6.819 Advances in Computer Vision_Spring2022

Ibrahim Ibrahim

914997060

iibrahim@mit.edu / ibrahimibrahim@gsd.harvard.edu

Pset2

Problem 1a

1: theta_1 = c/a

2: n * theta_2 = theta_1 + theta_S

3: theta_S = theta_2 + theta_3

4: n * theta_3 = theta_4

5: theta_4 = c/b

Derivation:

$$\boxed{\theta_1 = \frac{c}{a}}$$

$$n = \frac{\theta_1 + \theta_S}{\theta_2}$$

$$\Rightarrow \boxed{n\theta_2 = \theta_1 + \theta_3}$$

$$\Rightarrow \boxed{n\theta_3 = \theta_4}$$

$$\theta_S + 0 = \theta_2 + \theta_3$$

$$\Rightarrow \theta_S = \theta_2 + \theta_3$$

$$\Rightarrow \boxed{\theta_4 = \frac{c}{b}}$$

Problem 1b

f = R / n - 1

Derivation:

$$\theta_2 + \theta_3 = \theta_5$$

$$\frac{\theta_1 + \theta_5}{n} + \theta_3 = \theta_5$$

$$\frac{\theta_1 + \theta_5}{n} + \frac{\theta_4}{n} = \theta_5$$

$$\theta_1 + \theta_4 = \theta_5 (n-1)$$

$$\frac{c}{a} + \frac{c}{b} = \theta_5 (n-1)$$

$$\cancel{c}\left(\frac{1}{a} + \frac{1}{b}\right) = \frac{\cancel{c}}{R}(n-1)$$

$$\frac{1}{f} = \frac{n-1}{R}$$

$$\boxed{f = \frac{R}{n-1}}$$

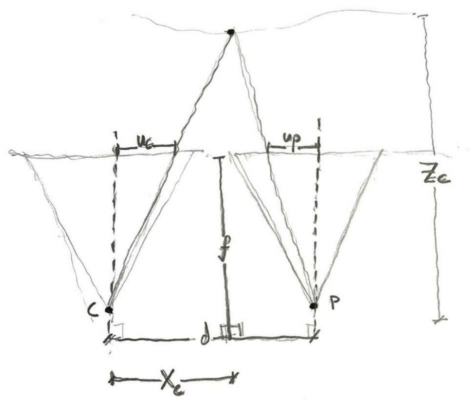
Problem 2a

Z_c = d * f / u_c - u_p

X_c = d * u_c / u_c - u_p

Y_c = d * v_c / u_c - u_p

Derivation:



$$\frac{d - u_c + u_p}{d} = \frac{Z_c - f}{Z_c}$$

$$Z_c\left(d - u_c + u_p\right) = d\left(Z_c - f\right)$$

$$\cancel{Z_c d} - Z_c u_c + Z_c u_p = \cancel{d Z_c} - df$$

$$Z_c\left(u_p - u_c\right) = df$$

$$\Rightarrow \boxed{Z_c = \frac{df}{u_c - u_p}}$$

$$\frac{X_c}{Z_c} = \frac{X_c - U_c}{Z_c - f}$$

$$X_c = \frac{Z_c X_c - Z_c U_c}{Z_c - f}$$

$$X_c Z_c - X_c f = Z_c X_c - Z_c U_c$$

$$X_c f = Z_c U_c$$

$$X_c \cancel{f} = \frac{U_c d \cancel{f}}{U_c - U_p}$$

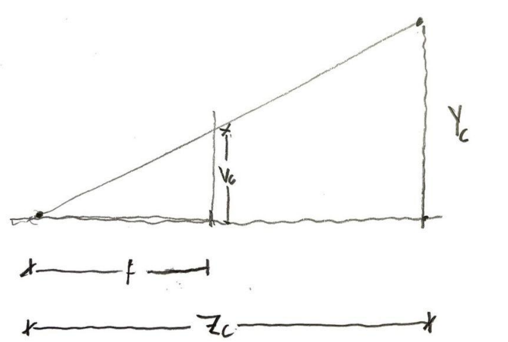
$$\Rightarrow \boxed{X_c = \frac{U_c d}{U_c - U_p}}$$

$$\frac{V_c}{f} = \frac{Y_c}{Z_c}$$

$$Y_c = \frac{Z_c V_c}{f}$$

$$Y_c = \frac{df}{V_c - U_p} \frac{V_c}{f} = \frac{d \cdot V_c}{V_c - U_p}$$

$$\boxed{\Rightarrow Y_c = \frac{d V_c}{U_c - U_p}}$$



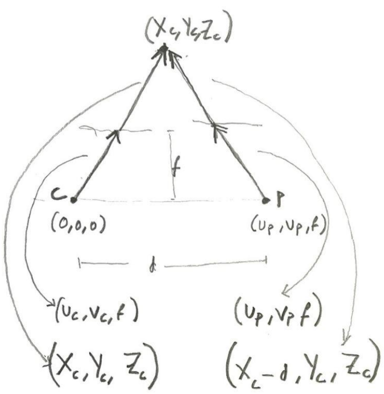
side view

Problem 2b

```
T1 = np.array([[1/f, 0, 0, 0], [0, 1/f, 0, 0], [0, 0, 0, 1], [0, 0, 1,0]])

T2 = np.array([[1, 0, 0, d], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])

T3 = np.array([[f, 0, 0, 0], [0, f, 0, 0],[0, 0, 1, 0]])

T = T3@(T2@T1)
```

Matrix Derivation example for T1

T_1 $\quad \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \end{bmatrix} \rightarrow \begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix}$



$$(X_c, Y_c, Z_c) = a(u_c, v_c, f)$$
$$(X_c - d, Y_c, Z_c) = a(u_p, v_p, f)$$
$$X_c = a u_c$$
$$X_c - d = a u_p$$

$$X_p = \frac{u_p}{f} \qquad Y_p = \frac{v_p}{f} \qquad Z_p = 1 \qquad w_p = \frac{1}{z_p}$$

$$= \boxed{\begin{bmatrix} \frac{1}{f} & 0 & 0 & 0 \\ 0 & \frac{1}{f} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}} \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \\ 1 \end{bmatrix}$$

$$\Downarrow$$

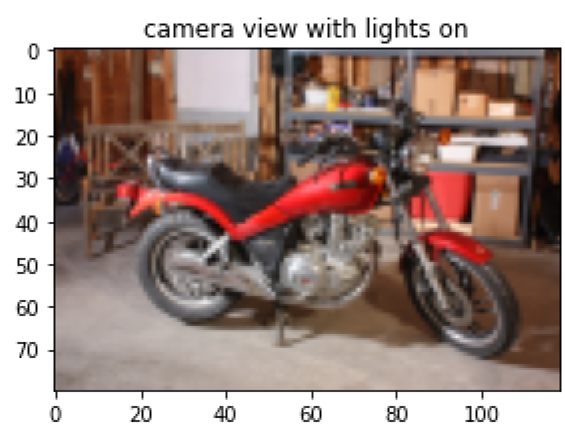
$$T_1$$

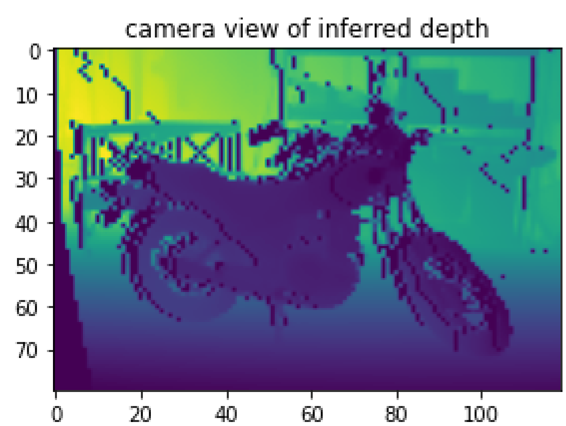
Problem 2c

Wiggling exists both in the vertical and horizontal simulations due to the depth changes in the image. Horizontally, the wiggle is due to the disparities and sudden large shifts along the X dimension. This disparity is further exaggerated in the vertical due to the larger change in X from the vertical view line.

Problem 2d

```
def inferDepthFromMatchedCoords(uv_c, uv_p, f, d):

    Z_c_arr = []
    for i in range(len(uv_c)):
      Z_c_list.append((d*f)/(uv_c[i][0]-uv_p[i][0]))

    return (np.array(Z_c_arr))
```

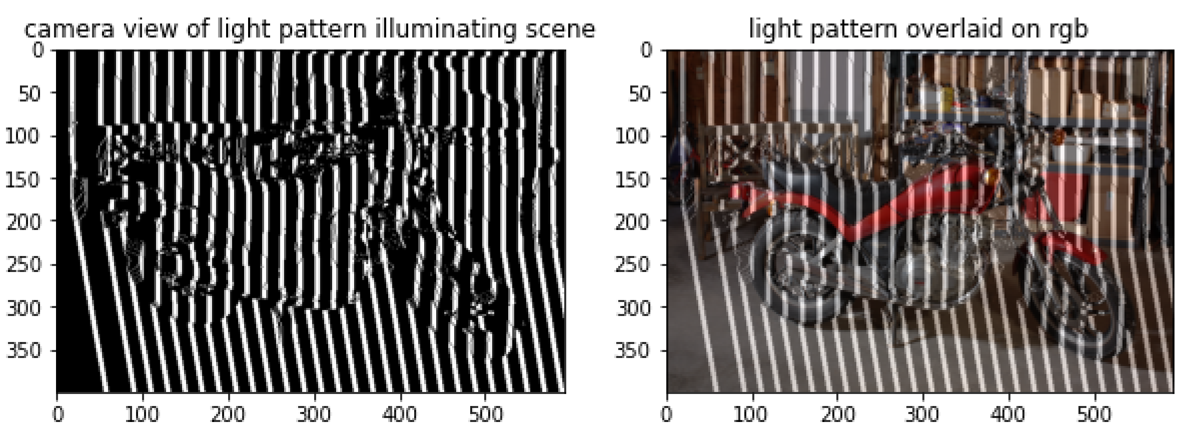




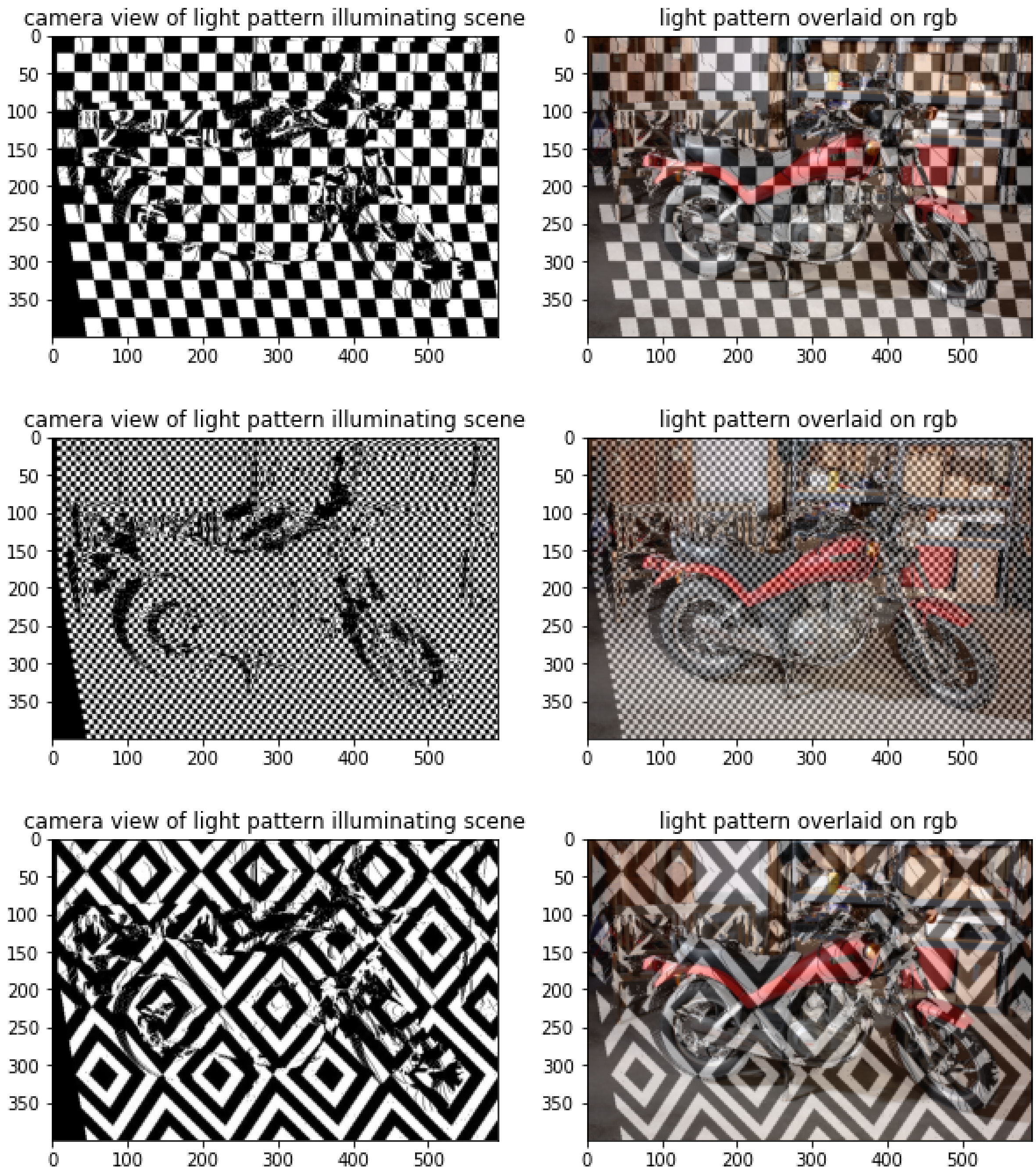
The result is a low resolution inferred depth image, so details are not being detected well. This is dependent on the resolution of the original image and the fact that the inferred image is a continuous pixel scan.

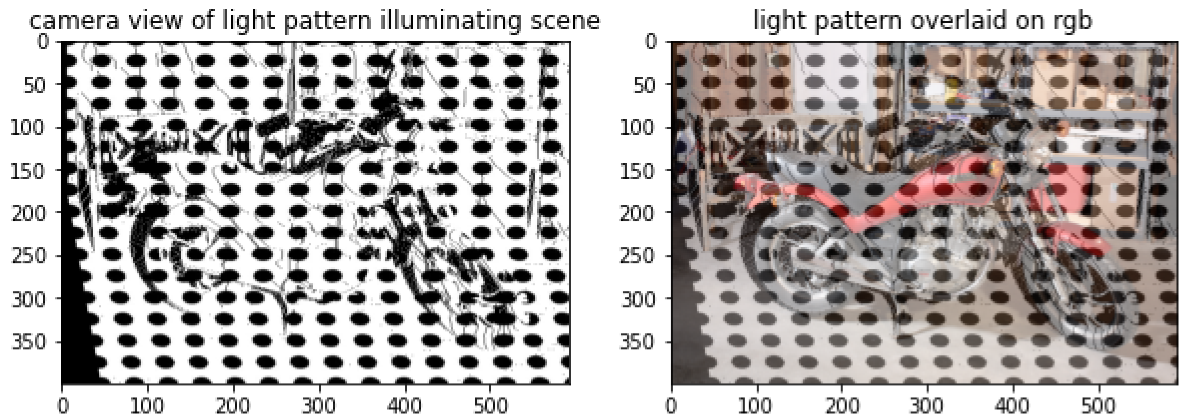
Problem 2e

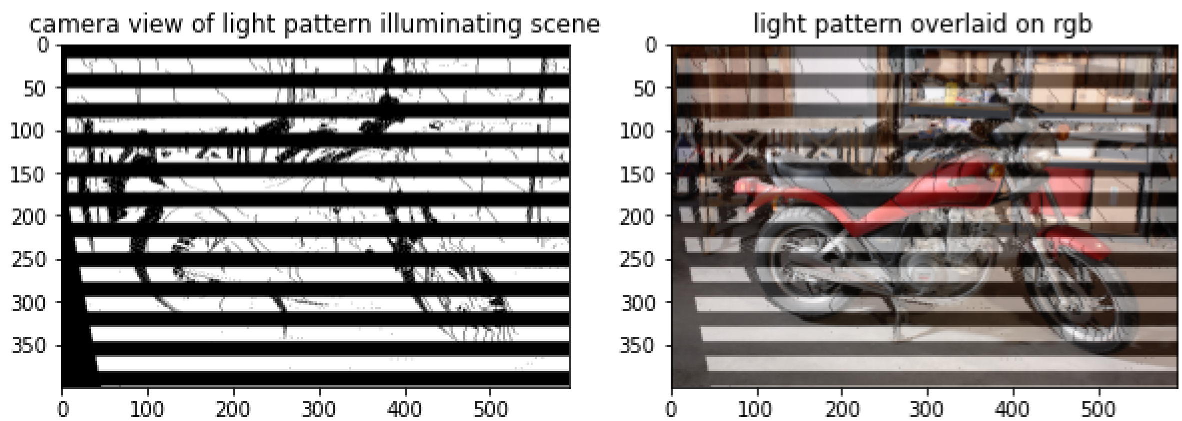
```
h = img_size[0] # extract image height = 400
w = img_size[1] # extract image height = 593
N = h * w

xy_p_arr = []

for y in range(0,h): # loop over the vertical
  for x in range (0,w): # loop over the horizontal
    xy_p_arr.append([x,y]) # add the position coordinates to the xy_p arr.
xy_p = np.array(xy_p_arr)


def getImgCoordinatePairs(Z_p_img, T, img_size, cx_p, cy_p, cx_c, cy_c):
    xy_p = np.array(xy_p_arr)
    Z_p_img_reshape = np.reshape(Z_p_img,(N,1))
    xy_c = transform_xy_p_to_xy_c(xy_p, Z_p_img_reshape, cx_p, cy_p, cx_c,
cy_c, T)

    return xy_c, xy_p
```



Occlusion happens as a result of the shift between the range camera and the projector or in stereo imaging where parts of the scene can only be seen by one camera, making some areas where we do not have information to be collected.


camera view of light pattern illuminating scene
light pattern overlaid on rgb
camera view of light pattern illuminating scene
light pattern overlaid on rgb
camera view of light pattern illuminating scene
light pattern overlaid on rgb



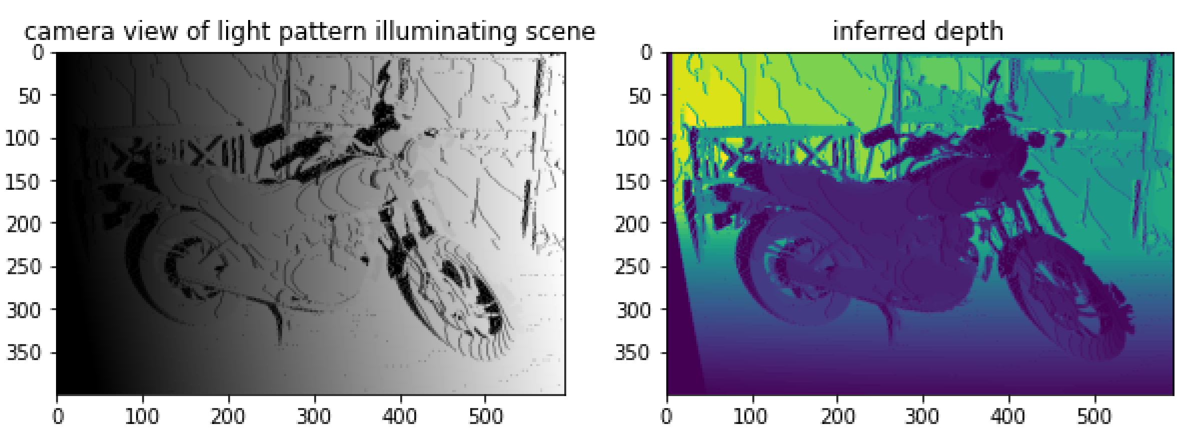


Horizontal lines did not yield good depth results since the scanning is happening horizontally.

The more detailed the projected pattern is, the better the depth results. Arbitrary or irregular patterns seem more difficult to project good results.

Problem 2f

```
def getStructuredLight(img_size):
    h = img_size[0]
    w = img_size[1]

    N = h * w

    xy_p_arr2 = []

    for y in range(0,h):
      for x in range (0,w):
        xy_p_arr2.append(x)
    xy_p_2 = np.array(xy_p_arr2)
    xy_p_2 = np.reshape(xy_p_2,(h,w))

    L_p_img = xy_p_2
    return L_p_img

def F(L_c, xy_c):
    xy_p_arr = []

    N = h * w
    for i in range(0,N):
      xy_p_arr.append([L_c[i],xy_c[i][1]])
    xy_p = np.array(xy_p_arr)
    return xy_p
```



Derivation:

$$\begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix} = T_1 \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \end{bmatrix} \qquad \begin{bmatrix} u_c \\ v_c \end{bmatrix} = T_3 \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = T_2 \begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix} \qquad \begin{bmatrix} u_c \\ v_c \end{bmatrix} = T_3 \times T_2 \times T_1 \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \end{bmatrix}$$

$$T_1 \cdot T_2 \cdot T_3 \cdot \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \\ 1 \end{bmatrix}$$

$$\begin{array}{cccc} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 & d \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} \frac{1}{f} & 0 & 0 & 0 \\ 0 & \frac{1}{f} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \\ 1 \end{bmatrix} \\ T_3 & T_2 & T_1 & \\ 3 \times 4 & 4 \times 4 & 4 \times 4 & 4 \times 1 \end{array}$$

$$\begin{bmatrix} f & 0 & 0 & fd \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{f} & 0 & 0 & 0 \\ 0 & \frac{1}{f} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \end{bmatrix}$$

$$\begin{bmatrix} u_c \\ v_c \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & fd & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ \frac{1}{z_p} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u_c \\ v_c \end{bmatrix} = \begin{bmatrix} u_p + \frac{fd}{z_p} \\ v_p \\ 1 \end{bmatrix} \Rightarrow \begin{aligned} u_c &= u_p + \frac{fd}{z_p} \\ v_c &= v_p \end{aligned}$$

Another strategy?

An invertible function of u_p as intensity would work, taking the inverse of the function on the camera to find u_p.