## Field of View Corrections

Whenever we capture an image with a camera sensor, the result doesn't fully match the real world. This is caused due to many issues, most of which are closely related and can be included in the term "field of view" (abbreviated FoV). These problems become very noticable because the camera is tilted 45° in regards to the ground level, as can be seen on figure 1.

The issues are a result of representing a 3D world on a 2D picture. The issues include:
- Size and distance appear smaller the further away these are from the center of the picture.
- The higher we look on the picture, the wider the arch of vision used for the picture was.
- Perspective causes two lane markings to converge as they get closer to the horizon, even though they are always the same distance apart.

In order to use a camera sensor to capture the coordinates of the lane markings accurately, we need to be able to correct for the field of view influences. On figure 1 we have used the known data (marked as red) to calculate values (using the trigonometric functions), among those the minimum and maximum distance that the camera can view. Note that we are calling the distance from the camera as seen on figure 1 the x-axis. As seen on the captured picture, this means that the axis going from the bottom to the top is called the x-axis as well, even though it would usually be the y-axis.

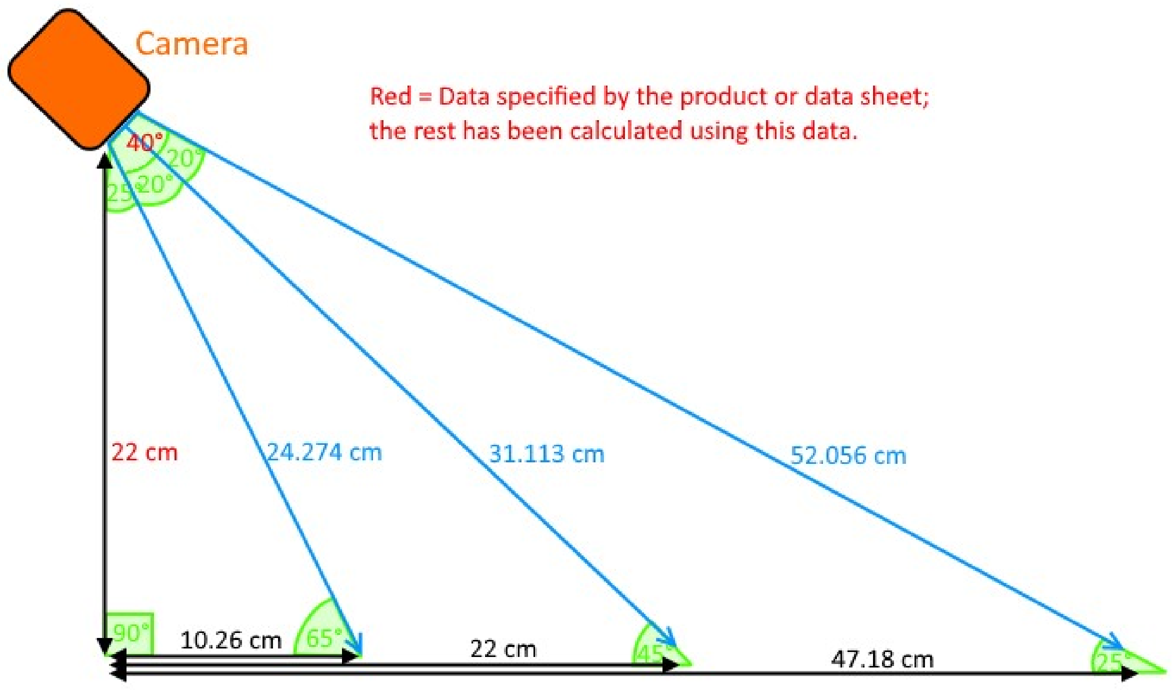

Camera
Red = Data specified by the product or data sheet;
the rest has been calculated using this data.
40°
20°
20°
25°
22 cm
24.274 cm
31.113 cm
52.056 cm
90°
10.26 cm
65°
22 cm
45°
47.18 cm
25°

*Figure 1: A YZ axis illustration of the camera sensor FoV with annotated measurements and calculated results. The x-axis is facing perpendicular to the image. The blue colored arrows border the area that the sensor captures.*

The intent is to map each point in the picture taken by the camera to a corresponding coordinate set on the floor. From the data sheet of the nxtCamV4 we know that it has a field of view of 40° (not actually true, because the 40° are an average of the data sheets and of view of 40° (not actually true, because the 40° are an average of the data sheets and another sensor test. Fix this). In our own sensor tests we concluded that the field of view of our camera is [NOT DONE].

To do this, we first convert every position on the picture into into degrees of rotation, out of the possible 40 degrees of vision that the camera offers. As such, if a point placed at the top edge of the picture (thereby it has the highest possible x-value), it will convert to 40 degrees of rotation along the x-axis as shown on the figure. With this we conclude that the corresponding point on the ground is placed at an x-value of 47.18 cm. Now to calculate this result dynamically for any x-value on the picture.

$$FoV := 40\ deg$$
$$FoV_X := FoV$$
$$FoV_Y := FoV$$
$$cam_{height} := 22\ cm$$
$$cam_{rotationX} := 45\ deg$$
$$cam_{blindSpotDeg} := cam_{rotationX} - \frac{FoV}{2} = 25\ deg$$

**X-Coordinate**

We convert our x-value (from 0 to 100) to a corresponding degree in the field of vision:

$$f_{getDegX}(x) := \frac{x}{\left(\frac{100}{FoV_X}\right)}$$

We use a trigonometric formula to calculate the distance from the origin (0,0) to our corresponding point on the ground.

$$\tan(A) = \frac{opposite}{adjacent} \xrightarrow{solve, opposite} adjacent \cdot \tan(A)$$

$$f_{groundX}(x) := cam_{height} \cdot \tan\left(f_{getDegX}(x) + cam_{blindSpotDeg}\right)$$

**Y-Coordinate**

Imagine an object placed on the furthest left point of the picture (meaning a y-value of 100 of 100). Because the camera is tilted, if the object had a high x-value, it's real position upon the y-axis would be even further left than if it had a low x-value. This means that the y-value of an object is dependant on both the x and y from the picture, and not only the y-value. The way this manifests in our calculations can be seen on figure 2.

25
Z-Axis
Camera
Angle Along Y Axis
20

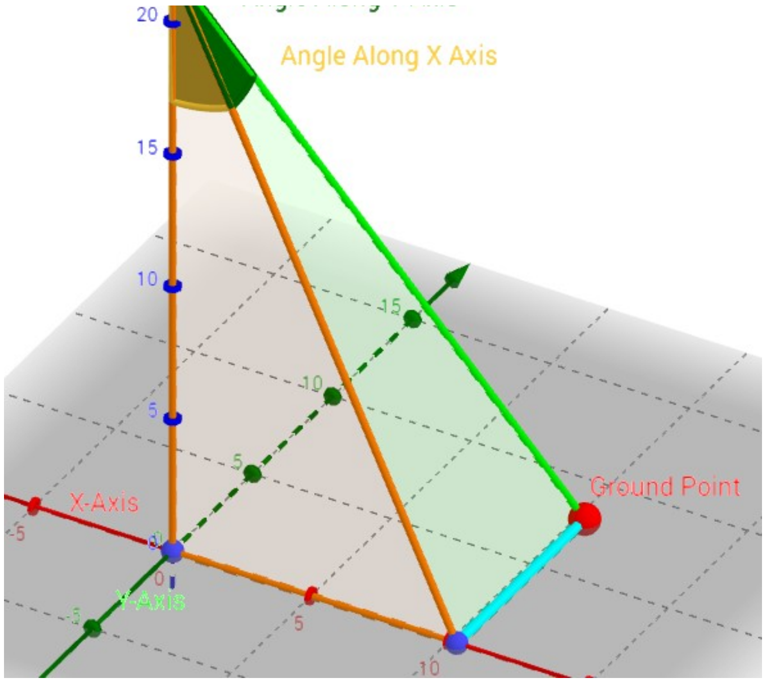


*Figure 2: A 3D plot showing the calculations needed. The point with highest z-value is the camera, and the red marked point is the point on the ground of which we intend to find the coordinates. As such, the value we are interested in is the length of the light-blue line.*

To get this length, we require the hypothenuse of the orange triangle, which represents the ground point's distance from the x-axis.

As previously, we start by converting position on the y-axis of the picture to a corresponding degree. In this case, we imagine that the center of the picture (20° to either side) is placed in a y-value of 0.

$$f_{getDegY}(y) := \frac{y}{\left(\frac{100}{FoV_Y}\right)} - 20\ deg$$

Using pythagorean theorem we calculate the hypotenuse of the orange triangle.

$$f_{hypoX}(x) := \sqrt{\left(cam_{height}\right)^2 + f_{groundX}(x)^2}$$

The hypotenuse length is now used as one of the sides for the triangle to the ground point on the y-axis. As previously, the tangent trigonometric formula is used to calculate the distance along y-axis to the corresponding point.

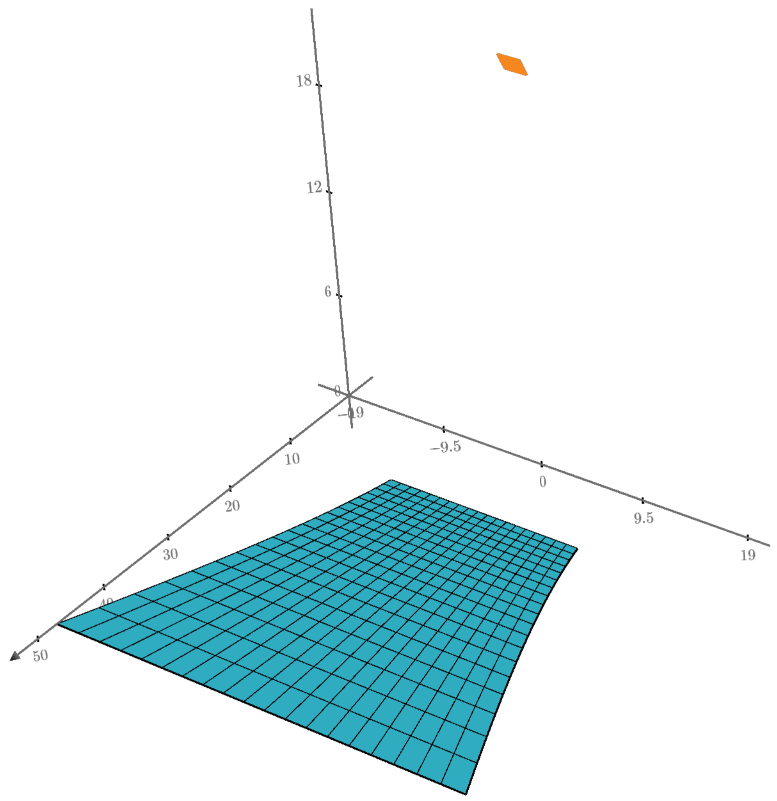
$$f_{groundY}(x, y) := f_{hypoX}(x) \cdot \tan\left(f_{getDegY}(y)\right)$$

## Picture coordinates to ground coordinates

We now describe a matrix function that calculates x,y and z values using the coordinates x and y from the picture. See figure 3 for how the results look in a 3D plot.

$$f_{ground}(x, y) := \begin{bmatrix} f_{groundX}(x) \\ f_{groundY}(x, y) \\ 0\ cm \end{bmatrix}$$

---



$$\text{CreateMesh}(f, 0, 1, -1, 1)$$

$$\text{CreateMesh}\left(f_{groundWithoutUnits}, 0, 100, 0, 100\right)$$

*Figure 3: A 3D plot showing the values of x and y (minimum 0, maximum 100) from the picture as converted to their corresponding ground coordinates. The orange square at the top represents the camera and its angle.*

Because the values used are from 0 to 100, the entire blue area are the coordinates of ground space that the camera will capture with one picture. Now, the function $f_{ground}(x, y)$ fully maps x and y coordinates on the picture to their corresponding x, y and z coordinates on the ground (in centimeters). This algorithm will now be implemented into the nxtCamV4 controller class.