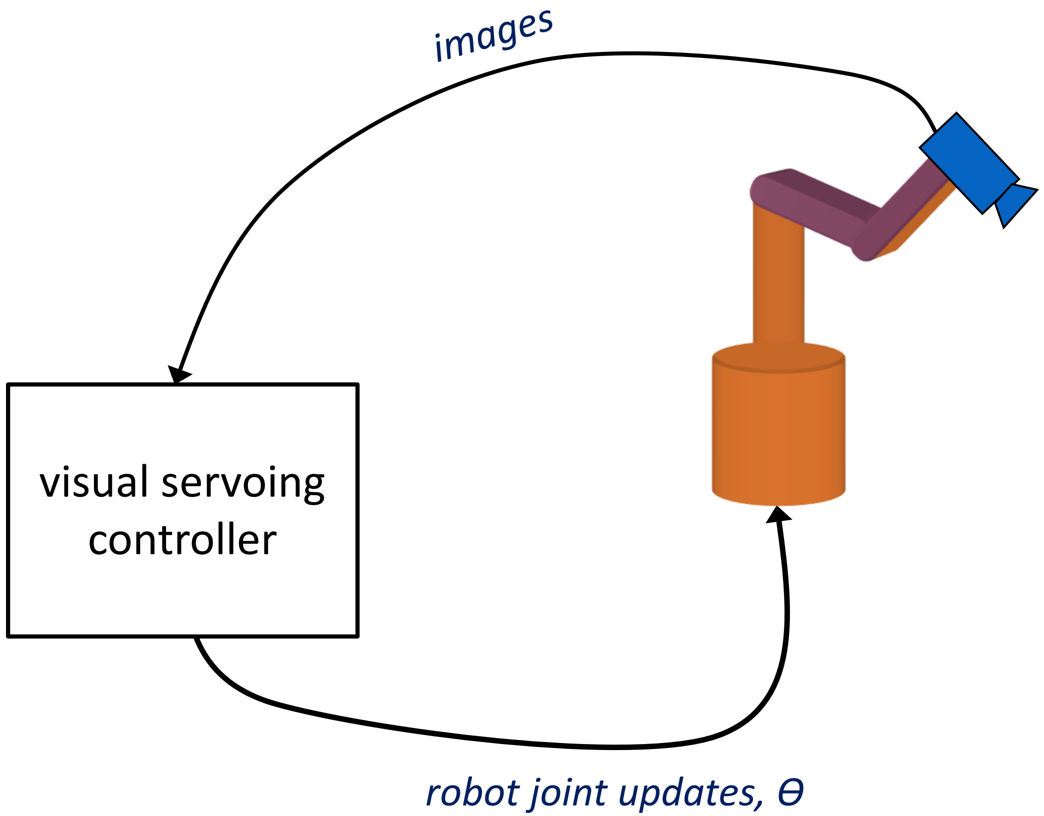
Visual servoing (VS) control uses real-time camera images as feedback.

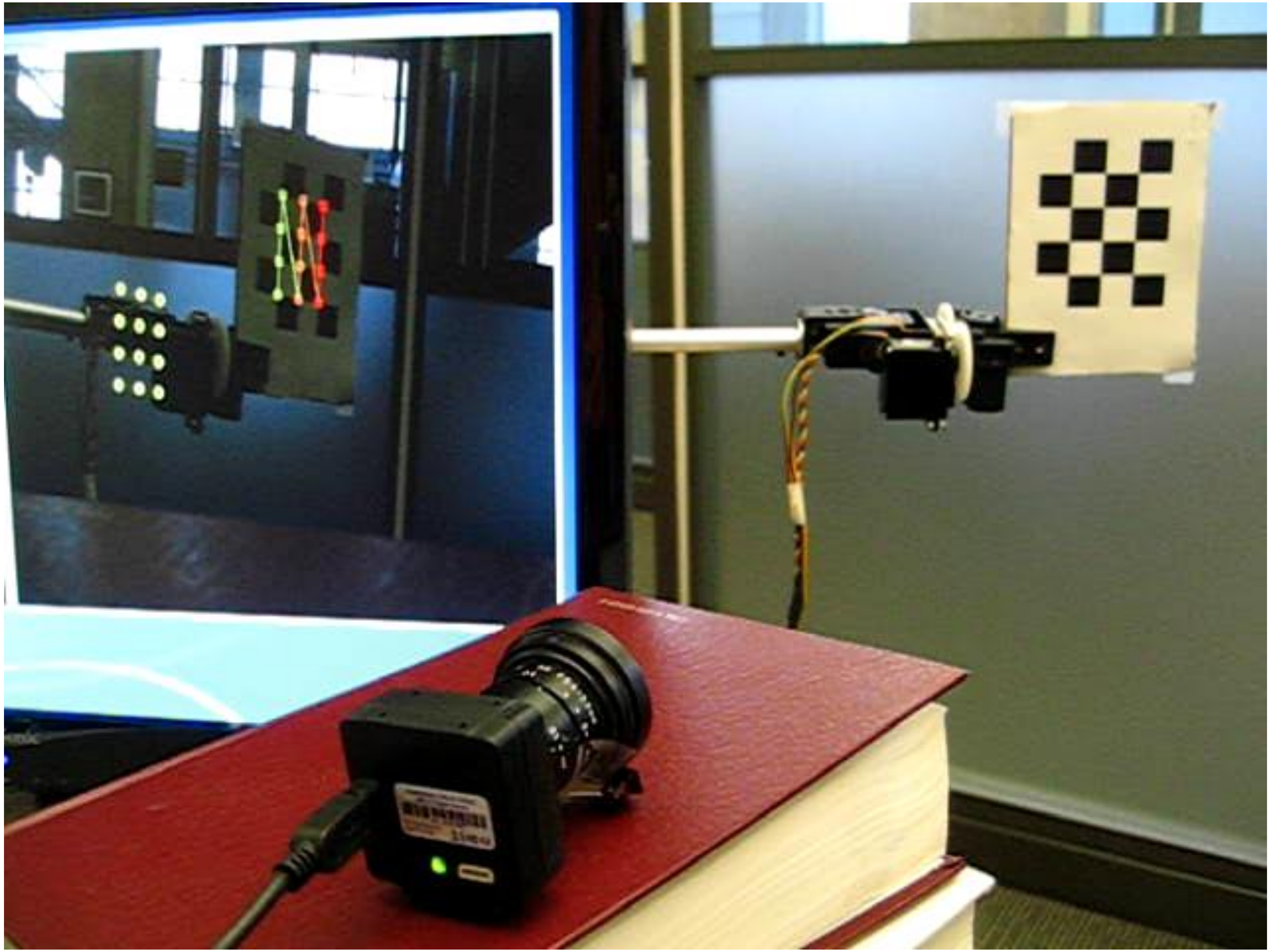


Learns mapping from image space to robot joint space.

Camera calibration not needed.

Robot model not needed.

desired locations of checkboard corners in image



real-time object detection of checkerboard corners

non eye-in-hand example *[VIDEO]*

Mathematical details:

$$V = J(\Theta)\dot{\Theta}$$

where V = Cartesian velocity vector of robot end-effector

J = Jacobian matrix

Θ = robot joint angles .

Rigidly attach a camera onto end-effector:

$$\dot{e} = J'(\Theta)\dot{\Theta}$$

where e = vector of image features (generally the feature error)

J' = image Jacobian matrix .

J' units: change in pixels / change in axis (rad or m or PSI)

$$\dot{\Theta} = J'(\Theta)^{-1}\dot{e}$$

To obtain exponential feature error decrease, desire:

$$\dot{e} = -\lambda\, e \quad \text{, where } \lambda > 0 \quad .$$

Visual servoing iteratively **updates the commanded robot joint vector** Θ:

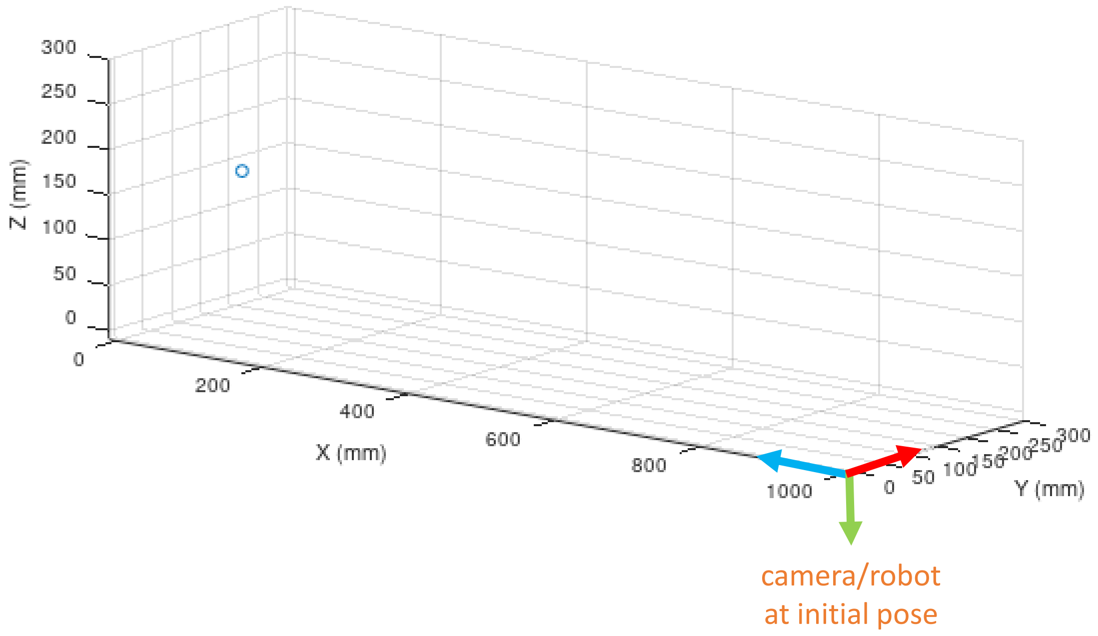
$$\Theta_k = \Theta_{k-1} - \lambda\, J'(\Theta)^{-1} e_k \quad .$$

updated robot axes = previous robot axes + correction term

EXAMPLE: 2-DOF visual servoing

one-point physical target locations (w.r.t. XYZ world coordinate frame in mm):

P1 = (110, 69.904, 178.923)



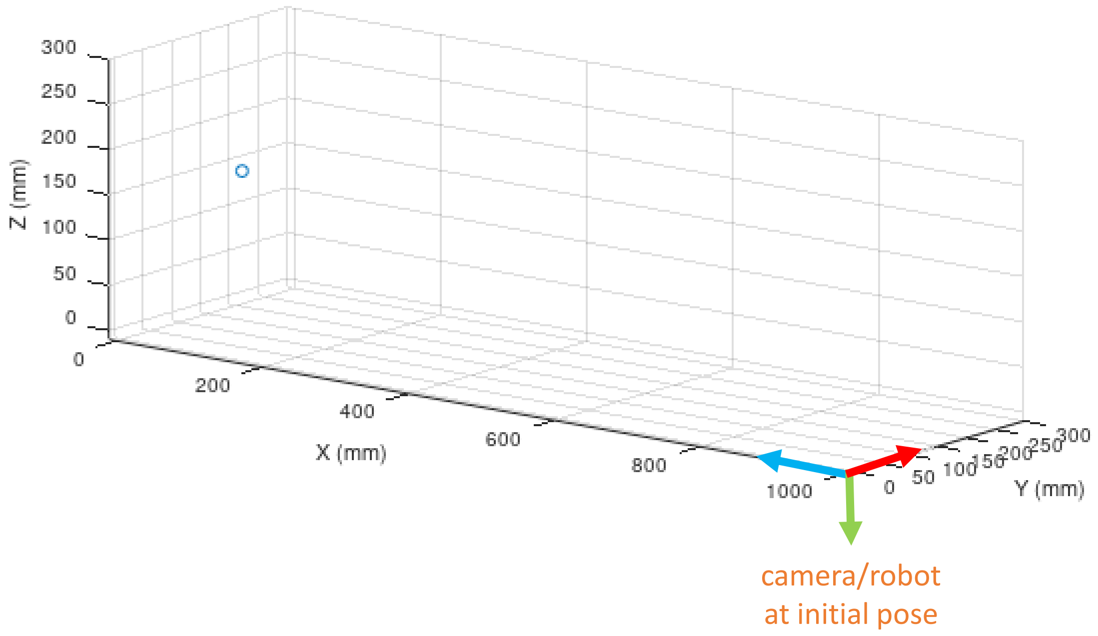
2-DOF "flying robot": D435 camera that can move along Y & Z (only)

Task: implement visual servoing method to actuate the 2 robot axes so that the single target point <u>coincides</u> with the center pixel location (640,360) in image space.

EXAMPLE: 2-DOF visual servoing

one-point physical target locations (w.r.t. XYZ world coordinate frame in mm):

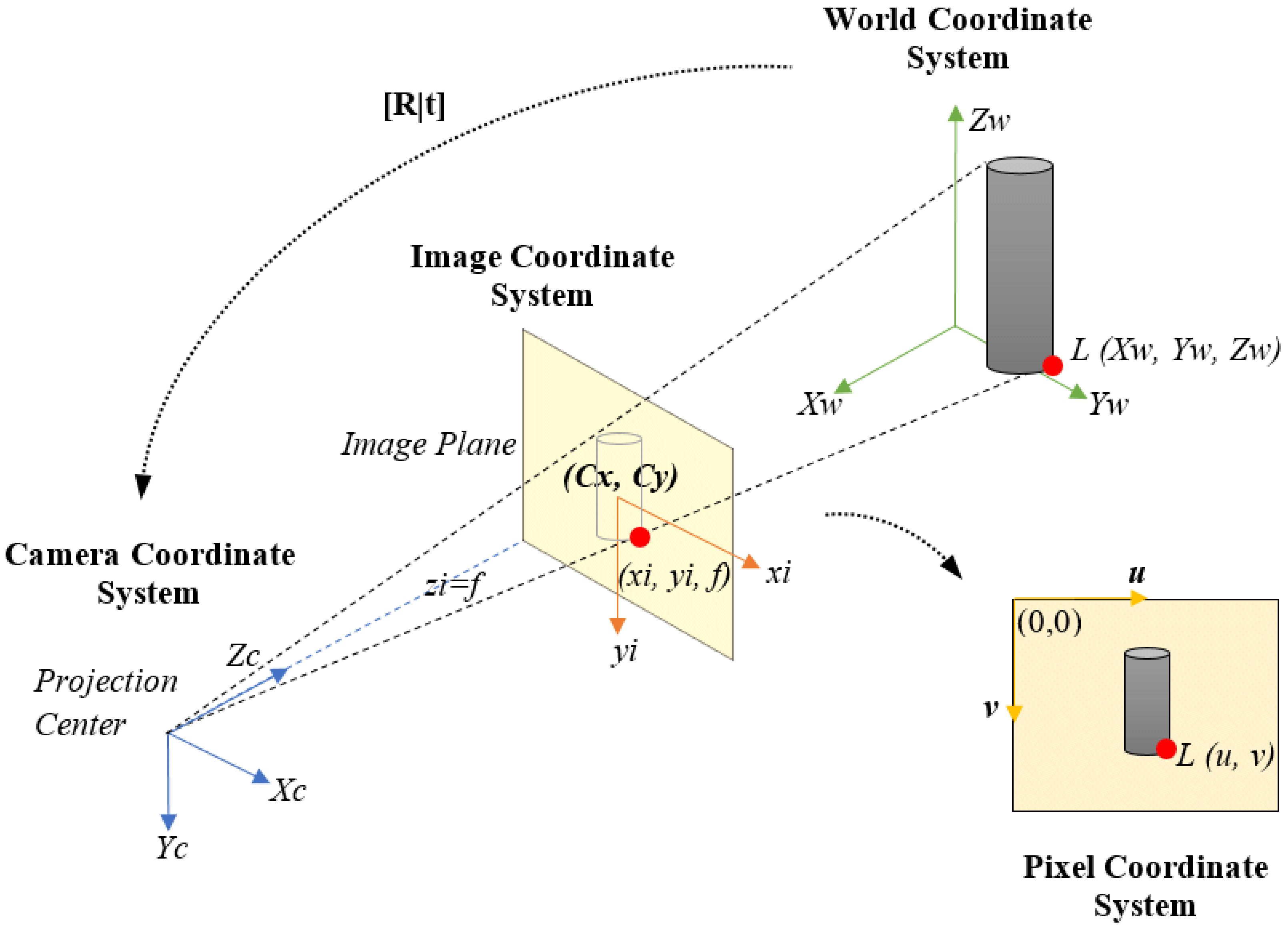
P1 = (110, 69.904, 178.923)

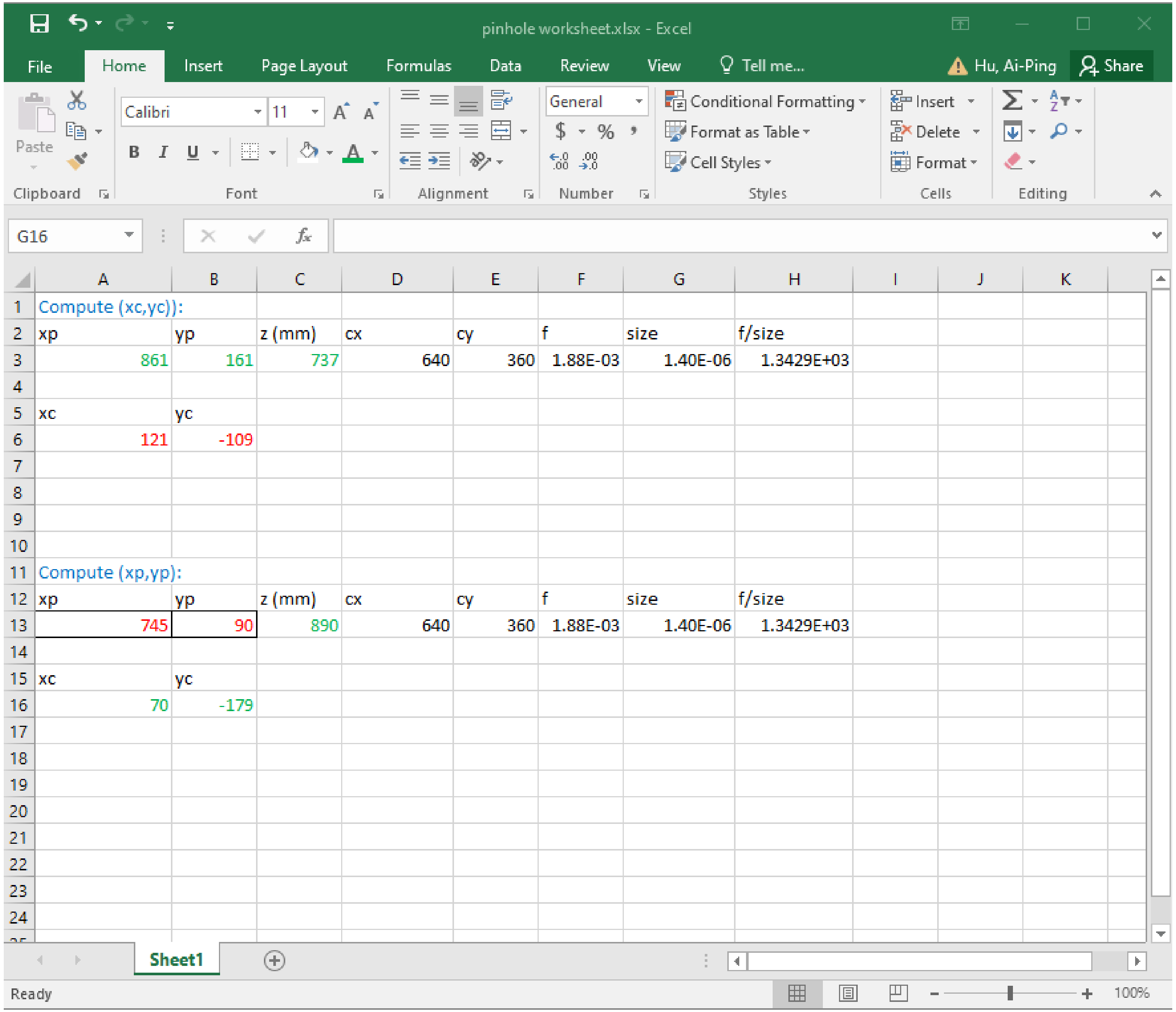


2-DOF "flying robot": D435 camera that can move along Y & Z (only)

Task: implement visual servoing method to actuate the 2 robot axes so that the single target point <u>coincides</u> with the center pixel location (640,360) in image space.

From inspection, require: robot (Y,Z) = (69.904,178.923)


World Coordinate System
[R|t]
Zw
Image Coordinate System
L (Xw, Yw, Zw)
Xw
Yw
Image Plane
(Cx, Cy)
xi
(xi, yi, f)
Camera Coordinate System
zi=f
yi
u
(0,0)
v
L (u, v)
Zc
Projection Center
Xc
Yc
Pixel Coordinate System



	A	B	C	D	E	F	G	H
1	Compute (xc,yc)):							
2	xp	yp	z (mm)	cx	cy	f	size	f/size
3	861	161	737	640	360	1.88E-03	1.40E-06	1.3429E+03
4								
5	xc	yc						
6	121	-109						
11	Compute (xp,yp):							
12	xp	yp	z (mm)	cx	cy	f	size	f/size
13	745	90	890	640	360	1.88E-03	1.40E-06	1.3429E+03
14								
15	xc	yc						
16	70	-179						

EXAMPLE: 2-DOF visual servoing (continued)

Initial pixel coordinates of P1 can be computed as: (745,90).

Calculate image Jacobian numerically by perturbing each axis separately and noting change in image space of P1.

Perturb from Y = 0 to Y = +3 mm. New P1 pixel coordinates are: (741,90).

Return to initial pose.

Perturb from Z = 0 to Z = -3 mm. New P1 pixel coordinates are: (745,86).

Form image Jacobian matrix (once):

$J' = [dx_p/dY \; dx_p/dZ; \; dy_p/dY \; dy_p/dZ]$ *2 × 2 matrix*

$= [(741-745)/3 \; (745-745)/(-3); \; (90-90)/3 \; (86-90)/(-3)]$

$= [-4/3 \;\; 0; \; 0 \;\; 4/3]$

Take inverse:

$inv(J') = [-0.75 \;\; 0; \; 0 \;\; 0.75]$

Update 2-DOF robot axes via VS (set $\lambda = 1$):

image error vector, e:
actual – desired pixel coordinates

$[Y_{new}; Z_{new}] = [Y_{prev}; Z_{prev}] - inv(J')*[745 - 640; 90 - 360]$

$= [0; 0] - inv(J')*[745 - 640; 90 - 360]$

$= [78.75 \text{ mm}; 202.5 \text{ mm}]$

→ corresponds to P1 pixel coordinates of (633,355)

Additional VS iterations:

$[Y_{new}; Z_{new}] = [78.75; 202.5] - inv(J')*[627 - 640; 396 - 360]$

$= [65.75 \text{ mm}; 166.5 \text{ mm}]$

→ corresponds to P1 pixel coordinates of (646,341)

$[Y_{new}; Z_{new}] = [65.75; 166.5] - inv(J')*[646 - 640; 341 - 360]$

$= [59.75 \text{ mm}; 185.5 \text{ mm}]$

→ corresponds to P1 pixel coordinates of (655,370)

$[Y_{new}; Z_{new}] = [59.75; 185.5] - inv(J')*[655 - 640; 370 - 360]$

$= [74.75 \text{ mm}; 175.5 \text{ mm}]$

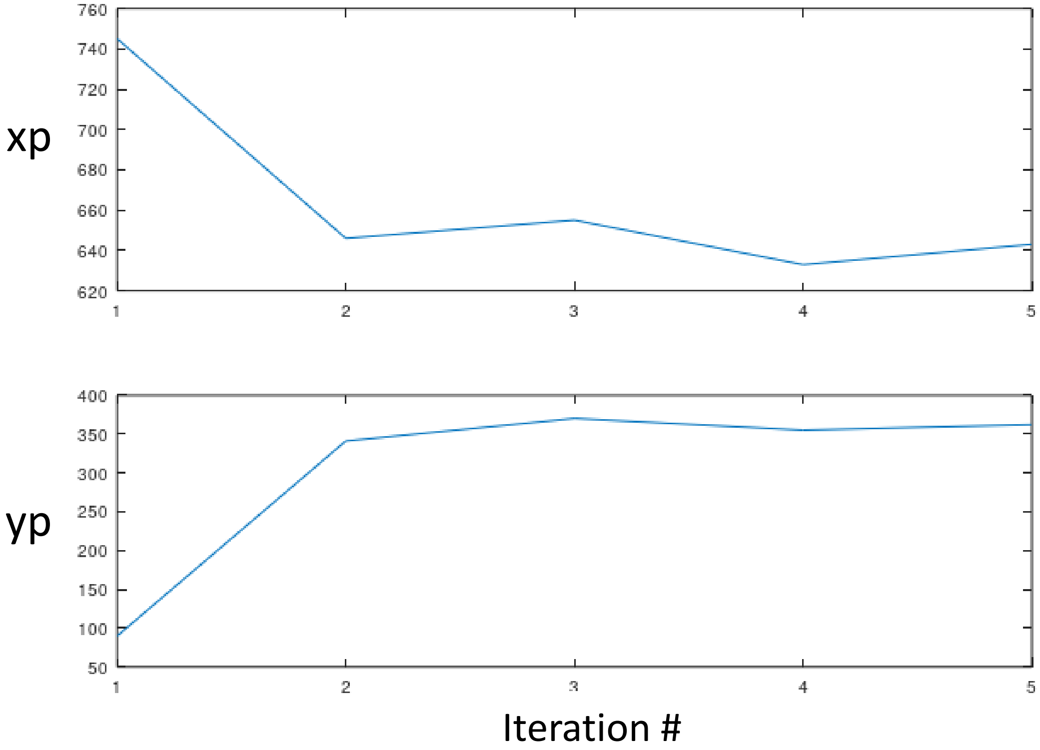
→ corresponds to P1 pixel coordinates of (633,355)

$[Y_{new}; Z_{new}] = [74.75; 175.5] - inv(J')*[633 - 640; 355 - 360]$

$= [67.75 \text{ mm}; 180.5 \text{ mm}]$

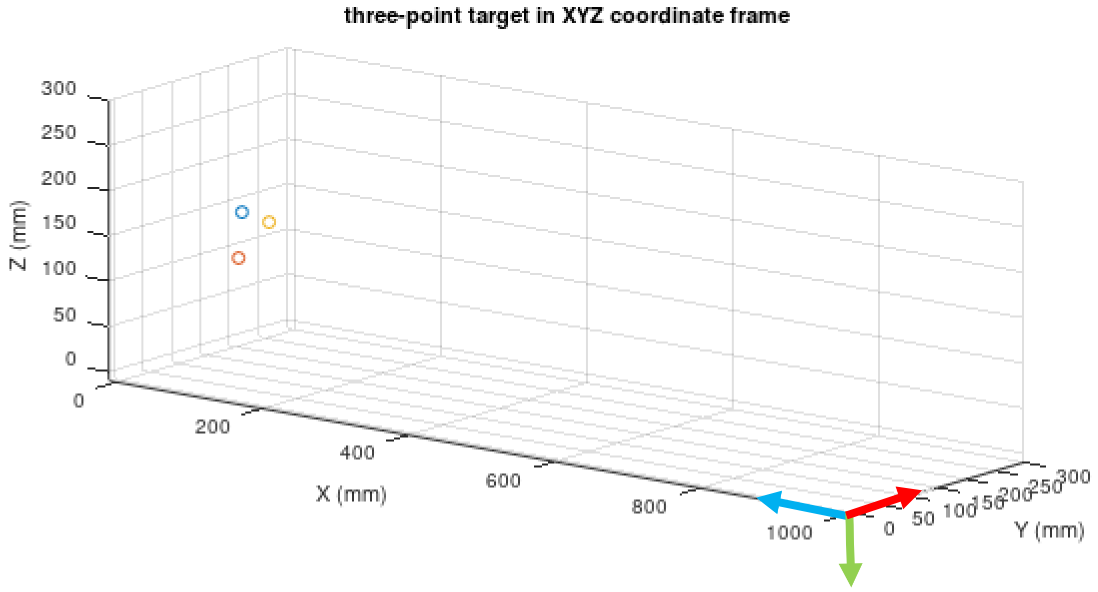
compare to known solution of (Y,Z) = (69.904,178.923)

→ corresponds to P1 pixel coordinates of (643,362)

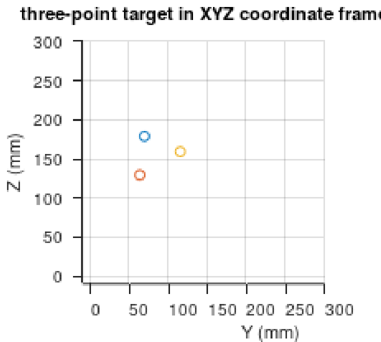


three-point physical target locations (w.r.t. XYZ world coordinate frame in mm):

P1 = (110, 69.904, 178.923)
P2 = (110, 63.923, 129.282)
P3 = (110, 115.885, 159.282)



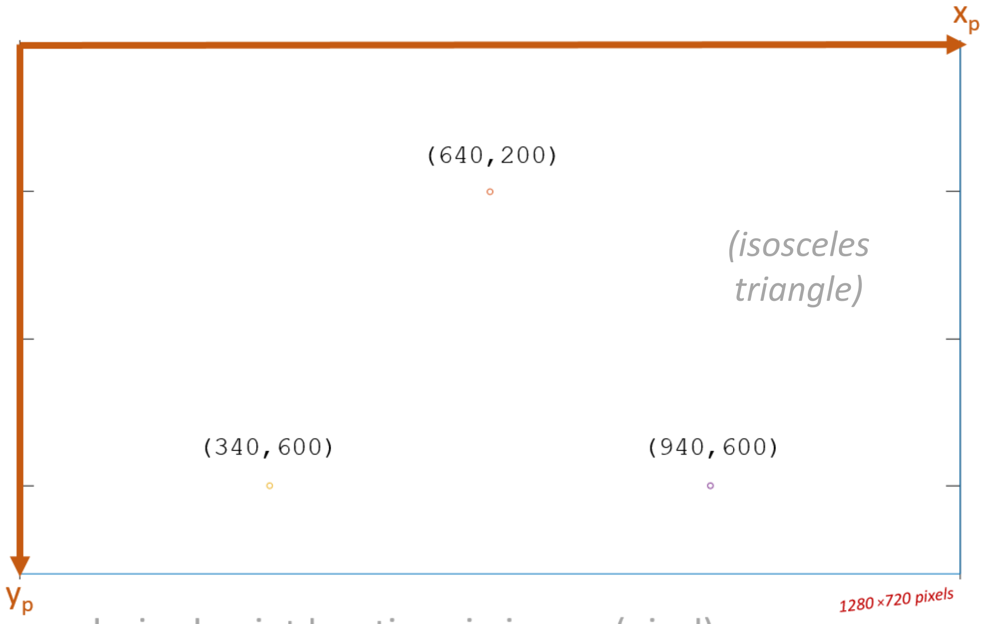
camera/robot at initial pose



4-DOF "flying robot": D435 camera that can move along X, Y, & Z and can rotate about z_c

Initial robot pose: XYZ = (1000, 0, 0) and z_c is parallel to -X and x_c is parallel to +Y (see figure at left)

Task: implement method to actuate the 4 robot axes so that the three target points <u>coincide</u> with the desired pixel locations in image space (below).



desired point locations in image (pixel) space