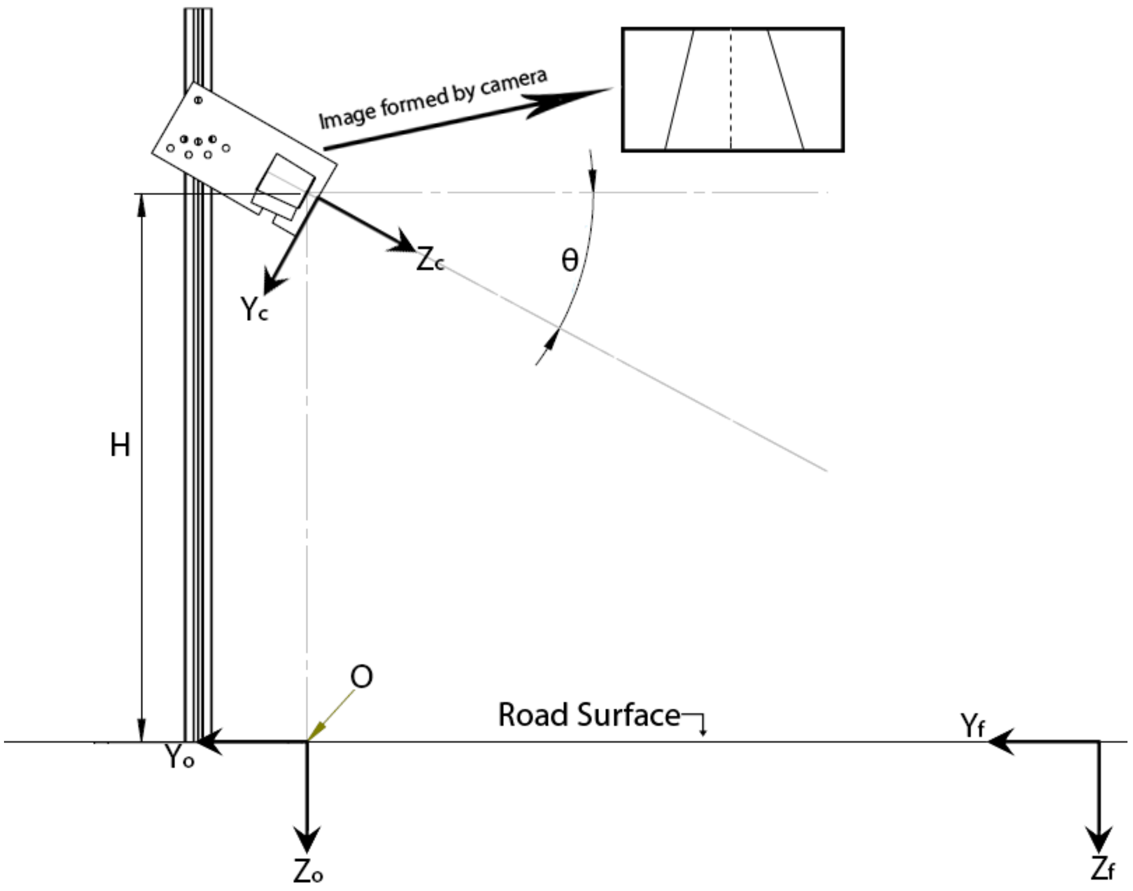
# Inverse Perspective Transformation

## Setting up the problem

Given a (naturally) perspective-transformed image from a camera located at a certain known height and orientation w.r.t. ground, an inverse perspective transform is to be applied in order to get an image showing how the ground surface would look from the top view.

**Inputs (Real-time):** Naturally perspective-transformed image, camera height and orientation w.r.t. ground (from **I**nertial **M**easurement **U**nit).
**Known Information:** Camera intrinsics ($f_x, f_y, c_x$ & $c_y$).
**Parameters to be chosen:** Field of view (origin, width, height), scaling factor for output image.
**Output (Real-time):** A top-view image of the selected field of view at the given scaling factor ($s$).



## Frame definitions

**Convention:**

- Uppercase $X, Y, Z$ denote real world coordinates (measured in meters).
- Lowercase $x, y$ denote image coordinates (measured in pixels). In the image, $x$ points to the right and $y$ points downwards.

**Camera frame:** Optical frame attached to the camera recording the input image.

- $X_c$ points towards the real-world direction corresponding to 'right' in the camera image,
- $Y_c$ points towards the real-world direction corresponding to 'down' in the camera image, &
- $Z_c$ points towards the real-world direction corresponding to 'inwards' in the camera image.

Before defining the FOV frame, we define an **intermediate frame** for convenience, whose origin is at the projection of the optical centre of the camera on the ground, denoted by $O$.

- $X_o$ is parallel to $X_c$,
- $Z_o$ points into the ground, and
- $Y_o$ is defined such that $X_o, Y_o$ & $Z_o$ form a right-handed coordinate system like the others.

**FOV frame:** Frame attached to the desired field of view on the ground. This frame is defined to be parallel to the intermediate frame with its origin such that the point $O$ is at $(O_X, O_Y, 0)$ in this frame. The desired field of view is a rectangle of width $W$ (along $X_g$) and height $H$ (along $Y_g$), whose top-left corner (corresponding to the pixel $(0, 0)$ in the top-view image) is at the origin of this frame. The remaining pixels are related by $(x_f, y_f) = (sX_f, sY_f)$ where $s$ is the desired scaling factor in pixels/m.

- $X_f$ points towards the real-world direction corresponding to 'right' in the top-view image,
- $Y_f$ points towards the real-world direction corresponding to 'down' in the top-view image, &
- $Z_f$ points towards the real-world direction corresponding to 'inwards' in the top-view image.

This is not a world-fixed frame since the field of view moves as the vehicle moves.

**World frame:** A world-fixed frame to which features in the FOV frame can be easily transformed to after the top-view transform. The transform from the world frame to the FOV frame depends on the location and heading of the vehicle.

## Transformation algorithm

In order to form the top-view image, we use the following algorithm:

1. Create an empty top-view image of size $(sW, sH)$
2. For each pixel $(x_f, y_f)$ in the top-view image, locate the corresponding pixel $(x_c, y_c)$ in the camera image.
3. Copy the RGB values of the pixel nearest to $(x_c, y_c)$ (since the calculated value may not turn out to be an integer), and set the pixel $(x_f, y_f)$ in the top-view image to those RGB values.
4. Repeat the procedure for each pixel in the top-view image.

On termination of the algorithm, we will get an image that looks what the specified field of view on the ground would look like, as seen from the top (scaled by $s$ pixels per meter).

All we need to do now is to get $(x_c, y_c)$ corresponding to a given $(x_f, y_f)$.

## Deriving the transformation equation

Given:

$$Top - view\ pixel\ coordinates \equiv (x_f, y_f)$$

We now perform all the necessary transformations one-by-one using elementary transformations such as scaling, translation and rotation. For simplicity, we assume camera roll to be zero.

$$(X_f, Y_f, Z_f) \equiv \left(\frac{x_f}{s}, \frac{y_f}{s}, 0\right)$$

$$(X_o, Y_o, Z_o) \equiv (X_f - O_X, Y_f - O_Y, Z_f)$$

$$(X_c, Y_c, Z_c) \equiv (X_o, (H + Z_o)\cos(\theta) + Y_o \sin(\theta), (H + Z_o)\sin(\theta) - Y_o\cos(\theta))$$

Also, using the pinhole-camera model, we get $(x_c, y_c)$ (the use of this model is justified because the input image is corrected for spherical distortion):

$$(x_c, y_c) \equiv \left(\frac{X_c}{Z_c} f_x + c_x, \frac{Y_c}{Z_c} f_y + c_y\right)$$

Substituting all these equations into the next one-by-one, we get:

$$(x_c, y_c) \equiv \left(\frac{\left(\frac{x_f}{s} - O_X\right)}{H\sin(\theta) - \left(\frac{y_f}{s} - O_Y\right)\cos(\theta)} f_x + c_x, \frac{H\cos(\theta) + \left(\frac{y_f}{s} - O_Y\right)\sin(\theta)}{H\sin(\theta) - \left(\frac{y_f}{s} - O_Y\right)\cos(\theta)} f_y + c_y\right)$$

## OpenCV Implementation

While using the above algorithm is sufficient, OpenCV provides a better implementation which includes added features such as linear interpolation (rather than nearest-neighbour interpolation) and which performs faster, in general. The function warpPerspective()[1] can perform general transformations of the form:

$$\mathtt{dst}(x, y) = \mathtt{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

Where $dst(x, y)$ represents destination image coordinates, $src(x, y)$ represents source image coordinates and $M$ is a $3 \times 3$ matrix given by the user. Note that we have found the source coordinates for a given destination coordinate, which is exactly what this transform represents (the names $dst$ and $src$ might seem incorrectly swapped if you compare our transformation equation as-it-is with this equation, but a logical comparison will show that it is not the case). By performing some algebraic manipulation, we can get the values of $M$ by comparing our transformation equation with the above equation:

$M_{11} = f_x$

$M_{12} = -c_x \cos(\theta)$

$M_{13} = s(c_x(\mathrm{H}\sin(\theta) + O_Y \cos(\theta)) - f_x O_X)$

$M_{21} = 0$

$M_{22} = f_y \sin(\theta) - c_y \cos(\theta)$

$M_{23} = s\left(c_y(H\sin(\theta) + O_Y\cos(\theta)) + f_y(H\cos(\theta) - O_Y\sin(\theta))\right)$

$M_{31} = 0$

$M_{32} = -\cos(\theta)$

$M_{33} = s(H\sin(\theta) + O_Y\cos(\theta))$

[1] This function is documented here:
https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#warpperspective