Bird's Eye View

Barak Gorodissky barak.gorodissky@mail.mcgill.ca
Daniel Duenias daniel.duenias@mail.mcgill.ca
Julie Alhosh julie.alhosh@mail.mcgill.ca
Professor Kaleem Siddiqi

1 Introduction

The problem of obtaining an overhead view, also known as bird's eye view (BEV), is a popular problem in computer vision due to its usefulness. The idea of transforming an image to an overhead view of a scene is utilized in applications that require a complete understanding of the surroundings. One such application is in the field of Advanced Driving Assistance Systems (ADAS) where the private driver uses the overhead view in tasks such as parking, navigating, and more.
Another application is remote-operated machines in worksites, where the operator operates a machine from a distance using a camera's feed thus removing the risks involved in operating such a machine on site. Cameras positioned around the site are used in the composition of a bird's eye view of the site. This global view will serve as a virtual map used to help the operator adequately specify the commands to be sent to the vehicle under his control [2].
Additionally, an overhead view provides a better understanding of the scene geometry, and measurements can be taken directly from the image. Hence, it can be used as a pre-processing step for many other computer vision tasks like object detection and tracking.

2 Background

The common setting used to define this problem is assuming the scene is a 3D plane with the camera positioned above it with some tilt, pan, and roll angles. One wants to transform the image as if it was taken from right above the plane, from a bird's eye view. Due to the importance of this computer

vision problem, many papers have discussed and suggested solutions including [1, 4, 3]. Most of the solutions include using a convolutional neural network (CNN) [1, 4] or multi-camera images [3]. One interesting approach was described in [1] which involves using a convolutional neural network (CNN) to estimate geometric entities, namely the vertical vanishing point and the horizon line, then using these entities they calculate the homography to obtain an overhead view.
Another approach described here [2] is calculating a homography that transforms image plane points into 3D plane points. Under the assumptions made here [2] the homography is calculated using pre-calibrated parameters, camera's height, tilt angle, and pan angle.
In our approach and under our assumptions of the scene [3.1], we detect the vertical vanishing point using classical computer vision methods (Canny edge detector, Hough transform) and then using similar ideas to [1], we estimate the homography needed to transform the image plane to an image plane taken by a virtual camera that is directly above the plane, i.e. overhead view.

3 Methods

In the following section, we explain the three main steps we followed to achieve our goal[1] (See **Figure 6**). For this project, we assume that the relation between the plane of one image to the plane of the next image in our dataset is of translation and rotation, specifically tilts (rotation around the shared X-axis) and minor pans (Please, see our dataset **Figure 7**). We first introduce the notation we use and we explain the calibration step.

3.1 Notation

In this report, we use K for denoting a camera's intrinsic matrix.

$$K = \begin{bmatrix} fm_x & 0 & p_x \\ 0 & fm_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Where f is the focal length, m_x, m_y are pixels per unit length, (p_x, p_y) is the location of the principal point in the image plane. We assume that $(p_x, p_y) = (\frac{w}{2}, \frac{h}{2})$ for w, h the width and height of the image respectively.

[1]Please note that we used python and a few useful libraries to implement our code, including OpenCV, scikit-image, NumPy, and Matplotlib.

This report builds on the concepts shown in class where in the case of two cameras viewing one scene plane, there exists a homography H which transforms one image plane points to the other image plane points. Using that, we get a mapping between the corresponding pixel points by:

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = K_2 H_2 H_1^{-1} K_1^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{2}$$

Where H_1, H_2 are the homographies transforming points on the scene plane to the respective image planes. We assume that it is actually one camera thus $K_1 = K_2 = K$. Thus 3 boils down to

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = K H_2 H_1^{-1} K^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{3}$$

To add robustness to the method, so it can be used on images that do not have the K matrix, we assumed that f and m_x, m_y are unknown and there is a need for estimating them to create the K matrix. Assuming $fm_x = fm_y = \alpha$ we can write K as,

$$K = \begin{bmatrix} \alpha & 0 & width/2 \\ 0 & \alpha & hight/2 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

3.2 Calibration Step

There is a need to estimate the K matrix, more specifically, the α parameter. For the estimation of α, we need some prior knowledge about at least one image. Consider the settings described in **Figure 3**. Assuming we know the exact tilt angle θ of one image (see **Figure 1**), we detect its vanishing point, as will be mentioned in step A (3.3), and use equation (8). With the known θ, h and the y coordinate of the vanishing point, we can directly calculate α. This estimation is good enough for our purposes.

3.3 Step A: Detecting vanishing point

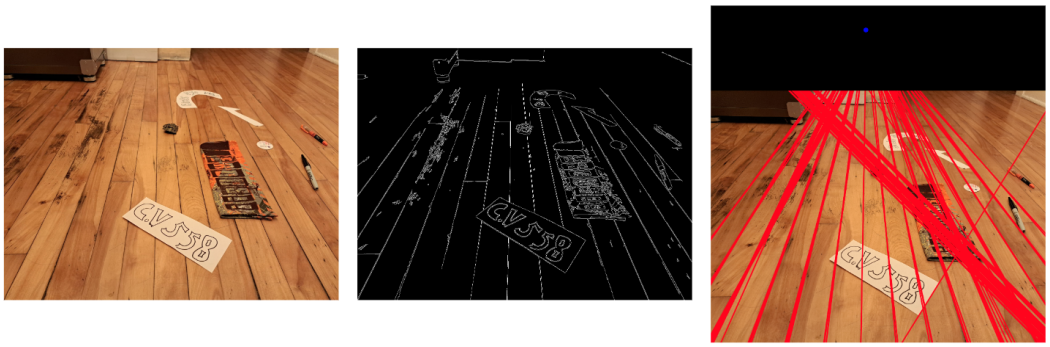
We would like to locate and use vanishing points, therefore, we need to assume that we have a 1-point or 2-point perspective scene and that the camera's X-axis is parallel to the horizon line. Moreover, we assume that the scene contains at least two parallel lines that help detect and locate the vanishing

Figure 1: The image used for calibration ($\theta = 45°$).

point. We also assume that the scene should be relatively flat without tall or high structures. Meaning that it should be close to a 3D plane. For each frame in our data set, we start by using OpenCV's Canny edge detector to obtain a set of edges (See **Figure 2(b)**). We want to filter those edges to get a set of edges that represents the parallel lines in the "ground plane" that should intersect at the vanishing point. To do that, we use Hough transform to fit the lines and we filter out the ones that have an angle $\theta < 45°$ or $\theta > 135°$ keeping the somewhat vertical lines and disregarding the somewhat horizontal lines in order to have a robust approach for having both 1-point and 2-point perspective scenes. This step is necessary to remove the lines that are somewhat horizontal and would have a different vanishing point than the one we are trying to detect. The lines we get at this point are still noisy. Therefore, we use RANSAC to ignore outliers and approximate the vanishing point, (x_v, y_v), where the maximum possible number of lines intersect (See **Figure 2(c)**). In this figure, we observe that the estimation of the vanishing point looks good for the human eye even though there are many outlier lines. This demonstrates the robustness of the RANSAC algorithm against outliers. In our RANSAC implementation, we are randomly choosing 2 lines and finding the point of their intersection (candidate vanishing point). The consensus set contains the lines that are close enough to that point (distance from a point to a line).

3.4 Step B: Calculating and applying the homography

For the purpose of approximating a homography, we need to make a few assumptions about the images. First, we assume that the focal length of the camera is known since we need this intrinsic property to approximate a homography. We also assume that the camera is far enough from the scene plane



(a) (b) (c)

Figure 2: (Step A) **(a)** Original image. **(b)** Result of Canny edge detector. **(c)** Result of our vanishing point detection algorithm (vanishing point location in blue).

to guarantee the existence of a homography.

Now, given the vanishing point $v = (v_x, v_y)$ obtained from 3.3, we compute the homography based on the setup in **Figure 3**.

$$\begin{bmatrix} \omega x \\ \omega y \\ \omega \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & 0 & \frac{w}{2} \\ 0 & \alpha & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & sin\theta \\ 0 & -sin\theta & cos\theta \end{bmatrix}}_{\text{camera's rotation}} \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & q \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}}_{\text{point on the 3D plane}} = H \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \tag{5}$$

where w, h are the width and the height of the image respectively. q is the height of the camera with respect to the ground plane. Assuming that $q \neq 0$, H is invertible and we get the following equation:

$$\begin{bmatrix} \omega s \\ \omega t \\ \omega \end{bmatrix} = H^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} — & H_1^{-1} & — \\ — & H_2^{-1} & — \\ — & H_3^{-1} & — \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6}$$

We know that the vanishing point v is in the horizon line of the image which is the projection of the vector $\begin{bmatrix} s \\ t \\ 0 \end{bmatrix}$ for any arbitrary (s, t). Therefore, the following should hold for v:

$$\begin{bmatrix} s \\ t \\ 0 \end{bmatrix} = H^{-1} \begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix} \Rightarrow H_3^{-1} \cdot (v_x, v_y, 1) = 0 \tag{7}$$

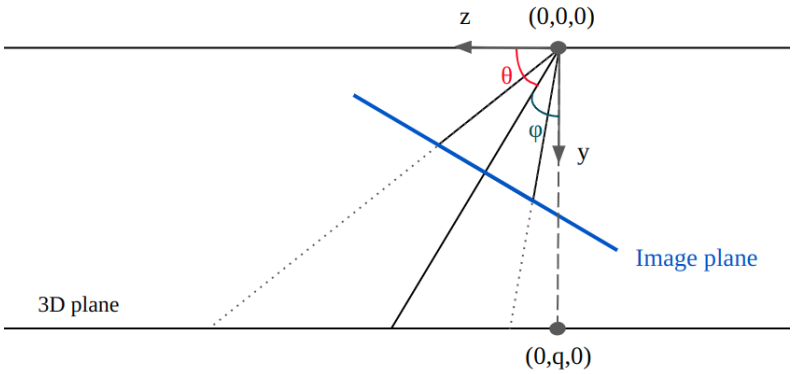


Figure 3: Diagram showing the setup for calculating the homography.

plugging in the value for H_3^{-1}, we get:

$$v_y cos\theta - \frac{h}{2} cos\theta - \alpha sin\theta = 0 \Rightarrow tan\theta = \frac{v_y - \frac{h}{2}}{\alpha} \tag{8}$$

From the diagram representing the scene geometry (**Figure 3**), we observe that $\varphi = \frac{\pi}{2} - \theta$ and we get:

$$\frac{1}{tan\varphi} = \frac{v_y - \frac{h}{2}}{\alpha} \Rightarrow \varphi = arctan \frac{\alpha}{v_y - \frac{h}{2}} \tag{9}$$

Therefore, we need to rotate the camera by φ around the X-axis to get the overhead view we are after. Meaning that we need to multiply by

$$H_{BEV} = K \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\varphi & sin\varphi \\ 0 & -sin\varphi & cos\varphi \end{bmatrix} K^{-1} \tag{10}$$

in order to send the vanishing point to infinity and obtain the bird's eye view of the original image. Moreover, we crop the resulting images to avoid the parts that are out of focus or the parts that are extremely distorted (see **Figure 4**).

3.5 Step C: Stitching

We apply the previous two steps to a sequence of images and we get multiple overhead view images of a scene plane. Because all the images are of the same 3D plane, there exists a homography, H_{align}, that describes the transformation between them. To be more specific, the fact that the images are

(a) (b) (c)

Figure 4: (Step B) **(a)** Original image. **(b)** Result of applying the homography (Full BEV). **(c)** Cropped BEV.

from an overhead view the relation between them can be described by mere rotation around the Z-axis, translation, and scaling. Therefore, we can write the homography between two overhead views as the following transformation:

$$H_{align} = \begin{bmatrix} s_x & 0 & T_x \\ 0 & s_y & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} H_1 & H_2 & H_3 \\ H_4 & H_5 & H_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Both intuitively and from the matrix multiplication, one can understand that there is no perspective transformation here and simpler affine transformation fully models the images' relation. An affine transformation has only 6 variables thus making the estimation of it a bit easier than perspective. However, in our case, this part of the pipeline is likely to accumulate noises and errors so for better-looking stitched images we did use the projective transformation. As anticipated, our estimated perspective transformation's third row had values that are close to $(0, 0, 1)$. The transformation was estimated with SIFT features matching and RANSAC. This step can be observed in **Figure 5**.

4 Results

For our data, we use the sequence of images shown in **Figure 7**.

In this section, we present our main result along with the results of each of the intermediate steps in our method. First, we present the results of our vanishing point detection algorithm for all the images in our dataset in **Figure 8**. Next, we have **Figure 9** showing cropped BEV images which were the result of applying the homography calculated according to **Section 3.4**. **Figure 10**

(a) (b) (c)

Figure 5: (Step C) **(a)** Cropped BEVs. **(b)** Homography estimation. **(c)** Resulting stitched image.

shows our main result which is the result of stitching all the cropped BEV images from **Figure 9**.

Finally, we can compare our main result to the actual BEV of the scene in **Figure 11**.

5 Discussion

As we can see in the previous section, our vanishing point detector looks very accurate to the human eye 8. Also, our homography estimation step works well as we see that the resulting images are true overhead view images where the parallel lines in the 3D plane are parallel in the BEV images 9. Finally, our main result looks realistic as it is very similar to the actual BEV of the scene 11.

We would like to mention that, initially, we planned to use a few videos we shot in McGill's library as our dataset. Those videos satisfied our assumption about parallel lines in the scene to the human eye. Unfortunately, the images were noisy. We tried using different parameters for Canny edge detector but the parallel lines were not detected (See **Figure 12**).

Now, we will reflect on our assumptions and method. In section 3.3, we made the assumption that the scene is as flat as possible (i.e. close to a 3D plane). This assumption is important because the contours of high structures in the scene might be voted as parallel lines in the scene when using Hough

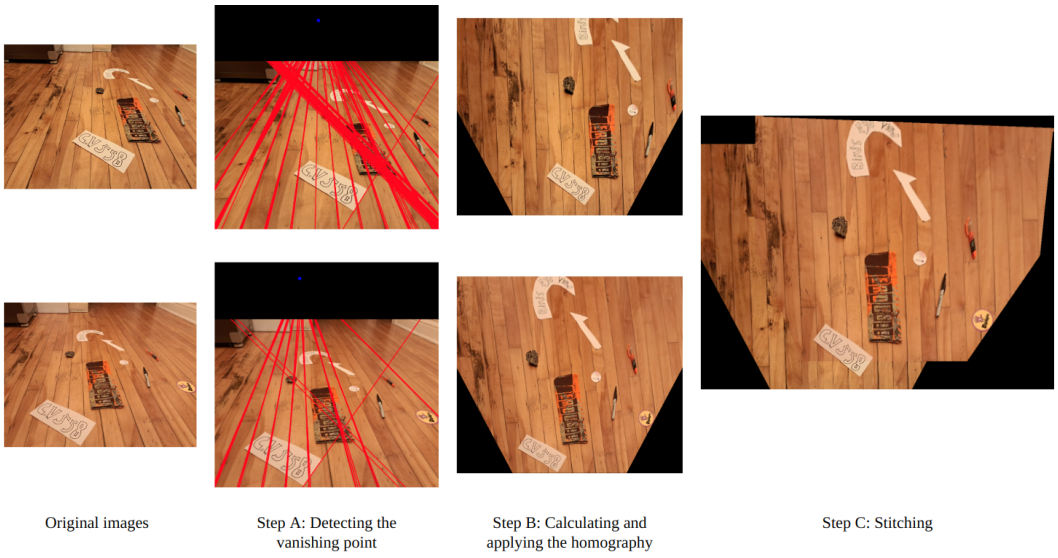


Figure 6: The first column has the original images. The second column (Step A) shows the lines that are considered when calculating the vanishing point (in red) and the location of the vanishing point (in blue). The third column (Step B) shows the BEV of each image. The image on the right (Step C) shows the result of stitching more than on BEV images obtained in Step B.

transform and therefore, would change the location of the vanishing point. For example, applying our vanishing point detection algorithm to an image of Google Street View, where there exists a tall building in the scene, gives an incorrect location of the vanishing point (See **Figure 13**).

Moreover, a part of our method that can be improved is estimating α. For better α estimation, it is possible to use more images with known θ and average over their α estimations or use some other method to reduce the noise of the estimation by ignoring α outliers.

It is important to note that our method is designed to be applied to a sequence of images taken by translating and tilting the camera. In addition, there is a significant advantage of our method which is that it is robust against minor pans since they are corrected in the stitching step 3.5.

One of the disadvantages of our method is that it contains many algorithms that require specifying parameters and thresholds (such as Canny, Hough, RANSAC, SIFT etc.). That makes it less robust to other datasets. Meaning that if one wishes to use our algorithm on a different dataset, it would require a lot of hyperparameter tuning to get decent results.

Future work and extensions. The most appealing possible project

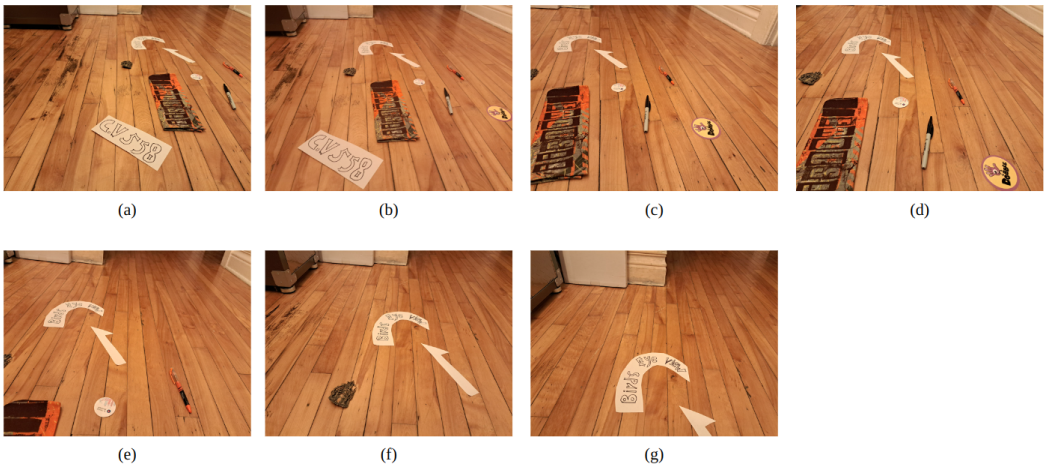


Figure 7: A sequence of images. Each image is taken after translating the camera in the Z-axis with respect to the location of the camera for the previous image in the sequence with some tilts and minor pans.

extension is to use this pipeline to create a real-time application that maps a video to BEV. There are several speedups that might be interesting to check out. One is to use optical flow to track distinct key points so that the stitching will be faster and will not need to calculate and matching SIFT features. Another one is to use the x coordinate of the vanishing point to estimate the camera's pan rotation. This will yield BEV images that are aligned and differ only by translation and scaling (without rotation). Thus, there will be a need for only 4 parameters to estimate for the stitching part (T_x, T_y, s_x and s_y) – fewer matches needed (only 2 matches are enough) and less time to fit a model. More possible and interesting extensions might be trying to find solutions that will help loosen some assumptions, such as the no-roll assumption, to make the algorithm more robust. Another one is to automate the process of finding the thresholds and parameters for some algorithms in the pipeline using some statistics of the images.

6 Splitting of the work

- Developing the analytic/theory of the pipeline - all together
- Implementing the solution - Daniel
- Writing the report - split equally.

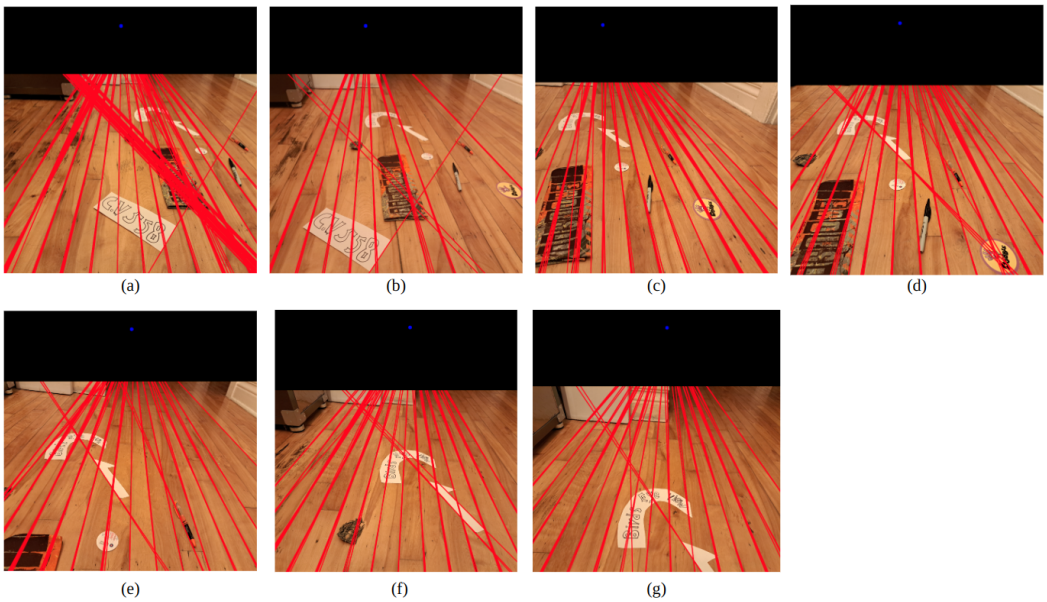


Figure 8: The result of applying our vanishing point detection algorithm to the images in our dataset. The location of the vanishing points is indicated in blue.

- Research and reading - everyone.

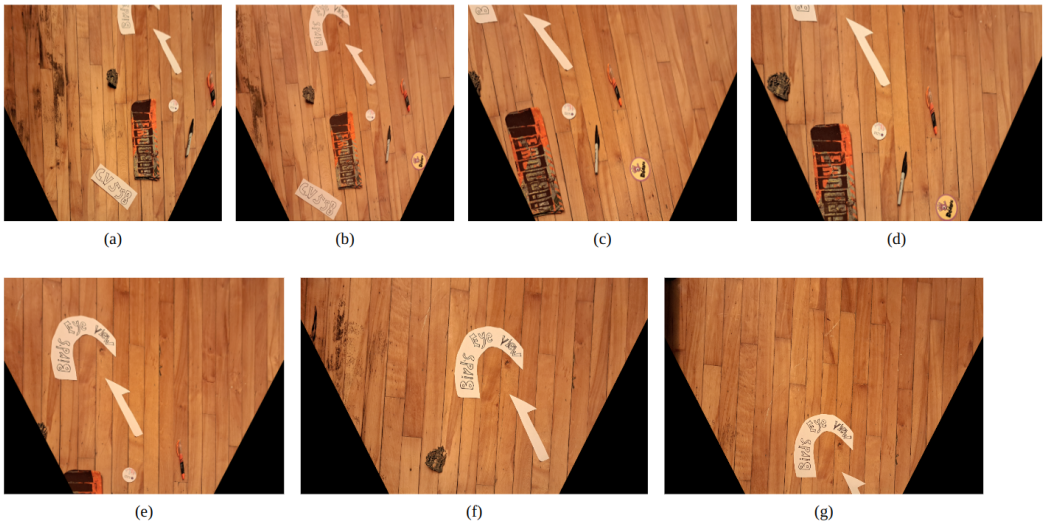


Figure 9: Cropped images of the result of applying the calculated homography to the images in our dataset.

Figure 10: Our main result; stitched BEV.

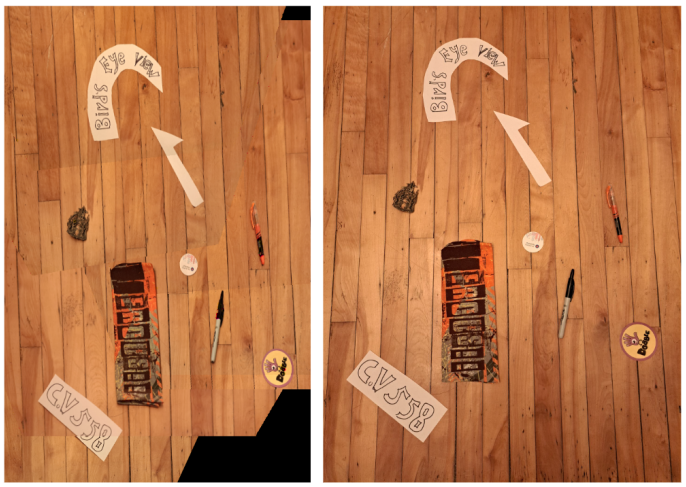


Figure 11: **(a)** BEV result of our approach. **(b)** Actual BEV of the scene.

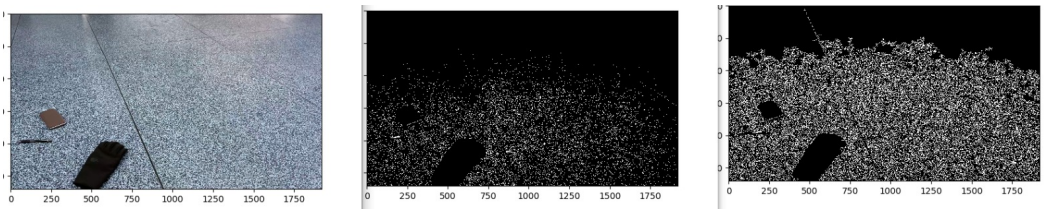


Figure 12: The left image is the original scene. The middle and right images are the result of Canny edge detector using two different settings.

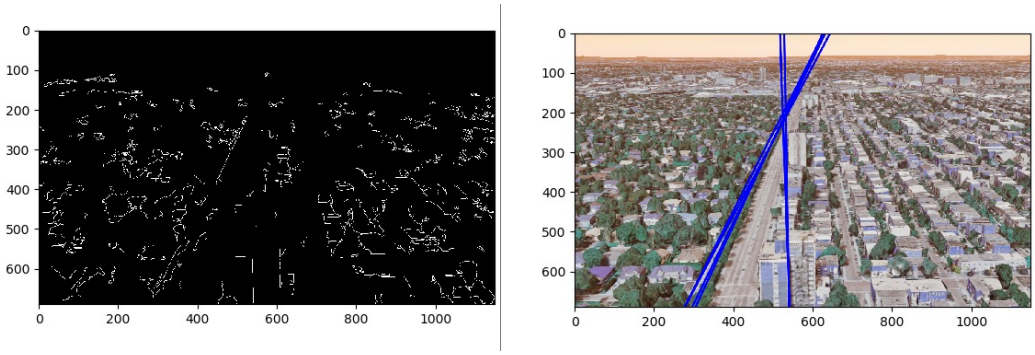


Figure 13: The left image is the result of applying Canny edge detector to a scene taken from Google Street View. The right image shows the result of our vanishing point detection algorithm. We can see that the vanishing point location is incorrect as the tall building's sides in the middle lower part of the image are mistakenly considered parallel lines in the ground plane.

References

[1] Ammar Abbas and Andrew Zisserman. A geometric approach to obtain a bird's eye view from an image. *CoRR*, abs/1905.02231, 2019.

[2] Robert Laganiere. Compositing a bird's eye view mosaic. In *Proc. Conf. Vision Interface*, volume 1, pages 382–387, 2000.

[3] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Qiao Yu, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers, 2022.

[4] Andrea Palazzi, Guido Borghi, Davide Abati, Simone Calderara, and Rita Cucchiara. Learning to map vehicles into bird's eye view. pages 233–243, 10 2017.