3D Vehicle Reconstruction via Monocular Camera with Deep Learning Models and Direct Linear Transformation

2024-2 Robot Vision (M3228.003000)

Thomas Putzer
Computer Science and Engineering
Seoul National University
Seoul, South Korea
putzerthomas55@gmail.com

Weihao Chao
Mechanical Engineering
Seoul National University
Seoul, South Korea
theweihao@gmail.com

Anggraini Ditha
Civil and Environmental Engineering
Seoul National University
Seoul, South Korea
dithanggraini1598@gmail.com

***Abstract*—Hello here goes the abstract**

I. Project Objectives

This project aims to develop a workflow for reconstructing wireframe 3D vehicle models from monocular camera images. The approach involves detecting keypoints of vehicles using a deep learning framework, OpenPifPaf, that is designed for detecting human poses (key points) and other object structures within images. Using the CarFusion dataset (Reddy et al., 2018), we trained the deep learning model to recognize key vehicle features and use those to estimate a basic 3D model of vehicles and to estimate vehicle speed.

Previous research in 3D vehicle reconstruction often relies heavily on deep learning models throughout the pipeline, utilizing techniques such as Graph Neural Networks (GNNs) in combination with Convolutional Neural Networks (CNNs) or transformers. While these methods can achieve high accuracy, they are computationally intensive and require substantial processing power, making them less practical for real-time applications or deployment on resource-constrained devices. Our project takes a different approach by limiting the use of deep learning to the initial stage of keypoint detection, where it excels at identifying critical features such as vehicle parts and keypoints from 2D images. Once these keypoints are identified, the methodology transitions to geometry-based point tracking and traditional computer vision techniques to perform 3D reconstruction and pose estimation. This strategic design significantly improves computational efficiency, as geometry-based methods are less resource-intensive than deep learning while still being capable of leveraging the spatial relationships inherent in 3D space. Our project also proposes the application of the RANSAC (Random Sample Consensus) algorithm for robust handling of errors in detected keypoints. Finally, while previous research focuses on static 3D pose reconstruction, our project extends its scope to estimate vehicle speed dynamically over time by utilizing primitive frame-by-frame car tracking 1.

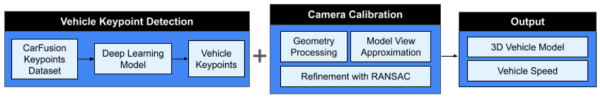

Fig. 1: Project workflow.

II. Vehicle Key Point Detection

In this project, we adapt the OpenPifPaf pose-detection deep learning model for car keypoint detection. Using Carfusion datasets for testing and training data. For initial training, we utilized a pre-trained model with a ResNet-101 backbone. We trained the model on a subset of 10 images over 150 epochs. For this model, we successfully generated the weight file as well as the keypoint data of the predicted image.

(a) label 1 (b) label 2

Fig. 2: Examples of the predicted key points from the OpenPifPaf model. Detected keypoints are shown in yellow and the computed bounding box in red. The overall confidence score is shown above the bounding box.

III. 3D Pose Estimation

A. Geometry Processing

To understand the 3d pose estimation process a basic understanding of the geometry processing stage of the 3d rendering pipeline is required. That is the process of how a 3d model and its vertices are transformed from points in 3d space to pixels on a 2d screen.

The vertices of a 3D model are defined in their local coordinate system, the so-called local space. If the model is placed inside the 3D scene, its position in world space can be controlled with the model matrix M_{mod}. World space is the coordinate system of the *world*, the 3d scene where all the objects, lights, and the camera live. The camera's position and rotation in world space are determined by the inverse of the view matrix M_{view}^{-1}. For the next steps of the 3D rendering pipeline, it is generally preferred to have objects in view space rather than world space. View space is the reference frame of the camera, meaning the camera is located at its origin looking straight ahead. To transform an object from world- into viewspace, it just needs to be multiplied by the view matrix, meaning to get from local- to view space the composition of the model and view matrices can be used: this composition is called model view Matrix $M_{MV} = M_{view}M_{mod}$.

The next step is figuring out where on the screen a vertex should be drawn. Once in view space, the projected position on the $z = 1$ plane can be computed by dividing the position of a vertex by its z-component. More formally you can multiply by the projection matrix and divide by the w-component. This matrix representation of the standard projection onto the $z = 1$ plane looks like this:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \overset{\text{div z}}{\equiv} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ \frac{z}{z} \\ 1 \end{bmatrix} \tag{1}$$

IV. Introduction

[1] (Leave this citation for now, otherwise it breaks (if we have 0 citations, and the command at the end))

V. 3D Pose Estimation

($A = U\Sigma V^T$) [2] In the SVD $Ab = U\Sigma V^T b = 0$ implies $V^T b = 0$ since U is invertible and has no nontrivial solutions for $Ub = 0$. Thus $V^T b$ must be a vector where only the rows corresponding to zero singular values contribute to the null space. The columns of V that correspond to zero singular values in Σ combined form a basis for the null space of A.

$$\left[\begin{array}{ccc:c} & R & & t \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right]$$

$\leq$

```
#Pseudo code of the matching process
#cars keeps track of all the cars in the scene
#cars_n contains all the cars in the new frame
for old_car in cars:
    c, d = closest_car(cars_n, old_car)
    if d > max_distance:
        #the car left the frame
        cars.remove(old_car)
    else:
        old.match(c)
        #every car can only be matched once
        cars_n.remove(new_car)
#all remaining not matched new cars
for new_car in cars_n:
    #a new car has entered the frame
    cars.append(new_car)
```

double column text here. . double column text here. . .double column text here. . .lkasj fundingasdf . double column text here. . lsakdfj laks ich icha heise . double column text here. . laskdjf lasdffi cich ic ach .

lkasdjf lakjsd flkajsdl $x_p = (x'/2 + 0.5) \cdot width$ and $y_p = (y'/2 + 0.5) \cdot height$ fkjasdl fjasldj flaskdjf laskdjf laksjdfl kajsdflkajsdlfjasldkfjalsdkjflaskjdf laskdjf lasjkdfl aksjdflkasjd flaskjdf lkasdjf lkasjdf laksjdfl akjsdlfj asldkfj alsdjkf lasdjf laskdjf laskdjf laksdjfl aksdjfl askdjl askjf l

$$FOV = 2 \cdot \arctan\left(\frac{s_w}{2f}\right), r = n \cdot \tan\left(\frac{FOV}{2}\right) \tag{3}$$

double column text here. .double column text here. . . double column text here. . . double column text here. . . double column text here. . .lkasj fundingasdf double column text here. .double column text here. . . double column text here. . . double column text here. . . double column text here. . .lkasj fundingasdf double column text here. .double column text here. . . double column text here. . . double column text here. . . double column text here. . .lkasj fundingasdf

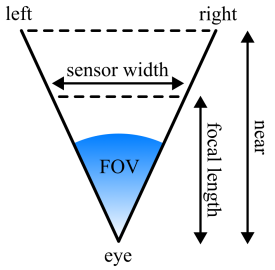

Fig. 6: The sensor width and the focal length are used to calculate the camera's field of view.

double column text here. .double column text here. . . double column text here. . . double column text here. . . double column text here. . .lkasj fundingasdf double column text here. .double column text here. . . double column text here. . . double column text here. . . double column text here. . .lkasj fundingasdf

Matrix representation of the standard projection onto the z = 1 plane.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \overset{\text{div z}}{\equiv} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ \frac{z}{z} \\ 1 \end{bmatrix} \tag{4}$$

The API projection Matrix

To get the screen-space position of a vertex, we multiply it by its Modelview Matrix and API Projection Matrix before doing the projective division. We have all the parameters except the Model Matrix. $p' = M_{API}M_{Modelview}p$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} \overset{\text{div w}}{\equiv} \begin{bmatrix} c_1 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 \\ 0 & 0 & c_3 & c_4 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$M_P = M_{P''} M_{P'} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{-(r+l)}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{-(t+b)}{t-b} \\ 0 & 0 & \frac{2}{f-n} & \frac{-(f+n)}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} c_1 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 \\ 0 & 0 & c_3 & c_4 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

Fig. 3: The values r, l, t and b can be calculated from the focal length and sensor size of the camera (see 6). E.g., r can be calculated in the following way: $r = n \cdot \tan\left(\frac{FOV}{2}\right), FOV = 2 \cdot \arctan\left(\frac{s_w}{2f}\right)$, ($s_w$ = sensor width, f = focal length, FOV = field of view).

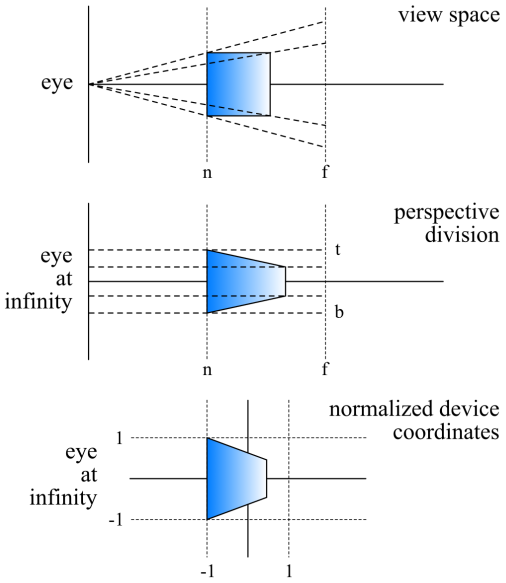

Fig. 4: Perspective projection from view space to NDC in 2d.

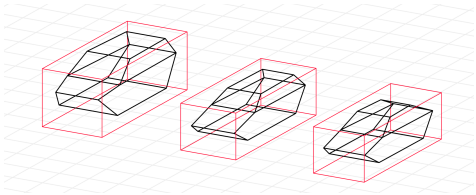

Fig. 5: Different car wireframes used. From left to right Ford Explorer, Volvo V60, Nissan Altima.

If we multiply that out, we get for x' and y':

$$x' = \frac{c_1 ax + c_1 by + c_1 cz + c_1 d}{ix + jy + kz + l}$$
$$y' = \frac{c_2 ex + c_2 fy + c_2 gz + c_2 h}{ix + jy + kz + l}$$

We can multiply by the denuminator and solve for 0. Since this holds for every pair of points $(p_1, p'_1), (p_2, p'_2), ..., (p_n, p'_n)$ that we track we can create the following system of equation:

REFERENCES

[1] J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, T. G. B. Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden, "Quantitative analysis of culture using millions of digitized books," *Science*, vol. 331, no. 6014, pp. 176–182, 2011. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1199644

[2] "Singular value decomposition," https://en.wikipedia.org/wiki/Singular_value_decomposition, accessed: 2024-12-06.

$$\begin{bmatrix} -c_1x_1 & -c_1y_1 & -c_1z_1 & -c_1 & 0 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & z_1x'_1 & x'_1 \\ 0 & 0 & 0 & 0 & -c_2x_1 & -c_2y_1 & -c_2z_1 & -c_2 & x_1y'_1 & y_1y'_1 & z_1y'_1 & y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -c_1x_n & -c_1y_n & -c_1z_n & -c_1 & 0 & 0 & 0 & 0 & x_nx'_n & y_nx'_n & z_nx'_n & x'_n \\ 0 & 0 & 0 & 0 & -c_2x_n & -c_2y_n & -c_2z_n & -c_2 & x_ny'_n & y_ny'_n & z_ny'_n & y'_n \end{bmatrix} \begin{bmatrix} a \\ b \\ \vdots \\ k \\ l \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{5}$$