International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 7 Issue 11, November-2018

Autonomous Landing of a Drone using Smart Vision

Taylor Ripke
Department of Computer Science
Central Michigan University
Mount Pleasant, MI

Tony Morelli
Department of Computer Science
Central Michigan University
Mount Pleasant, MI

***Abstract* - A key facet of ongoing research in unmanned aerial vehicles (UAVs) is the capability to perform an autonomous landing without human intervention using onboard sensors and vision. This paper presents a detailed autonomous landing approach sequence for a UAV consisting of two components: target localization and relative positioning. The first step involves locating the target using thresholding and image segmentation. Finally, the drone is moved to the target location and position is maintained and altitude decreased through the use of a proportional, integral, and derivative (PID) controller. The system was implemented in python 2.7.12 using the AR.Drone 2.0 with the PS-Drone [1] flight controller.**

Keywords – Vision, localization, PID, threshold

I. INTRODUCTION

A century since the Wright brothers took their first flight, the world has seen some of the greatest advancements in aeronautics from commercial flight to autonomous rocket landings [6]. SpaceX has been paving the way for commercial rocket reuse by successfully landing their rocket on a drone platform in the ocean [6]. Drones have also become an important asset in many fields, including photography, meteorology, and military use. Furthermore, their potential applications are actively being explored developed in the research community.

The following paper is building on research previously completed [7] that now focuses on the autonomous landing of an AR.Drone with motivation from [3]. This paper focuses on the autonomous landing of the drone that serves as a second pair of eyes for an unmanned ground vehicles (UGV). The UGV is the central hub and the drone works to send information to the robot for path planning assistance. The system, depicted in Fig. 1, shows the design of the UGV and the landing pad used to land the drone.

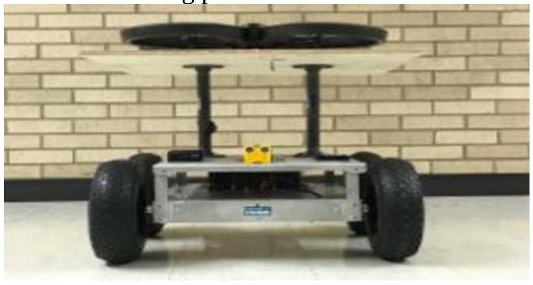

Fig. 1 Titan: Cooperative Robot-Drone Agents

This paper focuses on the autonomous landing of the drone that serves as a second pair of eyes for an unmanned ground vehicles (UGV). The UGV is the central hub and the drone works to send information to the robot for path planning assistance. The system, depicted in Fig. 1, shows the design of the UGV and the landing pad used to land the drone.

Autonomous flight and landing brings out the greatest challenges of artificial intelligence (AI), which include the ability to adapt to a rapidly changing environment and make informed decisions given the current data. Barták et. al. describes three major problems when developing real-world AI [3]:

1. The software must run in real time.
2. The environment is not always as the agent perceives it.
3. Things do not always work as the agent intends.

These facets of AI echo the challenges researchers face when implementing autonomous behavior in intelligent agents. A future section will highlight some of the hardware challenges that were overcome when developing a low cost, reliable method of controlling the system.

The motivation for this paper was derived from Barták et. al. [3] and their work on autonomously landing the AR.Drone platform. Similarly, the system presented in this paper achieves the same goal; however, in a different manner depending on system and environment constraints. The AR. Drone 2.0 platform described in the following section is equipped with a vertical camera used internally by the drone for stabilization and speed purposes; however, is also exploited for its ground view. The process consists of two parts (1) identifying the target location and (2) positing the drone above the target and descending using a PID controller. Preliminary results are discussed following the implementation and a detailed view of future work is presented.

II. AR DRONE 2.0

The drone used for our experiments is the AR.Drone 2.0 by Parrot [8]. The drone is the next generation of the AR.Drone line and features several feature upgrades over its predecessor. Of the improvements, one of the most notable is the improvement of the vertical camera, which is used extensively in autonomous landing protocols. The vertical camera features a 64-degree lens, capable of recording up to 60 fps [8]. It is lightweight, weighing approximately 420g with the indoor hull on (highly recommended) [8]. The battery lasts approximately 12 minutes per charge, depending on the activity done.

The drone communicates with a computer or compatible device over Wi-Fi. As a low-cost solution, the drone is capable of being controlled by a Raspberry Pi 3; however, it is recommended that it be controlled by a more powerful device, such as a NVIDIA Jetson TX2. Our setup uses this computer which also controls the robot. The PS-Drone flight controller used in our research is only compatible with Linux systems, and thus makes a great board for controlling the drone.

III. TARGET LOCALIZATION

Once the UAV has completed its tasks and is ready to land, the first step is to locate the landing pad. This approach assumes that the UAV is in the relative vicinity of the landing pad or has software guiding it towards the known landing pad location. There are numerous computer vision techniques available for target recognition including template matching, feature detection via key points (SIFT, SURF, and others), convolutional neural networks (deep learning techniques), and blob detection [3]. To keep computation relatively inexpensive, blob detection was selected. The following steps were used to identify the target landing location:

1. Obtain image from drone
2. Apply Gaussian Blur (OpenCV)
3. Threshold image (Eq. 1)
4. Apply dilation/erosion (closing)
5. Find largest round contour (OpenCV)
6. Find center x,y cords. (Eq. 2)

Once the image has been received, the first step is to preprocess the image. First, the image is converted into the hue, saturation, and value (HSV) color space. RGB images (or BGR in OpenCV) are difficult to work with as lighting conditions play a significant role. HSV separates image intensity from color information. Therefore, the hue can be isolated for color thresholding. A gaussian blur is applied to remove noise and smooth edges. OpenCV provides a method call to perform this operation [2]:

```
img = cv2.GaussianBlur(frame, kernel, 0)
```

The kernel parameter of the gaussian blur specifics "how much" blur we want to apply to the image. A larger filter will apply a greater blur to the image. Next, we apply Eq. 1 to threshold the image to locate the red circle we chose as our landing area. Any color can be selected; however, it is recommended to use vivid colors that vastly contrast every day colors (bright red, bright pink). Colors such as green or gray can easily be misclassified as grass or sidewalk while the drone is searching.

$$\text{img}(x,y) = \begin{cases} 1 & \text{if lowThresh} < \text{img}(x,y) < \text{highThresh} \\ 0 & \text{otherwise} \end{cases}$$

Eq. 1 Binary Thresholding

Following thresholding, a closing operation (dilation followed by erosion) is applied to the image to remove noise and fill gaps in the thresholded image. Color thresholding is a fast operation; however, being the most basic form of isolating blobs, it is very subjective to noise. Therefore, a series of operations are discussed to reduce the probability of incorrectly detecting a false blob. Thresholding the hue channel allows a specific color to be isolated; however, in some cases the entire blob may not be filled in because part of it is outside the threshold set. Therefore, the closing operation fills in gaps and provides a more robust isolation. Both are standard operations found within the OpenCV library [2].

The largest circular contour is subsequently found with help from OpenCV. In summary, a contour is essentially a line that joins all continuous points along a given boundary that have the same color. First, find all contours in the image using the following command [2]:

```
_,contours,_ = cv2. findContours(img,
cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
```

This returns all the possible contours in the image. Note: there may be many contours that are found due to noise or the color selected. If a good color is selected, there will be minimal noise. Next, try to isolate the circular objects using OpenCV's approxPolyDP function [2]:

```
approxCir = cv2.approxPolyDP(contour,
0.01*cv2.arcLength(contour,True), True)
```

This function can be used in different ways, combined with the contour area to approximate a circle. OpenCV provides other methods to achieve the same result [2]. All contours that fit the above criteria are isolated and kept as a probable landing pad. Usually at this point, there are very few, if only one probable blob being the landing pad. The next step is to isolate the largest blob. This is done by iterating over all remaining contours and finding the largest blob using the OpenCV function [2]:

```
area = cv2.contourArea(cnt)
```

The final step in identifying the target location is to compute the x,y coordinates of the contour selected. The landing pad and location relative to the center of the drone is shown in Fig. 2. The blue dot represents the calculated center of the blob (landing pad). The yellow line represents the distance to the center of the landing pad in the x,y direction. The red box represents the safe landing area. Once the center of the circle is within the red box, the drone may safely start descending. The green lines illustrate the different quadrants.

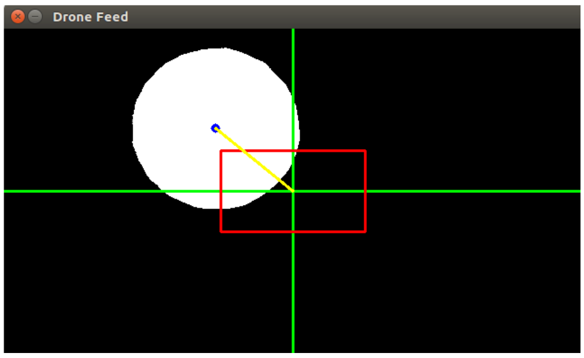

Fig. 2 Target Localization

The central moments are calculated using Eq. 2 shown below [2]:

$$\bar{x} = \frac{M_{10}}{M_{00}} \quad \bar{y} = \frac{M_{01}}{M_{00}}$$

Eq. 2 Components of Centroid

In OpenCV, this can be achieved by the following function [2]:

M = cv2. moments(bestContour)
xPos,yPos = int(M[‘m10’]/M[‘m00’], int(M[‘m01’]/M[‘m00’])

IV. LANDING PROCEDURE

Following the identification of the landing pad’s coordinates, there were two approaches considered for landing derived from [3]:

1. Navigate to the x,y coordinates (forward/backward, left/right), maintain position over landing pad, and descend.
2. Navigate to the x,y coordinates while descending.

Due to the design of the landing pad being 50 cm from the ground, we chose the former method. Depending on the height of the drone from its previous commands, the latter option proved to be more difficult in certain scenarios. First positioning the drone’s 2D position made it easy to hold the position using PID controllers while the drone descended.

V. COORDINATE TRANSLATION

As discussed previously, the x,y coordinates of the landing pad were obtained by calculating the central moments of the blob found. The image dimensions are 640x360. By positioning the drone directly above the landing pad and subsequently descending, the image can be treated as a graph during positioning, making the assumption that the altitude of the drone will remain relatively constant. The altitude of the drone was being controlled via a respective PID controller.

Using the image as a graph, it was divided into four quadrants, each of which represented different directions the drone had to navigate to as shown in Fig 3. Three PID controllers were used to navigate the drone (forward/backward, left/right, altitude). An additional PID can easily be added to the controller to control rotation respective to the landing pad; however, for our project it was not necessary.

The following is an example of the implementation. Based on the quadrant the landing pad was in, Fig. 3 depicts the operations need to be performed on the commands. For example, if the drone was in the upper right quadrant, the x speed needs to be multiplied by a negative one and y by a positive one.

First, the drone checks if it is in the center. If so, it maintains positions and begins descending. Otherwise it checks which quadrant it is in and applies the operation described below. As stated previously, the image is 640x360. Each quadrant is divided in half, so x:0-320 and y:0-180 represents the first quadrant. Next the distance to the point is calculated. If in the upper left quadrant, the calculation is as follows: xPos = 320 - xPos, yPos = 180 - yPos. Next, the speed is chosen and normalized: fx = -0.01*xPos/320 and fy = 0.01 * yPos/180. The data is is fed into the respective PID controller, which then translates it into a movement command for the drone.

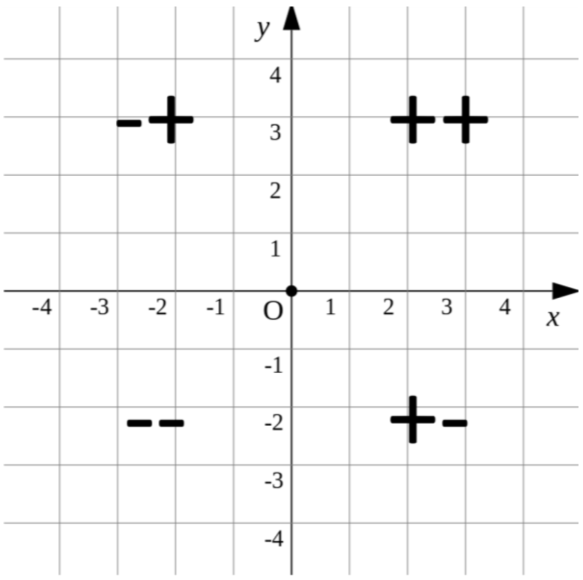

Fig. 3 Motion Coordinate Translation

Assuming a relatively constant height, movement was a function of how much the drone had to move forward/backward and left/right. In the center of the image as shown in Fig. 2 there is a box which specifies the region the drone can begin descending because it is close enough to the center. The 2D PID controllers are maintaining x,y position while the third is now minimizing the altitude error.

VI. PID CONTROLLERS

Three PID controllers were implemented, although the fourth can easily be added (yaw). The three primary controllers are forward/backward (pitch), left/right (roll), and altitude. The motivation for our PID controllers stemmed from [3], where Barták et. al. implemented four, as described previously.

The distance in the x,y direction was updated each time a new image was received from the drone. Assuming a 2D planar surface (altitude remains relatively constant), we first position the drone over the landing pad and then began a controlled descent. If the drone managed to drift too far away from the center, it would immediately begin correcting the x,y position, and subsequently begin its descent again.

A detailed description of PID controllers can be found in many text; however, the implementation for ours was from [4, 5]. The output of a PID controller seeks to minimize the error and bring the object closer to the desired value. George Gillard summarizes a PID controller as follows [4]:

1. Proportional - your error, the value between where it is now and where it should be.
2. Integral - the running sum of previously errors, used for making fine movements when the error is small.
3. Derivative - the change in errors, used to predict what the next error might be.

One of the most challenging aspects of PID controllers is tuning the constants for each part of the equation, the details of which are outlined in [4].

VII. IMPLEMENTATION

This application was created in python 2.7.12 with the assistance of an existing drone controller PS-Drone by J. Philipp De Graff. The application provides a seamless method of communication between the AR.Drone 2.0, providing access to movement commands and sensor data. It was also written in python and allows the user to control the drone via commands or the keyboard. OpenCV 2.4.13.5 is used for image processing with python wrappings.

The application itself is part of the larger project and is designed for use with an autonomous robot [7]. The robot has a large landing platform that is 50 cm above the ground, providing enough room for small error during descent. One of the challenges faced during development was the sensitivity to the elevated platform. Similar to [3], we employ the other method of landing the drone, as elevated landing platform caused several problems. Depending on the original height of the drone, if the drone simultaneously descended and moved towards the landing platform, it sometimes would miss the platform entirely or crash because it was elevated. In other cases, due to the inherent nature of the drone to maintain a relatively constant altitude (not rapid changes), if the drone was already low to the ground and suddenly flew over the platform, there the ultrasonic sensor detected that the ground was suddenly 50 cm closer, and the drone rapidly moved up as to avoid the ground.

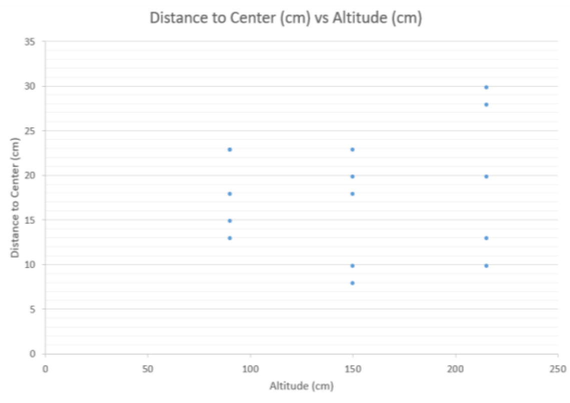

Fig. 4 Distance to Center (cm) vs Altitude (cm)

Therefore, we chose to position the drone above the target and slowly descend while maintaining position to minimize the problem of receiving the wrong depth information. In some cases where wind was present, the drone would away from the platform, which in that case it was programmed to hold its position, rise for a bit, and begin searching for the landing pad in the opposite direction of its last movement.

VIII. EXPERIMENTAL RESULTS

There were several experiments to test the precision of our autonomous landing procedure. For each test, we attempted the landing procedure until there were five successful landings. Barták et. al. [3] executed the landing procedure until they had four successful landings; however, did not specify how many were actually performed. We used similar metrics as [3] where we defined a successful landing as 30 cm from the landing point. The two results that we were interested in was the distance to the center versus the altitude and the altitude vs the time to land as shown in Fig. 4 and 5 respectively. The elevated platform that the drone has to land on is approximately 2 square feet (about 60 cm). If the drone landed outside of the 30 cm diameter, the test was discarded and deemed unacceptable as the drone would have a difficult time achieving flight again due to the pressure imbalance (i.e. two sets of propellers were over the platform - roughly 10 cm from ground - and the other set was hanging over the edge, approximately 50 cm off the ground, which would cause the drone to take off erratically).

There are numerous factors that can contribute to an unsuccessful landing: (1) information (packet) loss (2) losing the target and having difficulties relocating (3) the natural behavior of the drone (4) the drone got close to landing; however, drifted too far off center and had to ascend, causing the drone to maneuver to achieve stability due to pressure imbalance caused by the elevated platform.

In the experiments, the drone was manually flown to an estimated 90, 150, or 215 cm above the platform. At 90cm, nine landing attempts were done, 5 of which were within the acceptable parameters. At 150 and 215 cm, seven landing attempts were performed, of which 5 were successful from each. The drone was manually flown to simulate the drone performing a given assignment, and then preparing for the landing sequence. The drone was flown until the landing marker was visible and then the drone entered into the autonomous landing sequence.

IX. FUTURE WORK

Project Titan is a collaborative effort focused on investigating the interplay between intelligent agents sharing a common goal. Potential applications of this system include disaster recovery or planetary exploration.

As described previously, the current system is comprised of two UGVs and UAVs, although it has the capability to be expanded to more agents.

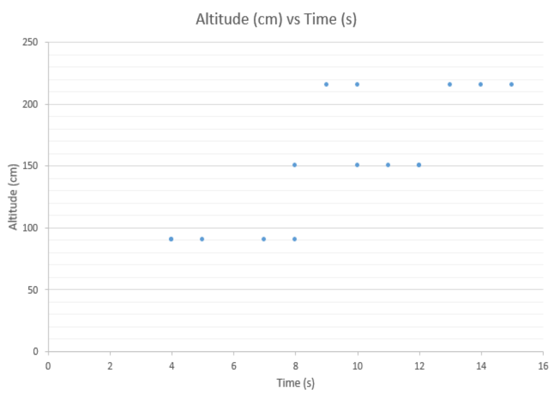

Fig. 5 Altitude (cm) vs Time (s)

This project is designed to explore the process of developing smarter algorithms capable of advanced decision making. For example, picture a highway with road construction and traffic is being diverted to the other side of the median. In current scenarios, the autonomous car would give control back to the human to get the car past the obstacle. However, the car should be able to make real time decisions for itself and follow the general flow of traffic. If the car sees that other cars are driving to the opposite side, it should follow the same behavior (swarm analysis).

Considerable research has been done concerning autonomous navigation for robots in indoor and outdoor environments. We hope to extend this work by introducing multiple agents working together to solve a particular tasks, such as planetary exploration and mapping.

X. REFERENCES

[1] J. Philipp De Graaff. 2014. PS-Drone. http://www.playsheep.de/drone/. Accessed January 19, 2018.
[2] OpenCV. 2014. OpenCV 2.4.13.5 documentation. https://docs.opencv.org/2.4/. Accessed January 20, 2018.
[3] Barták, Roman & Hraško, A & Obdržálek, D. (2014). On autonomous landing of AR.Drone: Hands-on experience. Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2014. 400-405.
[4] Gillard, George. 2012. An introduction and tutorial for PID controllers. https://udacity-reviews-uploads.s3.amazonaws.com/_attachments/41330/1493863065/pid_control_document.pdf. Accessed January 22, 2018.
[5] King, M. 2010. *Process Control: A Practical Approach.* Chichester, UK: John Wiley & Sons Ltd.
[6] Blackmore, Lars. 2016. Autonomous precision landing of space rockets. 46. 15-20.
[7] Ripke, Taylor & Reason, Kellen & Morelli, Tony. 2017. Cooperative Robot-Drone Agents for Obstacle Avoidance using Smart vision. International Journal of Engineering Research & Technology. Vol. 6 Issue 05, May - 2017.
[8] Parrot. 2017. Parrot AR.Drone 2.0. https://www.parrot.com/us/drones/parrot-ardrone-20-elite-edition. Accessed January 22, 2018.