Course - 2D and 3D Scene Analysis
Evaluation of Computer Vision algorithms with Tello Edu

Irfan Dwiki Bhaswara - s2181738
Sriram Natarajan - s2122154
Shu yu - s2143321
Shunxin Wang - s2205327

December 27, 2020

Contents

List of Figures

List of Tables

1 Introduction

Autonomous indoor mapping using mobile robots or any mobile controlled drones, finds applications in situations where human access is prohibited due to space constraints or security reasons [5]. For applications involving emergencies, Unmanned Aerial Vehicles (UAV) serve as assistive tool in planning the best and secure operations for the crew. UAVs also find applications in regions with reduced GPS information and serve as powerful tools for mapping the surrounding area. Autonomous mapping have been extensively used in different fields such as mapping the interiors of buildings, abandoned mines etc. This has led to the rapid development of low cost, high resolution and versatile drones by several manufacturers [6]. Tello Edu as seen in Figure 1 is one such small quadcopter that features a Video positioning system, an onboard camera and is equipped with Failsafe protection. The onboard camera captures 5 megapixel images and streams 720p live video on a mobile device. This motivates the choice of this drone for analysis of indoor mapping. An autonomous drone effectively maps the environment and extracts various kinds of information like obstacles, fire, free space for navigation etc. The work aims to develop an effective indoor mapping with Tello Edu using computer vision and machine learning techniques.

The report is organized into subsections as follows: Section 2 describes several algorithms studied namely: Structure From Motion (SFM), Autonomous navigation based on object tracking, Object detection and segmentation. Section 3 describes the outcome of the algorithms and Section 4 details about the inferences from this study specifically the advantages and disadvantages of various approaches. Section 5 discusses the overall understandings of the experiment and proposes scope for further research in this regard.

Figure 1: Tello Edu

2 Methodology

The evaluation of algorithms with Tello motivates the need to understand the specifications of the drone as described in Table 1. The detailed description regarding the construction and modes of operation have been described in the User manual [7].

Parameter	Value
Weight	87 grams
Dimensions	98x92x41 millimeter
Maximum Flight Distance	100 meter
Max speed	8 meters/sec
Max Flight Time	13 minutes
Max Flight Height	30 meters
Battery	1.1Ah/3.8V
Image	5MP (2592x1936)
Field of View	82.6 degrees
Max Flight Height	30 meters

Table 1: Specifications of Tello Edu

2.1 Structure From Motion

Structure From Motion (SFM) refers to the technique of reconstructing 3D structure of a scene or an object using series of 2D images. SFM produces a point cloud based 3D model. The application of 3D modelling for indoor environments has been an emerging topic. The objective here is to create a 3D representation of the environment. The study involves generation of a 3D point cloud of the track along which the drone navigates. The 2D images required for the mapping is captured using the on board camera of the drone. The approaches of SFM tried out include Meshroom, VisualSFM, OpenMVG, and MATLAB. The estimation of Fundamental Matrix (F) without camera parameters leads to an uncalibrated SFM. Hence, the drone camera has to be calibrated using MATLAB.

2.1.1 Camera Calibration

The Tello Edu has not published the camera parameters for its on-board camera. Hence the calibration pattern such as a checkerboard has been used to generate pattern images and the MATLAB calibration application has been used for generating the camera parameters. The estimated parameters are described in Table 2. The re-projection error of the calibration is 0.91 pixels as seen in Figure 2

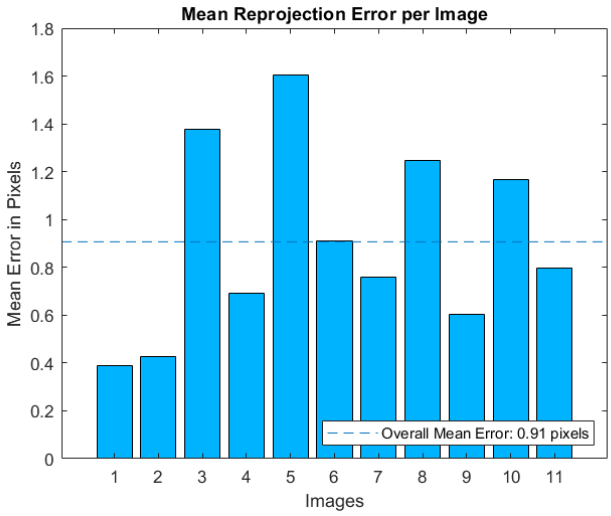


Figure 2: Reprojection Error

Parameter	Value
Number of columns (width)	960
Number of rows (height)	720
Focal length per pixel width (f_x)	1027.7
Focal length per pixel height (f_y)	1029.9
Image column center (c_x)	504.155
Image row center (c_y)	359.356

Table 2: Tello Edu camera intrinsic parameters

2.1.2 Meshroom

Meshroom is a free, open-source 3D Reconstruction Software based on AliceVision Photogrammetric Computer Vision framework [8]. Meshroom is available as an executable for Windows that enables to start off the application. The complete view of the tool could be visualized in Figure 3. The interface shows an Image viewer and 3D viewer. The application provides interface for Live reconstruction as well. The option could be enabled in View->LiveReconstruction and the folder to which the images are dumped could be set as the input. The tool automatically computes the intrinsic parameters of the camera based on the input images. It does not provide an option for the user to explicitly specify the camera parameters. The offline and online (Live Reconstruction) techniques have been evaluated and observations are discussed.

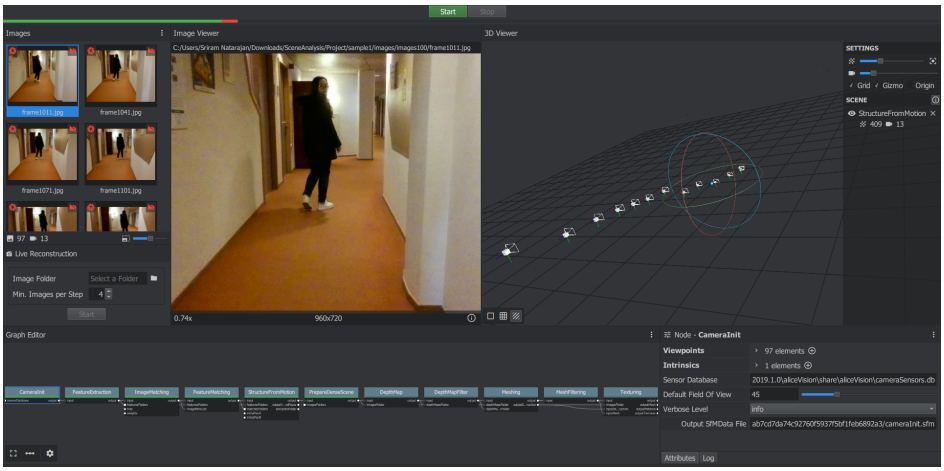


Figure 3: Meshroom sparse reconstruction

2.1.3 VisualSFM

Visual Structure From Motion is a GUI based application for 3D reconstruction [9]. The tool is available as an executable for Windows and the reconstruction is fast as it exploits the multicore parallelism for feature detection, matching and bundle adjustment. The detailed description of the usage and tutorials are available online. The tool provides several options for exploring the features matches in the inputs as seen in Figure 4 and 5. The VisualSFM uses SIFT based features and uniformity in scene may lead to wrong feature matches. The tool automatically calculates the camera parameters and generates feature detection files as [name].sift and [name].mat for each of the images. The procedure to run the GUI based SFM is:

- Launch VisualSFM.exe and load the input images.

- Computer Pairwise missing matches
- Run Sparse Reconstruction.
- Run Dense Reconstruction.

It is also possible to run this tool in command line using the command seen below.

VisualSFM sfm [options] input output.nvm [user_data_path]

where input - specifies the folder containing images and output.nvm is the output file that is generated in the current folder.

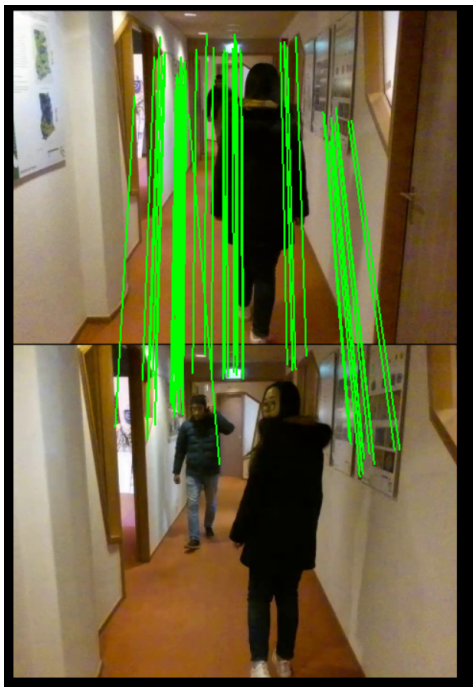


Figure 4: Features points matched correctly

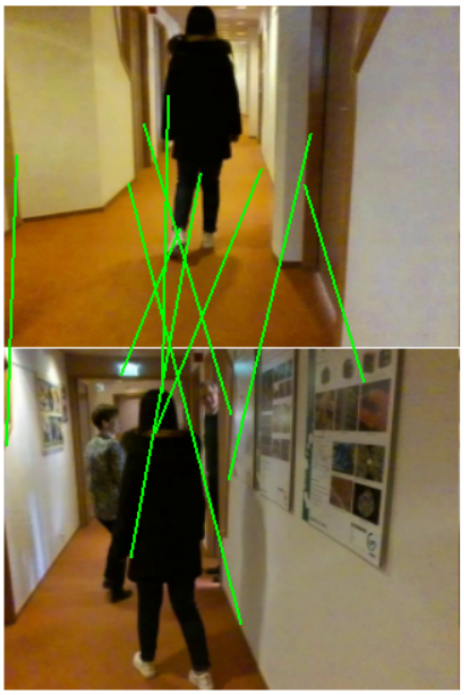


Figure 5: Incorrect feature matches

The tool uses Yasutaka Furukawa's PMVS/CMVS tool for Dense reconstruction [10] and the libraries are pre-built for Windows and available at [11] and have to be downloaded and included in the VisualSFM folder in order to run Dense reconstruction. The VisualSFM automatically estimates the camera parameters and is one among the most effective tools available online for SFM.

2.1.4 OpenMVG

OpenMVG (Open Multiple View Geometry) is a C/C++ based library for SFM [12]. The library has been built with Windows using Visual Studio and tested with images offline. The intrinsic parameters of the camera has to be explicitly mentioned to run the SFM pipeline in OpenMVG. The camera parameters generated in Section 2.1.1 has been used in this case. The output of the OpenMVG is generated as a .ply file and a python script has been used to plot the ply file [13]. The step to run are the built library are :

- Set the camera parameters in openMVG/src/software/SfM/SfM_GlobalPipeline.py

- Run python SfM_GlobalPipeline.py input_folder output_folder. The input_folder is the path to the folder containing input images and output_folder refers to the path where the output files are generated.
- To visualize the output run python plot.py output.ply. The plot.py file is obtained from [13] and the output.ply will be generated from the previous step.

2.1.5 MATLAB

MATLAB provides functionalities for estimating the 3D structure of a scene based on a set of 2D views. The SFM pipeline in matlab could be run directly as m script from [14]. The steps to run the 3D reconstruction in MATLAB are:

- Set the folder imageDir to input folder containing images.
- Load the camera parameters as a .mat file generated from precedure described in Section 2.1.1

2.2 Tracking

We apply OpenCV tracker to achieve object tracking. By using OpenCV, we don't need to detect object in each frame to achieve tracking, we select the object in the first frame, and then the OpenCV tracker will handle the subsequent frames. This tracker leads to a faster and more efficient object tracking pipeline. There are eight different trackers we can use in OpenCV and comparison has been listed in Table 3.

After comparing the performance of all these trackers, we decide to use KCF(Kernelized Correlation filters) in the end because even though its accuracy is slightly lower than other trackers, but it is faster, which is very important in dealing with the video in real time. The main point in our tracking algorithm is a function called *update*. In the first frame, we select a ROI(region of interest) by using a bounding box. After that, we only pass the frame to the function *update*, than it will locate the object's new position and return a *success* boolean. If the *update* successes, a new bounding box will be drawn on the new frame. However, if the *update* fails, the drone will stop moving and you need to select a new ROI on the currect frame again.

In order to make the drone keep on following the object, we set a safe region. The drone keep the object in the safe region by adjusting its speed to all the direction. We use a distance vector here, which contains xdistance, ydistance and size. The distance vector can be obtained by subtracting the bounding box vector from the frame vector. When the xdistance is large, it means our target is too far to the right, the drone needs to add a speed to the left. If the xdistancce is too small, it needs to add a speed to the right. Similarly, ydistance controls the height of the drone, the drone moves up and down according to the ydistance. Also, the drone moves back and forward by comparing the size in distance vector.

2.3 Object Detection

Object detection in this project is based on deep learning and OpenCV. And also it can works real-time with video stream. In the meantime, object detection can not only demonstrate what is in the image but also where is the object as well.

In order to construct a super-fast, real-time object detection model for resource constrained devices, we use the model in which *Single Shot Detectors*(SSD) and *MobileNets* are combined. *Single Shot Detectors* uses only a single deep neural network to detect the objects. And in the process of prediction, the network generates scores for each object category. SSD is easy to train and to be integrated into systems that need a detection component. The reason to apply *MobileNets* is that it is an efficient neural network especially for resource constrained devices such as our smart phone. When building obeject detection network, the existing network architectures can be very large, which is not suitable

Tracker name	Description
BOOSTING Tracker	Slow and doesn't work very well. This classifier needs to be trained at runtime with positive and negative examples of the object [15].
MIL Tracker	Better accuracy than BOOSTING Tracker but does a poor job of reporting failure.
KCF Tracker	Kernelized Correlation Filters. It is faster than BOOSTING and MIL, but it does not handle full occlusion well. This tracker utilizes the fact that the multiple positive samples in the MIL tracker have large overlapping regions. This overlapping region leads to some mathematical properties that is exploited by this tracker to make the tracking faster and more accurate at the same time [15].
CSRT Tracker	The tracker tends to be more accurate than KCF but slightly slower. It uses the spatial reliability map for adjusting the filter support to the part of the selected region from the frame for tracking [15].
MedianFlow Tracker	It does a nice job reporting failures; however, if there is too large of a jump in motion, such as fast moving objects, or objects that change quickly in their appearance, the model will fail. This tracker tracks the object in forward and backward directions in time and measures the discrepancies between these two trajectories [15].
TLD Tracker	The TLD tracker was incredibly prone to false-positives. This tracker decomposes the long term tracking task into three components: tracking, learning, and detection. The tracker follows the object from frame to frame. The detector localizes all appearances that have been observed so far and corrects the tracker if necessary. The learning estimates detector's errors and updates it to avoid these errors in the future [15].
MOSSE Tracker	It is very fast tracker but not as accurate as CSRT or KCF. It uses adaptive correlation for object tracking which produces stable correlation filters when initialized using a single frame [15].
GOTURN Tracker	The only deep learning-based object detector included in OpenCV. It takes two cropped frame as input where the first frame is used for the search region and the second frame is used for searching of what object to be tracked [15].

Table 3: description for eight trackers

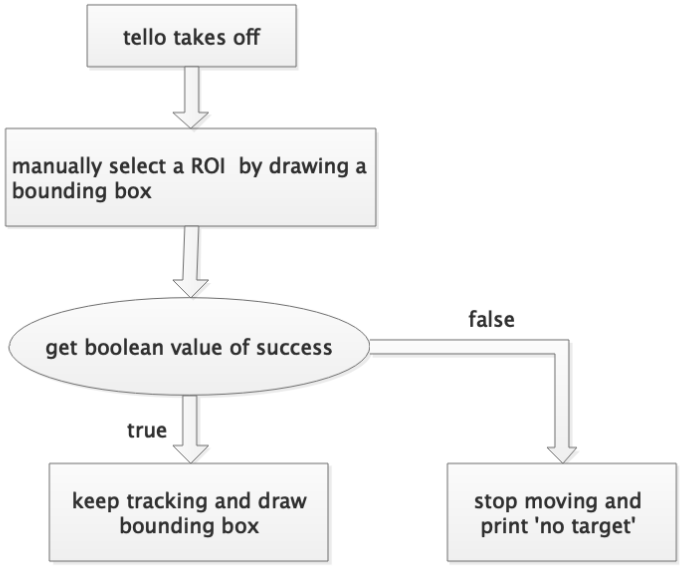


Figure 6: Tracking Procedure

for some devices without large computational capability. Different from conventional CNN, *MobileNets* uses depthwise separable convolution, as shown in figure 8 [2].

Before detecting the object, the dataset used to train such a model needs to be prepared. The images contained include two categories. One is *Person*[16], and another is *door*[17]. And the labeling of these two categories is done manually, by a software called *LabelImg*, which is a graphical image annotation tool[1]. And the process of label is shown in figure 7.

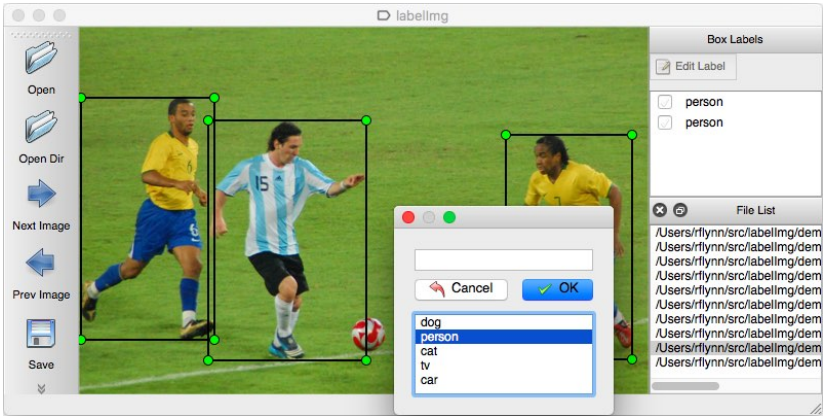


Figure 7: Process of labeling [1]

Since the dataset has been well prepared, we start to train the object detection model. The model is combined with *SSD* and *MobileNets*. The detailed procedure is described as follow:

- Install TensorFlow-GPU 1.5.
- Set up TensorFlow Directory and Anaconda Virtual Environment .

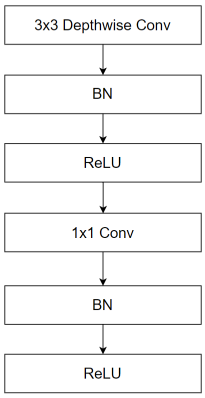


Figure 8: Depthwise separable convolution with depthwise and pointwise layers followed by batch normalization and ReLU [2]

Downloaded TensorFlow Object Detection API repository from GitHub [18];

Download the SSD-mobilenet-inception-V2 model;

Set up new Anaconda virtual environment;

Configure PYTHONPATH environment variable;

Compile Protobufs and run setup.py;

Test TensorFlow setup to verify it works.

- Run the Training [19].
- Use Trained Object Detection Classifier by the drone.

In the model we used, *SSD* and *MobileNet* are combined. And *MobileNet* applies a single filter (figure 9a) to each input channel, followed by a 1×1 convolution (figure 9b) to combine the outputs of depthwise filters 9a. The standard convolution (figure 10) combines the two steps into one step, while depthwise convolution separates it. It can help reduce the computational cost and the model size. In depthwise separable convolution, the process of convolution is splitted into two stages:

- 3×3 depthwise convolution
- 1×1 pointwise convolution

(a) Depthwise filters

(b) Pointwise filters

Figure 9: Depthwise Convolution

2.4 Segmentation

We use Fully Convolutional Network (FCN) [3] for segmentation part. FCN is basically a normal Convolutional Neural Network (CNN) where the last of fully connected layer is replaced with deconvolution layer. It means that at the final output layer, the height and the width will have the same

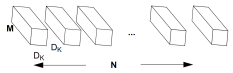


Figure 10: Standard Convolution

size with the input. Figure 11 shows the FCN configuration. The problem that occurs in FCN is in the low resolution image output because many parts in the FCN layers are downscaled into small size. This problems can be handled by using skip connection on the regular network as shown in Figure 12. The result after doing skip connection can be seen in Figure 13. We can see that from Figure 13 when there is a skip connection, it gives good segmentation than without using a skip connection. Here in our project for segmentation, we use FCN with 2 skips (FCN-8).

For doing the project, we used the dataset from the pretrained network [20]. This dataset contains 12764 images for training and 12764 images for label. To make it simple when training and testing the segmentation, in the preprocessing part, we adjust the labels into green color channel. In addition, we also change the size of the image into 80 x 160. Figure 14 shows the preprocessing part.

In training part, we use 20% for testing and 80% for training. We also use batch size and epoch equal to 128 and 10, respectively. The Adam optimizer is used in this training process. This training process will have result in trained weight called "segment_model.h5". The block diagram of training process is shown in Figure 15

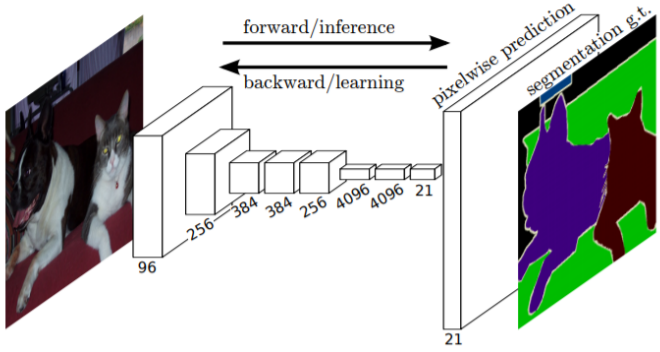


Figure 11: FCN configuration [3]

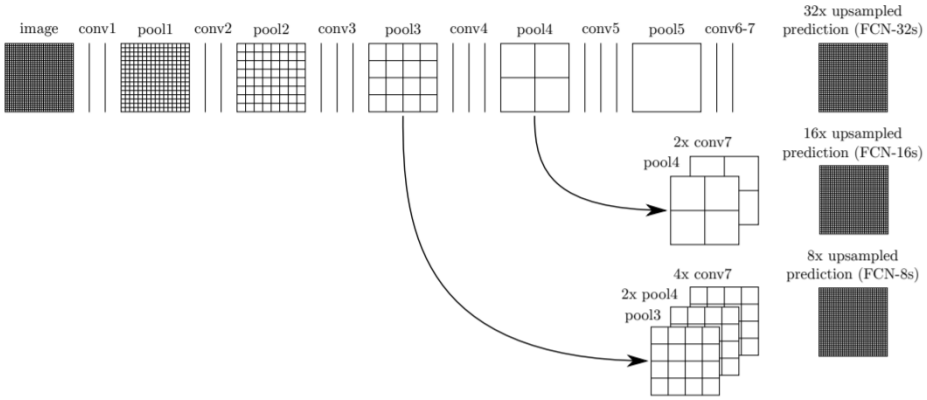


Figure 12: FCN Skip Connection [3]

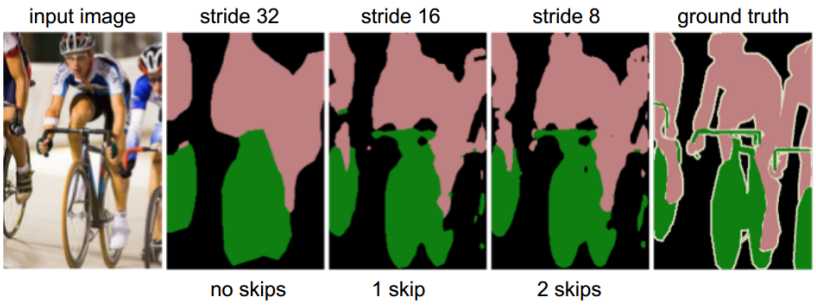


Figure 13: FCN Skip Connection Result [4]

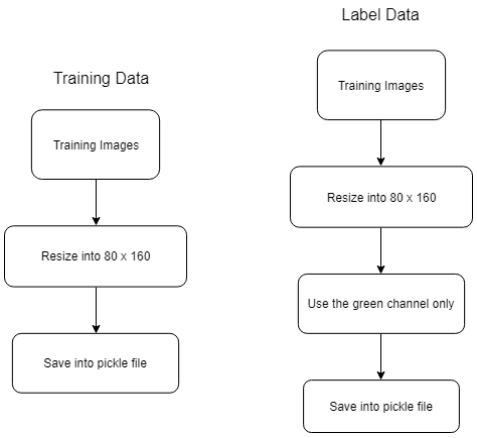


Figure 14: Preprocessing Step

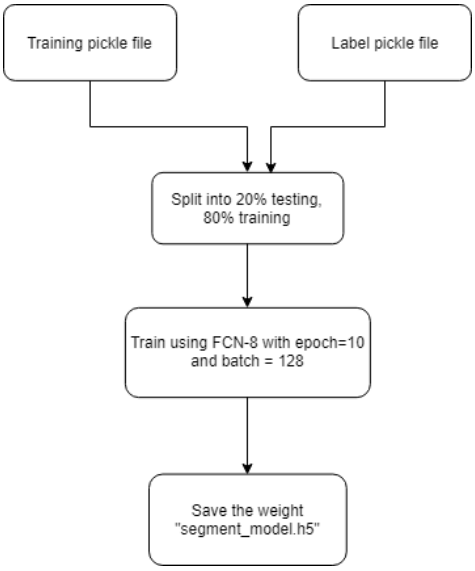


Figure 15: Training Step

3 Results

3.1 SFM

The reconstruction generated using VisualSFM is better than the other tools. The dense 3D point clouds are as seen in Figure 16 and Figure 17 in two different scales. The best reconstruction obtained in comparison with other tools is with VisualSFM.

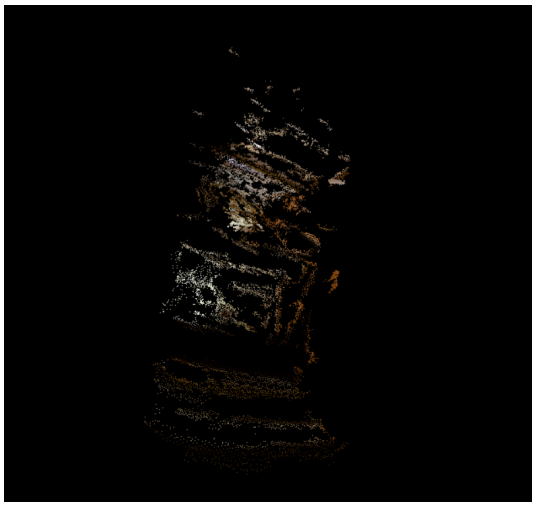


Figure 16: Reconstruction using VisualSFM

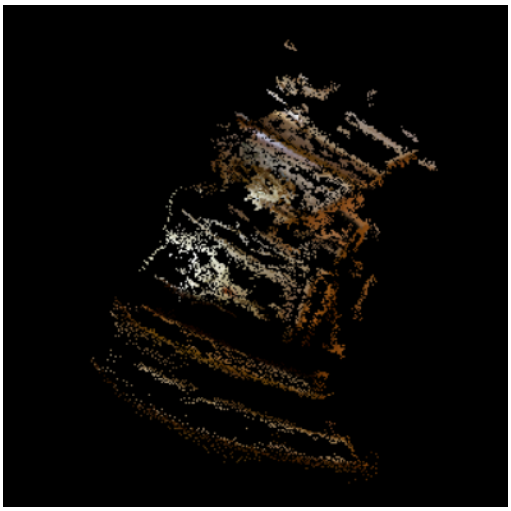


Figure 17: Reconstruction using VisualSFM

The reconstruction generated with MATLAB based SFM is as seen in Figure 19 and it could be inferred that very few points have been reconstructed. The SFM tool chain uses SURF for feature extraction and generates output only if there are enough match points between the images under test else throws an error. The images chosen also affect the calculation of Essential matrix as MATLAB throws an error if there is not sufficient information in the images for calculating the same.

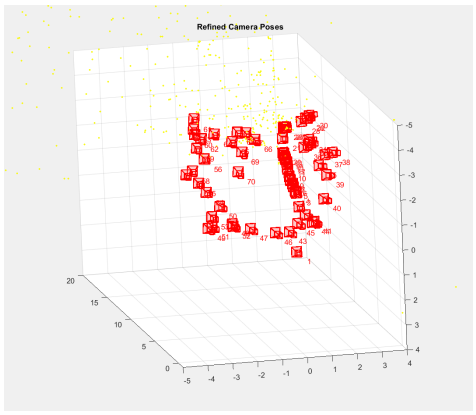


Figure 18: Refined camera poses generated from MATLAB based SFM

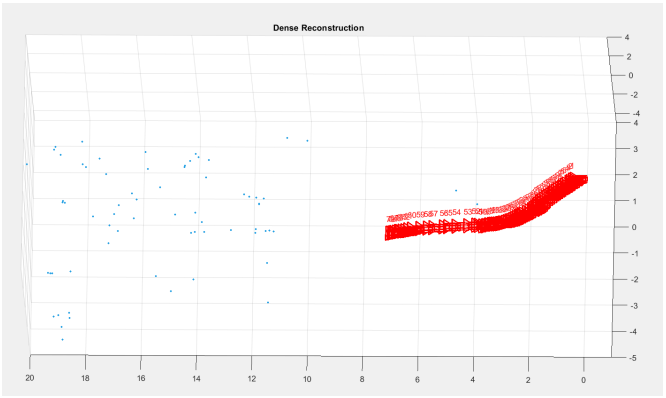


Figure 19: Dense reconstruction using MATLAB based SFM

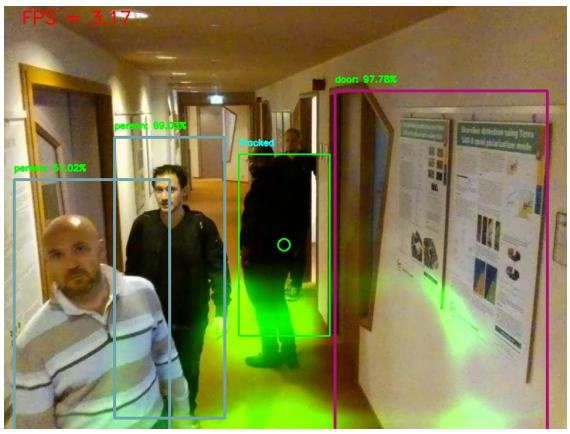
The reconstruction with other libraries such as Meshroom and OpenMVG is worse and for most of the instances it doesn't generate a point cloud at all. It could be attributed to the fact the algorithms couldn't detect feature points effectively or due to the nature of image sequence.

3.2 Object Detection, Tracking and Segmentation

In the following figures (Figure 20, Figure 21, Figure 22) are the result of the drone. Figure 20 shows the result of the drone in tracking, detecting, and segmenting. The green bounding box shows the tracking object, the blue bounding box shows the person object, the purple bounding box detects the door object, and the green color shows the segmentation part which is in our case the corridor floor.

The percentage above person and door bounding box is the percentage confidence in the detection. The higher the value, the more accurate in the prediction of an object.

Figure 21 represents the wrong object detection result. We can see that the door bounding box does not detect the door object, instead it detects the person object. In addition, Figure 22 depicts the image of the drone when the drone looses the tracking of an object. We see that the tracking bounding box (green bounding box) disappears from the image results of the drone.



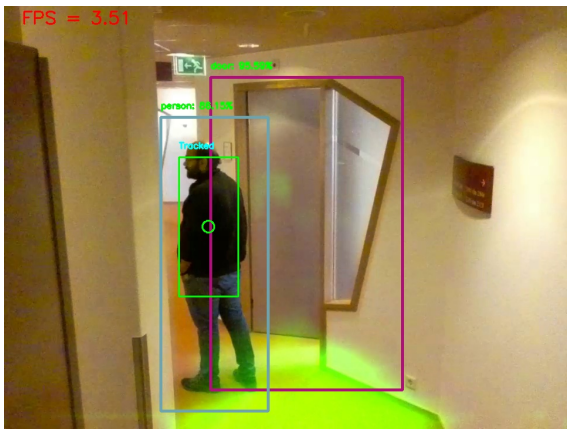


Figure 20: Result of Detection, Tracking, and Segmentation

Figure 21: Wrong result

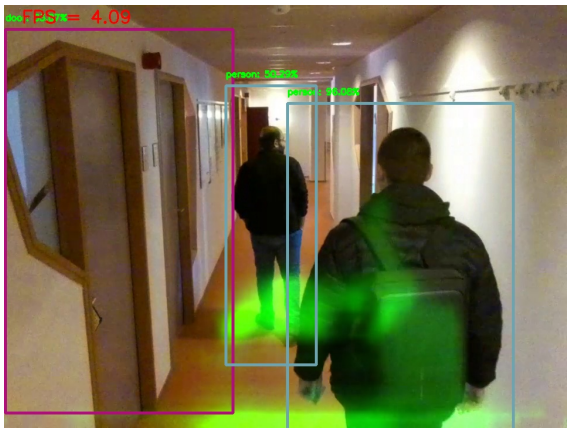


Figure 22: Loose tracking

4 Discussion

4.1 SFM

The scenario under study poses many limitations because surfaces such as white wall or uniform objects do not generate many feature points. The available feature points could also be matched incorrectly because of uniformity in objects as in Figure 5. The study has led to many inferences with the tools analysed for SFM and have been tabulated in Table 4. It could be inferred from the Table 4 that VisualSFM is better than other options for the indoor mapping.

Tool	Camera Parameters	Live Reconstruction	Inference
Meshroom	Intrinsics are calculated by the tool based on the input images	Possible via GUI and Command line	Reconstruction is heavily dependent on the parallax between images, very small movement of objects in sequence is expected for generating a point cloud
VisualSFM	Intrinsics are calculated by the tool based on the input image	Possible by integrating command line arguments with Python environment	Generates a stable point cloud and too little motion between the images leads to no pairwise matching
OpenMVG	Intrinsics have to be specified by user	Possible by integration with python environment	Difficult to generate a point cloud as it is highly dependent on the images.
MATLAB	Intrinsics have to externally specified	Possible provided the interface between Python based Tello library and MATLAB exists	Reconstruction is also dependent on images, few match pairs or choice of images may lead to errors.

Table 4: Comparison between the SFM libraries analysed

4.2 Object Detection, Tracking and Segmentation

4.2.1 Object Tracking

The object tracking in our system looks satisfying. It can track the person very stable. However, it has an issue. When the object moves too fast, the tracking will loose the object as it can be seen in Figure 22. This is because the tracking part is based on the initial shape position of the object. If the object changes its shape (only half part which can be seen), the tracking will disappear. We can give some improvements for the tracking method, for example combining the KCF tracking with the kalman filter. Or we can use directly particle filter to track the object.

4.2.2 Object Detection

In the object detection part, we were able to detect the door and person object. However, sometimes the detection is not quite good as it is shown in Figure 21. This is because we only use 200 image of doors and persons. We can increase the accuracy of the detection by adding some training images. But, if we try to increase the training data, it will take too much time to train the network. Moreover, increasing in the training file and time will also have an impact in the increase of the system memory for training which is now is not possible in our system.

4.2.3 Object Segmentation

The segmentation part that we did is corridor segmentation. As we can see from Figure **??**, the green color reflects the segmentation of corridor path. The result of segmentation is quite good although sometimes some of the parts are not segmented well as shown in Figure 21. This is due to the training image. If we add more combinations about the corridor path, we might get a better result in the segmentation.

5 Conclusion

The VisualSFM has given a better reconstruction than other tools analysed. The work done with reconstruction is offline. The work could further be improved by stabilizing the reconstruction and running an online reconstruction by integrating the command line versions of the tools with the Python environment of the Tello Edu. Tracking, detection, and segmentation for the drone have achieved a good result. Although, sometimes it looses the tracking, detects the wrong object, and segments only some parts of the corridor, but it is able to run quite smooth in the real time. The detection and segmentation could perform better by training the model with more label data. In addition, combination of the KCF tracker that we used here with kalman filter can also be done in the future to increase the tracker accuracy.

References

[1] Tzutalin, "Labelimg," 2015. [Online]. Available: https://github.com/tzutalin/labelImg

[2] A. Rosebrock, "Object detection with deep learning and opencv," 2017. [Online]. Available: https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/

[3] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2016. [Online]. Available: https://arxiv.org/abs/1605.06211

[4] L. A. dos Santos, "Image segmentation," 2016. [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_segmentation.html

[5] F. L. C. A. d. C. David Gonzalez Arjona, Alberto Sanchez and J. Garrido, "Simplified occupancy grid indoor mapping optimized for low-cost robots," *ISPRS International Journal of Geo-Information*, vol. 2, pp 959 - 977, 2013.

[6] I. Colomina and P. Molina, "Unmanned aerial systems for photogrammetry and remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 92, pp. 79-97, 2014.

[7] RYZE, "Tello user manual," 2018. [Online]. Available: https://dl-cdn.ryzerobotics.com/downloads/Tello/20180212/Tello+User+Manual+v1.0_EN_2.12.pdf

[8] AliceVision, "Photogrammetric computer vision framework," 2019. [Online]. Available: https://alicevision.github.io/

[9] C. Wu, "A visual structure from motion system." [Online]. Available: http://ccwu.me/vsfm/

[10] Y. Furukawa, "Clustering views for multi-view stereo (cmvs)." [Online]. Available: https://www.di.ens.fr/cmvs/

[11] P. Moulon, "Cmvs-pmvs." [Online]. Available: https://github.com/pmoulon/CMVS-PMVS

[12] "Open multiple view geometry." [Online]. Available: https://github.com/openMVG/openMVG

[13] D. Ranjan, "Python-plyfile." [Online]. Available: https://github.com/dranjan/python-plyfile/tree/master/examples

[14] MATLAB, "Structure from motion from multiple views." [Online]. Available: https://nl.mathworks.com/help/vision/examples/structure-from-motion-from-multiple-views.html

[15] S. Mallick, "Object tracking using opencv (c++/python)," 2018. [Online]. Available: https://www.learnopencv.com/goturn-deep-learning-based-object-tracking/

[16] N. Dalal, "Inria person dataset," 2005. [Online]. Available: http://pascal.inrialpes.fr/data/human/

[17] M. Clinic, "Mcindoor20000," 2018. [Online]. Available: https://github.com/bircatmcri/MCIndoor20000

[18] D. Dao, "Tensorflow models," 2004. [Online]. Available: https://github.com/tensorflow/models

[19] EdjeElectronics, "How to train an object detection classifier for multiple objects using tensorflow (gpu) on windows 10." [Online]. Available: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10

[20] M. Virgo, "Mlnd-capstone," 2018. [Online]. Available: https://github.com/mvirgo/MLND-Capstone