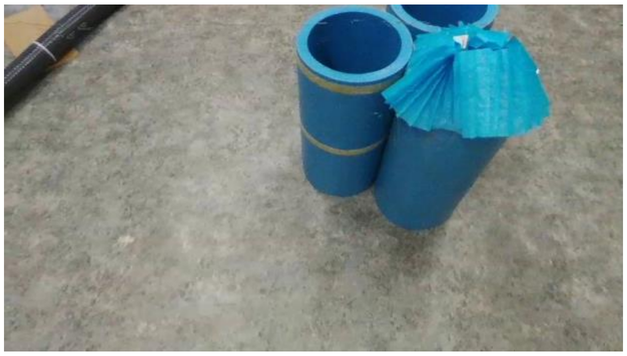
- Image merging:

Below are shown the three original images (Fig. 18,. A, B, C) and the final merged image (Fig. 18 D) obtained from this task's evaluation.

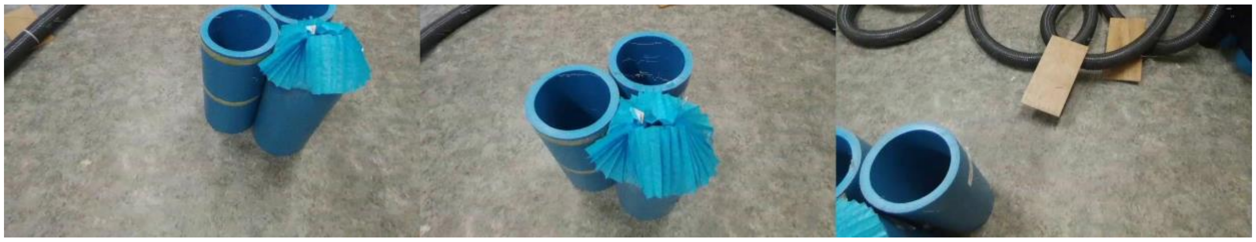
(A)



(B)



(C)



(D)

Figure 18. original images A,B,C. Final merged image D.

For videos and more evidences of results, consult appendices.

4.3.**Behavior**

The test results differs according to the battery or the take off position. However, after discussing and reviewing the taken pictures, it can be said that the system were improving from one test to another. the following pictures represents a test made with

100% full battery, with almost the same take off position, but with different type of marker.

The first test marker was only the blue cylinders, the set were meant to achieve a small marker. The next pictures (Fig 19) show the obtained the three pictures taken as the result of the test.

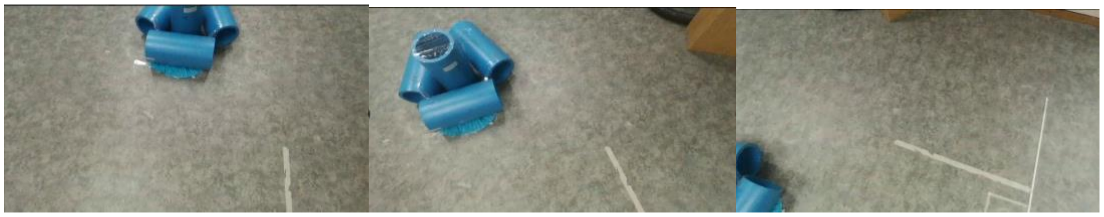


Figure 19A, B C. First, second and third photo of the target.

When running the test, the drone's behavior was not the wanted. The flight started by quickly reaching the desired altitude; however, the straight forward movement had a problem as the drone seemed to be stuck in the air. This lack of speed made the drone slide while rotating and the taken pictures were not good enough.

After many attempts, the marker changed to have more cylinders and the colored papers were added. The taken photos illustrates as following.

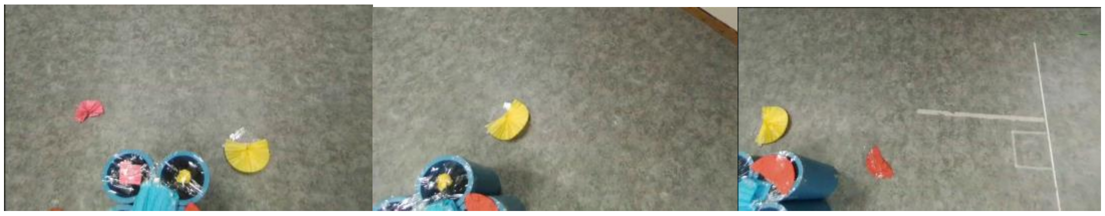


Figure 20A, B, C. First, second and third photo of the target.

Between these two tests a changes in the values for drone movement were modified. Even though the test was ran with a 100% battery, a better system response was noticed with 85% of battery. The taken picture's quality was better as the drone camera seems more stable. It can be concluded that behavior of the drone changed more according to the battery life than the changes in the speed values.

5. Conclusions

The system is able to perform different tasks. It performs target recognition, acquires pictures of the target and merges these ones to obtain a general overview of the target and the area around it.

The agile method used to coordinate the team during this project, has proven to be efficient. It allowed to made changes during the different sprints to give the group the opportunity to follow the different phases of the project in a more flexible way.

The language used to program all tasks of the system was python. Python was ideal given that it has a great support. Further development of the UAV system would include the different tasks being integrated in a single code, incorporating both programs, the drone sequence for autonomous flight and the image merging process.

However, there are several problems related to the performance of the used hardware. The Bebop presents different problems related to the flight stability and precision, especially in the presence of wind. These problems are related to a light slope of the drone position on the left, during the flight.

Other improvements can be achieved with the use of a drone with better performances and with a more accurate routine for its control. The object recognition part of the project can be upgraded elaborating a more sophisticated object recognition analysis that must be followed by a better hardware to ensure a real-time analysis.

References

[1] min2bro, 12 July 2017 . [Internet]. Available: https://kanoki.org/2017/07/12/merge-images-with-python/. [Accessed: 14 March 2018]

[2] Shawn Gomez, 19 June 2014. [Internet]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4081273/. [Accessed: 15 March 2018]

[3] A. Mendoza Filisola, E. Zaragoza Guajardo, J. Ruiz Argandona, L. Mirante and V. Olsen, "Intelligent UAV Technology for Visual Inspection Software Requirements Specification."

[4] Dlevy, S. (2018). *simondlevy/OpenCV-Python-Hacks*. [online] GitHub. Available at: https://github.com/simondlevy/OpenCV-Python-Hacks/blob/master/greenball_tracker.py [Accessed 14 Mar. 2018].

[5] The Codacus. (2018). *OpenCV Object Tracking by Colour Detection in Python - The Codacus*. [online] Available at: https://thecodacus.com/opencv-object-tracking-colour-detection-python/#.WrGfNOgZ42w [Accessed 13 Mar. 2018].

[6] GitHub. (2018). *Robotika - Katarina*. [online] Available at: https://github.com/robotika/katarina [Accessed 22 Feb. 2018].

[7] Docs.python.org. (2018). *15.1. os — Miscellaneous operating system interfaces — Python 2.7.14 documentation*. [online] Available at: https://docs.python.org/2/library/os.html [Accessed 7 Mar. 2018].

[8] Pypi.python.org. (2018). *opencv-python 3.4.0.12 : Python Package Index*. [online] Available at: https://pypi.python.org/pypi/opencv-python [Accessed 7 Mar. 2018].

[9] Docs.python.org. (2018). *17.4. signal — Set handlers for asynchronous events — Python 2.7.14 documentation*. [online] Available at: https://docs.python.org/2/library/signal.html [Accessed 7 Mar. 2018].

[10] Numpy.org. (2018). *NumPy — NumPy*. [online] Available at: http://www.numpy.org/ [Accessed 4 Mar. 2018].

[11] Matplotlib.org. (2018). *Installation — Matplotlib 2.2.2 documentation*. [online] Available at: https://matplotlib.org/ [Accessed 28 Feb. 2018].

[12] J. Malik, “Agile Project Management with GreenHopper 6 Blueprints”, Packt Publishing, 2013. [Book].

[13] GitHub. (2018). *Parrot BEBOP developer*. [online] Available at: https://github.com/AutonomyLab/bebop_autonomy/issues/77 [Accessed 7 Mar. 2018].

Appendices

Please refer to the Appendix folder to find the following:

1. **User documentation.**
2. **Installation instructions.**
3. **System code: Autonomous Flying and Merging Images codes.**
4. **Minute recollection.**
5. **Test results, photos and videos.**