3D pose estimation of known windows

1st Venkateshkrishna
Masters in Robotics
Worcester Polytechnic Institute
Worcester, MA 01609
vparsuram@wpi.edu

2nd Athithya, Lalith
Masters in Robotics
Worcester Polytechnic Institute
Worcester, MA 01609
lnavaneethakrishnan@wpi.edu
(using 1 late day)

3rd Gampa, Varun
Masters in Robotics
Worcester Polytechnic Institute
Worcester, MA 01609
vgampa@wpi.edu

***Abstract*—This project focuses on estimating the pose of the center of a window using the image data from a monocular camera. The monocular camera is the camera from the DJI Tello Nano drone. The dimensions of the window are known as well. For this project deep learning approach is used by which the corner points of the window in the image are detected. Further, given the dimensions of the window, the position of the corner points with respect to the center of the window in 3D is known. Using the PnP approach on the corner points in the image and knowledge of corner points in 3D with respect to the center of the image, the position of the center of the image is estimated with respect to the drone camera.**

I. Introduction

In this project, we describe the pipeline developed and used to detect the position of the window with respect to the drone camera. Firstly we describe how the dataset is generated with limited access to the real data. To work around this problem, we generated a simulated dataset using the Blender platform. An environment is created in Blender in which a few windows are imported that are of similar dimensions to the actual window and have the same markings. The generated training data consists of images containing Windows as inputs and the labels are the pixel coordinates of the corners of the closest window to the camera which is entirely in focus. Next, we augment the dataset with various backgrounds to generate a significantly large amount of training data. Next, we chose the efficient net-B0 as the deep learning network which could learn how to get the pixel coordinates of the closest window. After training is done, we calibrate the monocular camera of the drone to get its K matrix and distortion parameters. Finally, all of this is brought together, wherein the image is taken by the drone, and then the image is first rectified using the distortion parameters, the pixel coordinates of the corner points are estimated by the neural network, and finally, using the PnP function, which uses these image pixel coordinates, the K matrix and actual positions of the corner points with respect to window center, the position of the window center with respect to the camera is calculated.

II. Generation of dataset

Blender is an open-source 3D animation and rendering software, that can be used for generating photo-realistic renders of various scenes. We used Blender to set the scene in which three windows are in the scene. The dimensions of the windows are similar in scale to the dimensions of the actual window. The pattern on the border of the window is kept the same as the one in reality. The positions of the windows are random, with random orientations. The scene also has a camera, which captures windows. Further, in the scene, random spheres are added, which serve as possible occlusions to the windows. Finally, three light sources with randomly varying intensities are added. Using this setup, 3000 random scenes are generated with varying window positions, lighting conditions, occlusions, etc. In some scenes, only 1,2, or 3 windows are visible to the camera. The view seen by the camera is saved as a PNG file, note that it will have windows with black backgrounds. Next, a mask of the windows is saved as well. Along with this in a txt file, the pixel coordinates of the corners of the closest windows are saved. Now we load ten random backgrounds, a few of them have the laboratory background, and some of them have random noisy backgrounds. Now, we use any random mask that we generated for the corresponding image of windows along with pictures of the backgrounds. We mask the background with where the windows are and then add the windows to the masked background, this will simply add the window to the background. This is done for each image generated from Blender and for ten backgrounds. Hence this will generate 30000 training examples. Note that since we know which image we are adding to a background we already have the label corresponding to that.

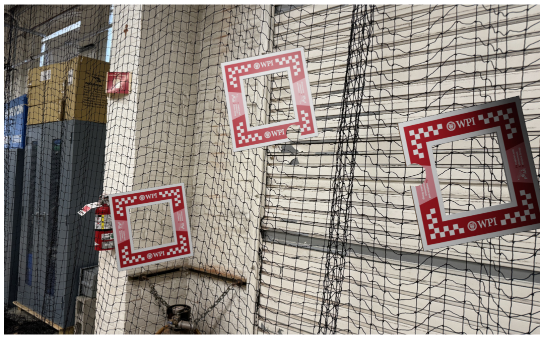
Fig. 1. Generated training sample by masking the background image based on the rendered image from blender and adding it to the background image

Fig. 2. Background image

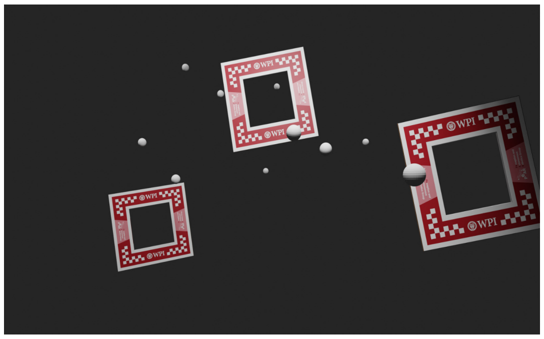

Fig. 3. Rendered scene from blender, in which occlusions are present

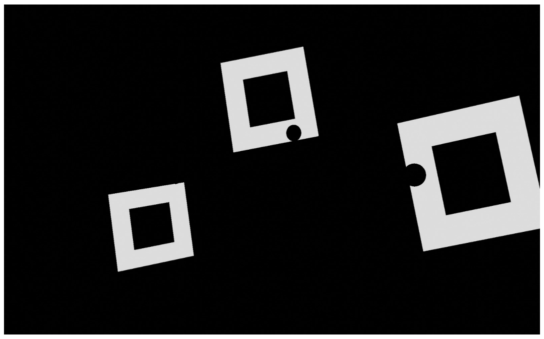

Fig. 4. mask of windows

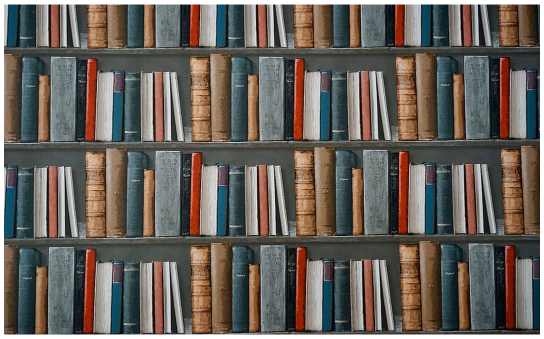

Fig. 5. Background image eg2

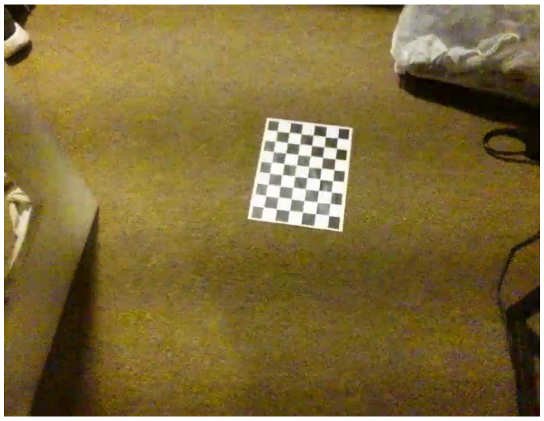

Fig. 6. Checker board pattern used for camera calibration

III. Using Efficient net-B0 for learning

Considering that we needed to evaluate a deep CNN on the Nvidia Jetson Orin Nano during runtime, we chose to select a relatively light network with a lesser number of parameters, but still complex enough to understand the data in various adverse conditions. For this reason, we chose to work with the efficient net-B0 model. Efficient net B0 has just about 4.5 million parameters compared to 11 million parameters of Resnet 18 but performs much better. Currently, it is considered the state-of-the-art CNN model, for classification, and it is used in the backbone of many architectures.

Hence we used the efficient net for generating 8 outputs which correspond to x and y coordinates of the 4 corner points. We used rmse loss function and starting with the pre-trained weights trained the entire network.

IV. Camera Calibration

We used the checkerboard pattern printed on an A4 size paper. We used a checkerboard pattern, with a 30 mm square size to generate the K matrix and the distortion parameters. This was done using the vision toolbox in Matlab. The checkerboard pattern used for calibration is shown in figure 6. The K matrix obtained is:

$$\begin{pmatrix} 940.5992 & 0 & 470.5372 \\ 0 & 956.1483 & 359.2646 \\ 001.0000 & & \end{pmatrix}$$

V. Pose Generation

To generate the pose of the window. Opencv's cv2.pnp is used. PnP or perspective n-point is a method to estimate the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. The camera pose consists of 6 degrees of freedom (DOF) which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world. For this to work efficiently 4 points are needed. In Opencv's implementation, we get the position of the world with respect to the camera frame.

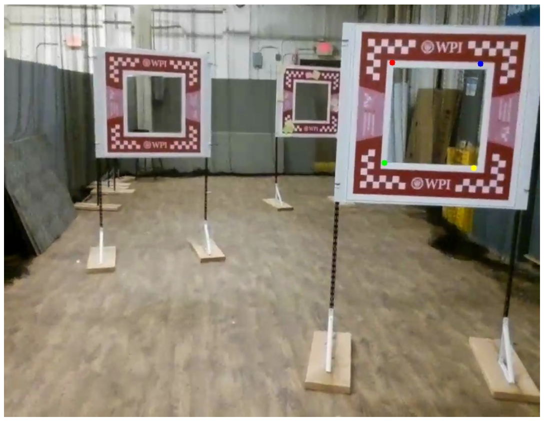

Fig. 7. Checker board pattern used for camera calibration

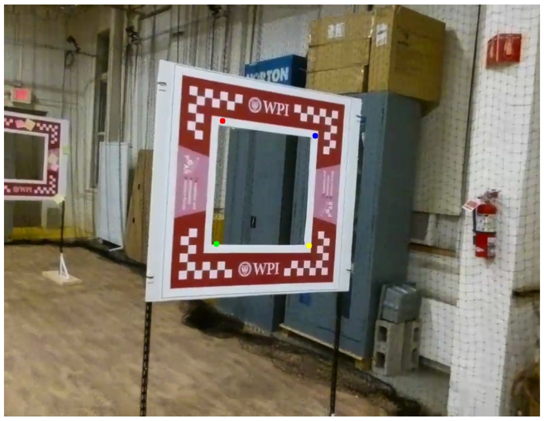

Fig. 8. Checker board pattern used for camera calibration

VI. Results

After all the elements of the pipeline are set, they are integrated so that, from the drone, the position of the center of the window can be estimated.

Camera detection's output is shown in figure7 and figure8. It can be seen that the neural network is able to determine the position of the corners, fairly accurately.

Finally, we estimated the pose of the window with respect to the camera using a tape measure and compared it against the estimate by PnP method. Note that, we know the coordinates of the four points with respect to the center of the window.

The table below shows the results versus the ground truth.

Results for figure 7:

	Ground Truth	PnP estimate
x	2.69	3.06
y	-1.29	-0.96
z	0.3	0.53

Results for figure 8:

	Ground Truth	PnP estimate
x	2.4	2.63
y	-0.2	-0.01
z	0.0	0.11

Note that ground truth would have some measurement errors, which couldn't be accounted for.

VII. Videos

A video of the footage containing the window and the neural network detecting the same is shown at videos

VIII. Conclusion

In this project, we built the pipeline for estimating the position of a window with a specific pattern from a single frame of video captured by the DJI Tello's onboard camera which has a resolution of 720x960. The results with respect to the ground are shared as well, which shows that the output is reasonably accurate, and can work with occlusions as well.

References

[1] Open CV's PnP: link
[2] Efficientnet: link