Automatic UAV landing

Tuesday 29th April 2015

James Fairbairn
Student
University of Canterbury
jaf138@uclive.ac.nz

Richard Green
Supervisor, Associate Professor
Department of Computer Science
University of Canterbury
richard.green@canterbury.ac.nz

Abstract*—This paper proposes a solution to the problem of having a UAV automatically land. The method proposed by this paper is to use a combination of fiducial marker tracking, depth detection and natural feature tracking to determine the location of landing and guide the UAV to a safe landing. The marker detection was done through a method of thresholding, extracting contours, extracting corners (using Harris corner detection) before attempting to match these corners to the markers being detected. The natural feature tracking was done using ORB detection for key point detection and a FLANN based matcher for matching keypoints between the reference image and the live image. Using these methods individually the average detection rate of markers over a live video was 63% for a 18x18cm marker from a range of 1.5m and the average detection rate for natural features was 99%. Using these together in conjunction allowed for an almost continuous rate of detection in the range of 1.5-2.5m above the ground.

I. Introduction

With the increase in use of UAVs within commercial areas it is becoming necessary for more of their functionality to be automated. Many drones are now being used as freighting vehicles, in photography/filming, for inspections, for disaster response and many in other aspects of life [1]. Giving these drones the ability to automatically land could help increase the number of uses they can have and the places that they can be used.

By giving drones the ability to automatically land or return to a set base station it would reduce the amount of work needed to be done by the pilot. It could also help if a situation arose when manual navigation back to a certain location cannot be easily achieved.

This research aims to find a method whereby a drone can determine its 'base station' and automatically land there using either some sort of marker or natural feature tracking of the environment. It will start by looking into creating a fiducial marker for tracking and depending on how the results for this go may be expanded to look at natural feature tracking so there is no need for a marker to be placed at the landing site.

II. Background Information

This type of tracking and landing is already being developed in some commercial areas. For example, Amazon's Prime Air service, which is currently in development, uses a marker system to determine where to land [2]. However, they tend to rely heavily on Fiducial markers and are therefore limited in their applications.

A. Fiducial Markers

Fiducial markers are usually black and white markers that are used in computer vision to mark out certain locations and are quite often used for navigation.

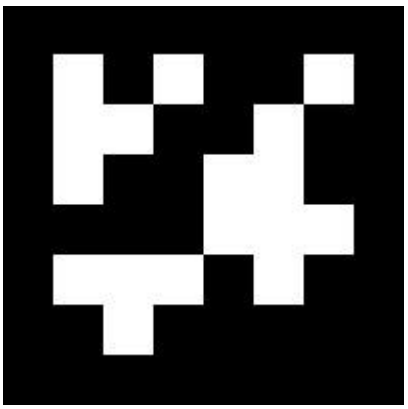

Figure 1: Example ArUco Marker

These types of markers have been around for a long time There have been many different variations of them and the methods used to track them. Some of which are ARToolKit, ARTag and ArUco. With each new variation of marker there have been attempts to try and improve on each other in their success rates for detecting markers by using a variety of marker and a variety of algorithms for detecting them.

The method of detecting these markers is usually follows this kind of algorithm:

- Firstly, some thresholding is performed on the image. This can be changeable and depends on lighting etc.

- From the thresholded image contours are extracted.
- Those which are not convex or do not approximate a square are excluded. There is also some other filtering for contours which are too big/small or too close to each other.
- Corner refinement. Making sure that you have accurately located the corners. Potentially using Harris corner detector.

Some marker detection stops at this point and will just accept any square marker in the image [3]. However, most will go on further and analyse the inner section of the marker. They will look for distinctive features within it that identify it as an actually marker and not just a square. For instance ArUco breaks down the marker and reads it in a similar way to a barcode in order to determine that it is actually a marker [4].

After you have this information it is possible to estimate the pose of the camera, which helps determine alignment with the marker and navigation.

B. *Natural Feature Tracking*

Natural feature tracking is different to the marker system in that it takes an environment and extracts key points from the environment. It then uses they key points to match up against other images to detect if it is looking at the same thing.

There are many different ways of extracting these key points and it can come down to the use case depending on which one you use.

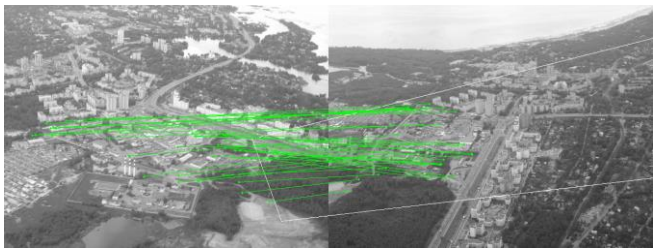

Figure 2: Natural Feature Tracking example of changing perspective. Detection using SIFT

This method would allow for more usages as all you would need would be an image of the landing zone that the drone could use as reference when it is searching for it.

C. *Alternative Methods*

Several other methods for the automated landing of UAV's have previously been explored. All these methods use varying techniques and ideas to allow drones to land automatically.

One of these methods was to use four infrared lights to mark the landing zone and install a camera on the drone that picks up the exact wavelength of the lights while filtering out other wavelengths. With this method the location of the landing zone could be determined by tracking the centre of the infrared lights and extracting from them the direction and location of the landing zone. [5]

One limitation with this maybe be interference from sunlight. However, this is not incredibly likely due to the filtering of the wavelengths that are accepted. It also relies on direct line of sight with the landing zone and may not accommodate if several of the lights are blocked by other objects.

III. Proposed Solution

There are many different ways of tracking and detecting environment and specific markers. The solution proposed for this problem of automatic landing of drones is to start by using a ArUco marker system where the ArUco marker is placed at the landing zone. This would enable the drone to detect the landing zone and then using the depth camera detect the distance the marker is away from the drone and therefore land accurately. The drone would locate the general position of the landing zone using its GPS location and then start searching for the marker for the landing zone. One of the limitations with this approach is that the markers would have to be reasonably large in order to be detected by the drone, especially if the drone is at a reasonable altitude. This could be overcome by getting the drone to hover at a certain altitude so that it definitely can see the markers.

If this solution works well and is easily completed with a high accuracy rate (>=90%), then the solution proposed by this project could be extended so that it uses natural feature tracking of the landing zone. This would involve having an image of the landing zone and extracting from it key points to track and then searching for them once the drone has arrived in the general location based on GPS. During this process tests for different algorithms that are used in extracting these keys points would be carried out to see if there is any algorithm that performs better under these circumstances. These algorithms could be any of the following:

- SIFT (Scale-Invariant Feature Transform)
- SURF (Speeded-Up Robust Features)
- BRIEF (Binary Robust Independent Elementary Features)
- ORB (Oriented FAST and Rotated BRIEF)

Using any of these methods could improve the accuracy with which UAVs can land. With the later having many practical use cases and not many limitations as the altitude approach accuracy should not be as great a factor when detecting natural features as it does with markers.

IV. Method

All of this project was undertaken using at custom compile of OpenCV 3.1 and some additional modules added into the build.

A. *Marker Detection*

The marker detection part of this paper will give the accurate information needed in order to make sure that the descent of the UAV is a controlled one. This is mainly due to the inaccuracies that arise with natural feature detection.

The basic concepts of how these markers work has previously been laid out. There are adjustments to this that can improve accuracy. Greying the image then applying some level of thresholding can greatly improve detection as the markers themselves are black and white. However, in certain lighting settings it is difficult to adjust this thresholding appropriately and it can lead to the marker not being detected at all. Experiments could be undertaken to try and find if this

method actually helps marker detection over a wide range of examples.

B. Depth Detection

After detection the marker using a depth camera to determine the distance from the ground as the UAV descends will help to bring it down in a gradual and controlled manner. The Intel RealSense cameras F200 and RS200 both have depth capabilities that can be used for this purpose.

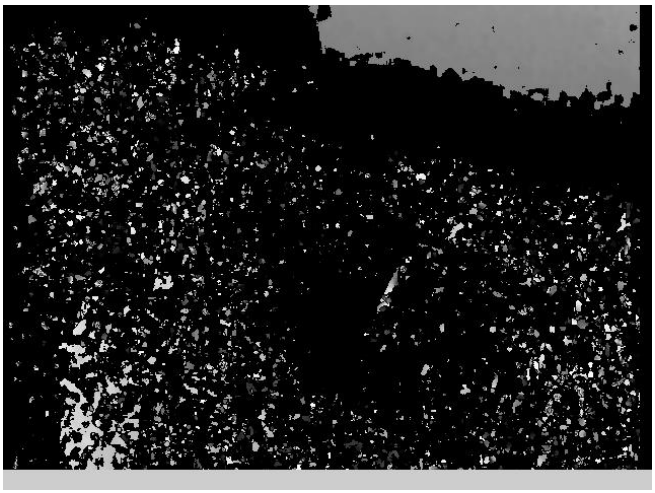

Figure 3: Depth image of Marker and Wooden floor from 1m above the ground

As depth values for each pixel are not always accurate, see Fig 3, it will most likely be necessary to take a number of depth values that are linked to the marker in some way and average them. One approach may be to use the edges and centre of the marker with a few additional points along the diagonals of the marker to determine the depth. Min and max values from these values could also be used to ensure that if there are invalid readings in some of the points detected that they are taken into account.

Figure 4: Determining to go down based on average depth values for 9 different places on the marker

C. Natural Feature Tracking

After looking into the different methods for extracting key points from images the ORB algorithm was decided upon. The main reasons for choosing ORB was that it is far less computationally intense than SIFT and SURF, as is discussed in "ORB: an efficient alternative to SIFT or SURF" [6]. Both SIFT and SURF are patented [7], so there are royalties involved in their usage in a real-world scenario.

ORB uses a combination of FAST keypoint detection and BRIEF descriptor. The way that FAST works is that it selects a circle of 16 pixels around a point that is being evaluated to determine whether it is a corner. It then determines whether these surrounding pixels are all darker/lighter than the pixel being inspected, plus or minus some threshold value. If they are it determines that it is a corner. These conditions can be written mathematically as follows:

For a set of N contiguous points S:

1) $\forall x \in S, I_x > I_p + t$

2) $\forall x \in S, I_x < I_p - t$

Where I_x is the intensity of the points in the circle around the point being inspected, I_p is the intensity of the point and t is the constant threshold value [8] [9].

ORB then makes some additions to the FAST algorithm so that it takes into account orientation.

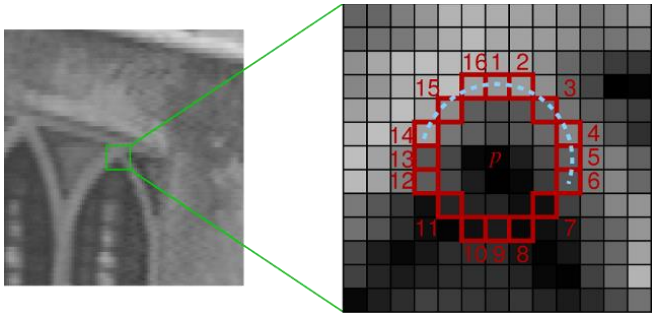

Figure 5: FAST corner detection [10]

ORB then uses BRIEF descriptors to identify each keypoint again with a few additions to take into account rotation [7].

After detecting the keypoints in a reference image these keypoints can be matched against real time footage. This can be used to determine the homography matrix that describes the relationship between the two images. Meaning that any point in the original image is translated to its position in the real time image using the homography matrix.

$$p' = Hp$$

Where p is the original point in the reference image, p' is the shifted point and H is the homography matrix.

Using this it is possible to estimate the direction it is necessary to move in order to realign the camera. This can be achieved by taking a 3 row vector full of 1's and multiplying it by homography matrix. Then taking the result away from the original vector. This gave me a vector that indicated the movement necessary to get back to the starting position. As it is assumed the majority of the motion will be on the horizontal plane the z value of this matrix was ignored. Then taking the x and y values estimations of which direction to move can be made (i.e. x is positive move right, if negative vice versa).

Estimations for rotation can also be made, as long as the rotation is only in the horizontal plane. This can be done using the equation:

$$\theta = \cos^{-1}(p'.p)$$

Where p' is the normalised vector of the shifted point and p is the normalised vector of the original point. This approximation is very basic and doesn't take into account the z-axis at all and is therefore limited. Meaning that this value should not be trusted fully when trying to manoeuvre the UAV. However, this doesn't appear to affect the movement vectors.

This could be used in conjunction with depth detection and marker detection to help get the drone into roughly the right position (assisted by GPS). While taking off reference images could be taken to use when attempting to detect the landing zone when coming in to land. This would overcome the problems faced by marker detection and depth detection over large distances.

Using all three of these different methods combined into one should hopefully overcome the limitations of each method and create a fast and efficient solution.

V. Results

All of these results were gained using the following specifications:

- Camera:
  - R200 Intel RealSense Camera [11]
    - Depth/IR 640x480 at 60 FPS
    - RGB 1080p at 60 FPS
    - Depth Range 3-4m
  - F200 Intel RealSense Camera [12]
    - Depth/IR 640x480 at up to 120 FPS
    - RGB 1080p at 60 FPS
    - Depth Range 0.2-1.2m
- OS: Windows 10
- Processor: i5 5200U at 2.2 GHz
- IDE: Visual Studio 2015
- Language: C++
- Device: Laptop
- OpenCV Version: 3.1

A. Marker Detection

The built in library without any configuration (using simply the default values and not calibrating the camera) managed to detect the markers a reasonable amount of the time.

As an experiment 4 different videos of 2 different sized markers were captured. One marker was 18x18cm the other was 8x8cm. These were filmed outside on grass from 2 different heights. The first height being around 1.5m and the second 2.5m. While filming a mixture of movements were performed in order to see how the detection of markers was given different movements etc.

After collecting these videos, they were processed within the detection algorithm and the results shown in Table 1 were found.

Table 1: Results of Initial Experiment In Detecting Markers.

Video	*Detected Frames %*	*Length of Video (s)*	*Time taken (s)*
18x18 at 1.5m	37.3	14	34.8
8x8 at 1.5m	0	14	51.1
18x18 at 2.5m	95.1	16	67.2
8x8 at 2.5m	0	14	56.7

From these results it can be seen that the processing of the images to actually find the markers takes a reasonable length of time and therefore in order to get it to work properly in real time there will need to be some adjustments made. This could be in the form of adding better threading support, or instead of analysing ever single frame we could simply grab every nth frame. Of course it would mean that we would lose some information but at the same time as long as it is able to detect the marker a significant amount of the time it should not matter. Testing this theory produced the results shown in Table 2 when looking at every 3rd frame.

Table 2: Results of Initial Experiment In Detecting Markers looking at every 3rd frame

Video	*Detected Frames %*	*Length of Video (s)*	*Time taken (s)*
18x18 at 1.5m	32.9	14	32.8
8x8 at 1.5m	0	14	16.3
18x18 at 2.5m	94.7	16	20.7
8x8 at 2.5m	0	14	19.2

Another thing that can be seen from this that the smaller marker was not detected at all in either of the clips. This may have been to do with several things, the marker being too small, the paper the marker was on was slightly bent or there was glare on the paper which obscured the marker's edges. This can be seen in Fig 2.

Figure 6: Smaller Marker showing glare and the curve of the paper it's on

Looking through this video with the rejected areas being drawn at the same time it also looks like the marker itself is never detected and rejected. Sometimes sections inside the marker are detected, but are discarded as they are too small.

Some improvements in the rate of detection could potentially be gained through calibrating the camera [13]. This

would also give the ability to estimate the pose of the camera at the time it is taking the video. It could also be possible to see some improvement by changing the default values of how the image is processed to extract the marker. This could be done by adjusting one of the following:

- Thresholding
- Contour filtering
- Corner refinement

All of which can be altered during the detection process.

As a proof of concept for this idea it was decided to see if detection of the marker could be improved by first taking the image and thresholding it so that it was black and white. Experimenting around with this for a while the results shown in Table 3 were found.

Table 3: Percentage of frames with the marker detected with black and white image passed in

Video	*Detected Frames %*	*Length of Video (s)*	*Time taken (s)*
18x18 at 1.5m	53.5	14	18.1
8x8 at 1.5m	2.2	14	22.4
18x18 at 2.5m	68.1	16	23.0
8x8 at 2.5m	35.7	14	20.0

From this it can be seen that there is a significant increase in the detection rates in some of the videos. However, this has been paid for by a decrease in detection rates in the 18x18 at 2.5m video. However, after testing this idea or a live video stream the results weren't overly encouraging. With only 24% of frames with markers in actually registering. Whereas without this thresholding the percentage of detections was much higher, at 63%.

One of the limitations with marker tracking is that good lighting is required. One way to overcome this problem would be to use and IR camera (with its own light source). Using the F200 several experiments were undertaken to see if this could be used practically.

In the first experiment the 18x18cm marker was placed on the ground and a test was run to see at what level above the ground the marker was detected using the IR stream from the camera. Repeating this experiment multiple times gave an average value of 38cm above the ground. From this in can be seen that the use of only IR is not an appropriate solution to the problems of lighting. Of course as the IR light on the F200 is not incredibly bright, due to safety reasons, therefore different results may be attained using a different light source.

B. Depth Tracking

Using the average depth method previously elaborated on it was found that after going over 1.5m away, using the F200 camera, from the target there were no good depth values left to use to estimate the distance to the target. This problem also occurred when using the R200. However, the range on the R200 was much further, being able to see distances of up to 3.5m away before the readings from it became useless for the purposes of this project.

Due to this limitation it was necessary to find a way that the drone can determine whether or not to descend when it has no depth data. The method used in this case was to store a history of the average depth values of each frame where the marker is detected. If these values are all 0 for some set number of frames, for the experiments performed the number was 5, then we can assume that the camera is too high up to detect the ground yet and should therefore descend. As soon as it has descended a certain amount the history is cleared and the process begins again.

C. Natural Feature Tracking

Using a combination of source code that already exists, mainly from online tutorials [14] [15], and some additional tweaks for what was needed in this project, a method for detecting natural features and using them to try and determine whether a live image was related to reference image was constructed. This method used ORB for detection of keypoints and a FLANN (Fast Library for Approximate Nearest Neighbour) based matcher for matching keypoints between the reference and live images.

To test the accuracy of natural feature tracking a reference image for comparisons was taken. Then a live stream of 60 seconds was compared to this reference image. The live stream was completely different to the reference image and contained no view of the initial image at all. A similar experiment was performed, except the reference image was in the live stream during the experiment. The results from this experiment can be seen in Table 4.

Table 4: Natural Feature Tracking detection errors

Video Stream type	*Detected Frames (%)*	*Matches per detection (out of possible 500)*
Reference image not in feed	62	27
Reference image in feed	99	93

From this information it can be seen that Natural Feature tracking definitely has its limitations as it quite regularly detects matches between a reference image and a live stream even when they're completely different images.

For use within this project in parallel with marker detection and depth natural feature tracking should only be used to give the faintest idea of where the location is.

Combining all of these results together into one program to try and take all the useful features of each method the following pipeline was constructed. Firstly, it would determine whether or not the live stream contained a marker, alongside processing for natural features. If any markers are detected, then it goes on the check and see if the marker is roughly in the centre of the screen. If so, then it will send a command to start descending. When there are no markers detected it will go to the natural feature tracking. If there are more than 25% matches it will try to determine the way to go from the natural feature tracking results. As a further safety measure for using natural feature tracking it will only be used if the marker has not been detected in the last 10 frames.

Even with taking all of this into account to try and find the best possible solution, this method still has its limitations. Firstly, the natural feature tracking is not incredibly accurate

and the method used for determining the direction that needs to be travelled in order to centre the image is an approximation at best. It will also only work if the camera is facing directly down as it doesn't take into account changes in the z axis. Another major limitation on this method, and therefore the results gained using it, is that it will only work when the lighting is good. This method would not work at all at night time using standard cameras. The research performed using specialised IR cameras also suggests that landing at night using this method is not going to be overly easy.

D. Comparisons to other methods

Comparing this to the aforementioned infrared camera method of detecting a landing zone the margin of error for the position of the marker is comparable. At 5m away from the landing zone the margin of error on the x and y axis was 0.073m and 0.006m [5] respectively for the infrared method. At 2m away using my method the margin of error is similar. However, their method is still very accurate from ranges of over 450m away. At this range the marker detection is unlikely to work, unless it is a really large marker. Therefore, natural feature track would be used in its place. As has been seen from the test results using natural feature tracking it isn't incredibly accurate. Meaning the method of detection is more viable over longer distances. In saying that it is still accurate over shorter distances and could be used for a more controlled descent.

VI. CONCLUSION

Using background research an algorithm was devised for guiding a UAV to landing. This algorithm involved the use of fiducial markers, depth detection and natural feature tracking. Using these methods individually the average detection rate of markers over a live video was 63% for a 18x18cm marker from a range of 1.5m and the average detection rate for natural features was 99%. However, the natural feature detection quite often threw false positives as comparing a live video feed to a reference image that was in a completely different place had detection in 62% of the frames. In saying that the number of matches in the frames that had no relation to the original image was much lower so could be used to filter out the bad matches. Using these together in conjunction allowed for an almost continuous rate of detection in the range of 1.5-2.5m above the ground.

A. Future Research

Future research could be undertaken to improve the detection rate of the markers by looking at some type of adaptive thresholding. Through doing this an attempt could be made to adjust the thresholding depending on the lighting of the image. This could potentially lead to better detection of the markers.

Testing on an actual UAV would also be a good area to look into for future research. It would allow more accurate experiments to be undertaken, so that not all tests were done using just a camera that is being held in an attempt to imitate a UAV.

VII. REFERENCES

[1] AirVid, "20 great UAV applications areas for Drones," 14 September 2014. [Online]. Available: http://air-vid.com/wp/20-great-uav-applications-areas-drones/.

[2] G. Bensinger and J. Nicas, "Amazon Delivery Drones Where will they Land?," Digits, 30 November 2015. [Online]. Available: http://blogs.wsj.com/digits/2015/11/30/amazons-delivery-drones-where-will-they-land/. [Accessed 23 03 2016].

[3] A. Rosebrock, "Target Acquired: Finding Targets in a drone and quadcopter video streams using Python and OpenCV," PyImageSearch, 4 May 2015. [Online]. Available: http://www.pyimagesearch.com/2015/05/04/target-acquired-finding-targets-in-drone-and-quadcopter-video-streams-using-python-and-opencv/. [Accessed 23 March 2016].

[4] Unknown, "Detection of ArUco Markers," OpenCV, 2015. [Online]. Available: http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html#gsc.tab=0. [Accessed 23 March 2016].

[5] Y. Gui, P. Guo, H. Zhang, Z. Lei, X. Zhou, J. Du and Q. Yu, "Airborne Vision-Based Navigation Method for UAV Accuracy Landing Using Infrared Lamps," *Springer Journal of Intelligent Robotic Design,* vol. 72, no. 2, pp. 197-218, 2013.

[6] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," 2010.

[7] OpenCV, "ORB (Oriented FAST and Rotated BRIEF)," 2015. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html. [Accessed 23 April 2016].

[8] Wikipedia, "Features from accelerated segment test," Wikipedia, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Features_from_accelerated_segment_test. [Accessed 23 April 2016].

[9] E. Rosten, "FAST Corner Detection," 12 September 2015. [Online]. Available: http://www.edwardrosten.com/work/fast.html. [Accessed 23 April 2016].

[10] Unknown, [Online]. Available: http://www.edwardrosten.com/work/corner.png. [Accessed 23 April 2016].

[11] Intel, "Buy a Dev Kit," Intel, 2016. [Online]. Available: https://software.intel.com/en-us/realsense/devkit. [Accessed 28 April 2016].

[12] Intel, "SDK Design Guidelines," 2014. [Online]. Available: https://software.intel.com/sites/default/files/managed/27/50/Intel%20RealSense%20SDK%20Design%20Guidelines%20F200%20v2.pdf. [Accessed 28 April 2016].

[13] ArUco, "Calibration with ArUco and ChArUco," ArUco, 2016. [Online]. Available: https://github.com/Itseez/opencv_contrib/blob/master/modules/aruco/tutorials/aruco_calibration/aruco_calibration.markdown. [Accessed 8 April 2016].

[14] OpenCV, "Akaze and ORB planar tracking," 2016. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/tutorials/features2d/akaze_tracking/akaze_tracking.html. [Accessed 28 April 2016].

[15] OpenCV, "Feautre Matching," OpenCV, 2016. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html. [Accessed 28 April 2016].

[16] ArUco, "Detection of ArUco Markers," ArUco, 2016. [Online]. Available: https://github.com/Itseez/opencv_contrib/blob/master/modules/aruco/tutorials/aruco_detection/aruco_detection.markdown. [Accessed 8 April 2016].