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*—You can set more than one value in the parameter, for instance, if you write [ht] LaTeX will try to position the table here, but if it's not possible (the space may be insufficient) then the table will appear at the top of the page. It is recommended to use more than one positioning parameter to prevent unexpected results.**

Notice also the command. This changes the alignment of the table within its container to centre instead of the default left.

en you can use the environment wraptable which takes two parameters: The first one is the alignment that can be l, r, c, i or o for left, right, centre, inner and outer respectively. The second one is the width of the table container, keep in mind that this latter parameter must be the same as the width of the table, otherwise things may not be properly aligned.

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.

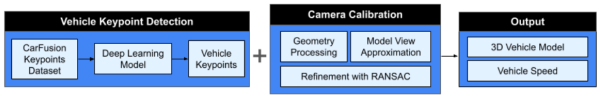

Fig. 1: Project workflow.

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.

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

(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.

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:

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

The projection still needs to be modified to have more control over what exactly is seen on the screen. Instead of projecting on the $z = 1$ plane, objects are projected onto the $z = n$ plane, the near plane. To preserve depth information objects are not completely projected onto the near plane, but instead to the space between the near- and far planes. This is done by multiplying with $M_{P'}$ before dividing by the w component. This has no implications for this project, since there exists no depth information in the detected keypoints. Additionally since it is easier to work in relative coordinates when faced with pictures of different sizes, the projected view coordinates are finally transformed into normalized device coordinates (NDC) by multiplying with $M_{P''}$ (see 2). Normalized device coordinates are a coordinate system around the origin with the bounds of $[-1, 1]$ for every axis. The relative pixel coordinates of a vertex can simply be read off the x and y positions of the points in NDC. To get the exact pixel positions these relations between NDC- and pixel coordinates can be used: $x_p = (x'/2 + 0.5) \cdot width$ and $y_p = (y'/2 + 0.5) \cdot height$.

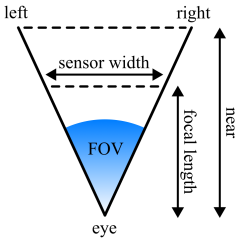

Fig. 4: The sensor width and the focal length are used to calculate the camera's field of view.

This process basically transforms the view frustum into a cube of side length two around the origin (see 5).

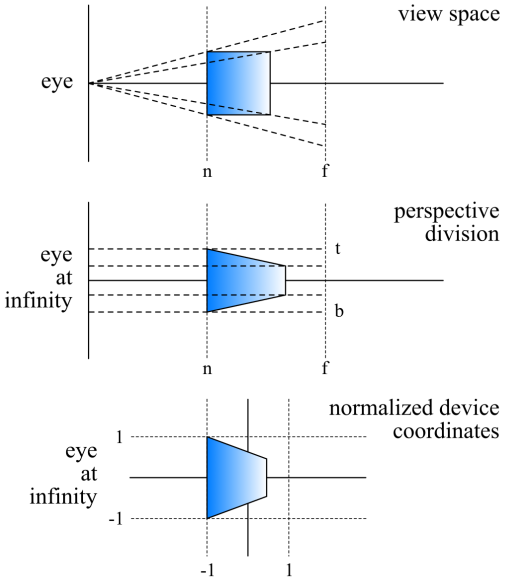

Fig. 5: Perspective projection from view space to NDC in 2d.

B. Modelview Matrix Approximation

To get the approximate 3d position of the cars, the detected 2d key points need to be correlated with their 3d counterparts. As discussed earlier the screen space positions can be computed by multiplying the vertices in local space with the modelview matrix M_{MV} and the projection matrix M_P before dividing by w.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \overset{\text{div z}}{=} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ \frac{z}{z} \\ 1 \end{bmatrix} \tag{3}$$

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

Fig. 3: The values r, l, t and b can be calculated from the focal length and sensor size of the camera (see 4). 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).

This matrix multiplication and the final division by the homogeneous component can be performed and solved for x' and y' that gives:

$$x' = \frac{c_1 ax + c_1 by + c_1 cz + c_1 d}{ix + jy + kz + l} \tag{4}$$

$$y' = \frac{c_2 ex + c_2 fy + c_2 gz + c_2 h}{ix + jy + kz + l} \tag{5}$$

Now one can multiply by the denominator and set both equations to 0. With the assumption that the focal length and sensor size of the camera are known and we have a matching from the key points to the vertices of a 3d model, the only unknown are the values of M_{MV}, meaning $c_1..c_4$. Since this relation hold for every pair of points $((p_1, p_1'), (p_2, p_2'), ..., (p_n, p_n'))$, the following system of equation can be created (see 6).

The solution to this homogeneous system of equations ($Ab = 0$) can be obtained by finding the null space of the matrix A. The null space is the set of all vectors b that satisfy $Ab = 0$. This set can be found with the help of a singular value decomposition (SVD) that factorizes a matrix A into a rotation, followed by a rescaling followed by another rotation ($A = U\Sigma V^T$) [1]. In the SVD the formula $Ab = U\Sigma V^T b = 0$ 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. In this way one solution for the system of equations can be found. Out of this solution the following matrix can be created:

$$\left[\begin{array}{ccc:c} & R & & t \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{7}$$

Here R stands for a 3×3 matrix that holds the rotation and scaling part of the solution and t for a vector of size 3 containing the translation. Together they do not uniquely define the modelview matrix, since this result is scale invariant. At this point the model cannot distinguish between a big object far away and a small object close by since their projections on the screen would look the same to the camera. But the fact that the scale is 1 solves this dilemma. This means that R is orthogonal, meaning it has only rotational components, but no scaling. The correct R can be calculated with another SVD, by just omitting Σ, which avoids any scaling; a new R can be calculated this way: $R' = UV^T$. The last step is scaling t by the same amount and replacing R by R' in the result; the modelview matrix is found.

For this process to work it is very important that the 3d model is as close as possible to the real geometry of the object where the key points came from. To minimize differences our model uses three different car wireframes 6 and picks the one with the smallest fitting error (the error is the sum of distances of the key points and projected 3d vertices in NDC). The different cars were picked to cover a wide range of possible car sizes, from midsized, over station wagons up to SUVs.

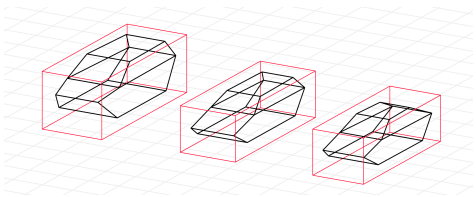

Fig. 6: Different car wireframes used. From left to right Ford Explorer, Volvo V60, Nissan Altima.

C. Estimation Refinement

Since the detected key points can be incorrect it is of utmost importance that the model is robust against outliers and small errors. This problem can be improved by utilizing random sample consensus (RANSAC), an algorithm that estimates the right parameters for a mathematical model from a set of measured/observed data that can contain outliers or values with large errors [2]. In this case the basic RANSAC algorithm has been modified slightly to check all subsets of a specific size, since the total number of samples is ≤ 14, meaning the total number of subsets is a manageable size. This transforms this version of RASAC into a deterministic algorithm that always produces the same output.

IV. Speed Estimation

A. Car Matching Over Multiple Frames

To calculate the speed, it is not enough to know the relative location of the car at a specific point in time. It is also necessary to know the location of the same car at a different point in time. Since the Keypoint Detection step only detects the car's key points and bounding box, it is not trivial to know

$$\begin{bmatrix} -c_1x_1 & -c_1y_1 & -c_1z_1 & -c_1 & 0 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & z_1x_1' & x_1' \\ 0 & 0 & 0 & 0 & -c_2x_1 & -c_2y_1 & -c_2z_1 & -c_2 & x_1y_1' & y_1y_1' & z_1y_1' & y_1' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -c_1x_n & -c_1y_n & -c_1z_n & -c_1 & 0 & 0 & 0 & 0 & x_nx_n' & y_nx_n' & z_nx_n' & x_n' \\ 0 & 0 & 0 & 0 & -c_2x_n & -c_2y_n & -c_2z_n & -c_2 & x_ny_n' & y_ny_n' & z_ny_n' & y_n' \end{bmatrix} \begin{bmatrix} a \\ b \\ \vdots \\ k \\ l \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

which car is which in the next frame. But since videos are usually shot at 30fps or more, a car's bounding box does not move very far from one frame to another. Considering this and the fact that generally there are not a lot of cars whose bounding boxes overlap, the simple approach of matching the two cars whose bounding boxes are closest to each other works just fine. To allow new cars to enter the frame and old cars to leave the frame the variable $max\_distance$ is used, which describes the maximum distance a car can move from one frame to the next.

```
#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)
```

Once the 3d position of a car over multiple frames is known, the car's speed can be calculated by dividing the distance traveled by the elapsed time.

V. Materials

The original source code for the keypoint detection can be found here [3] and the source code for the 3d pose reconstruction and 3d visualization is available on this github repository [4].

References

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

[2] "Random sample consensus," https://en.wikipedia.org/wiki/Random_sample_consensus, accessed: 2024-12-06.

[3] M. Bonnesoeur, "Semester project : Keypoint-based vehicle 3d localization," https://github.com/LouisNUST/keypoint-based-car-detector/tree/master, 2020, accessed: 2024-12-08.

[4] T. Putzer, "keypoint2pose," https://github.com/PuuTzzA/keypoint2pose, 2024, source code for the 3d pose approximation step.