

# Vision Strategies and "Tools You Can Use"

RSS II Lecture 5  
September 16, 2005  
Prof. Teller

## Today

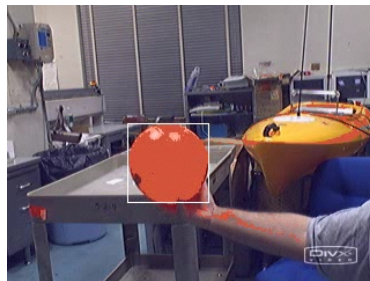
- Strategies
  - What do we want from a vision system?
- Tools
  - Pinhole camera model
  - Vision algorithms
- Development
  - Carmen Module APIs, brainstorming

## Vision System Capabilities

- Material identification
  - Are there bricks in vicinity? If so, where?
- Motion freedom
  - What local motion freedom does robot have?
- Manipulation support
  - Is brick pose amenable to grasping/placement?
  - Is robot pose correct for grasping/placement?
- Localization
  - Where is robot, with respect to provided map?
  - Which way to home base? To a new region?
  - Has robot been in this region before?

## Material Identification

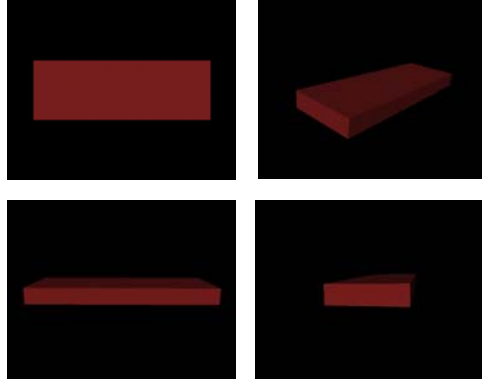
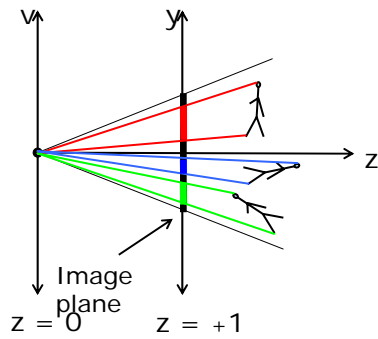
- Detecting bricks when they're present
  - How?



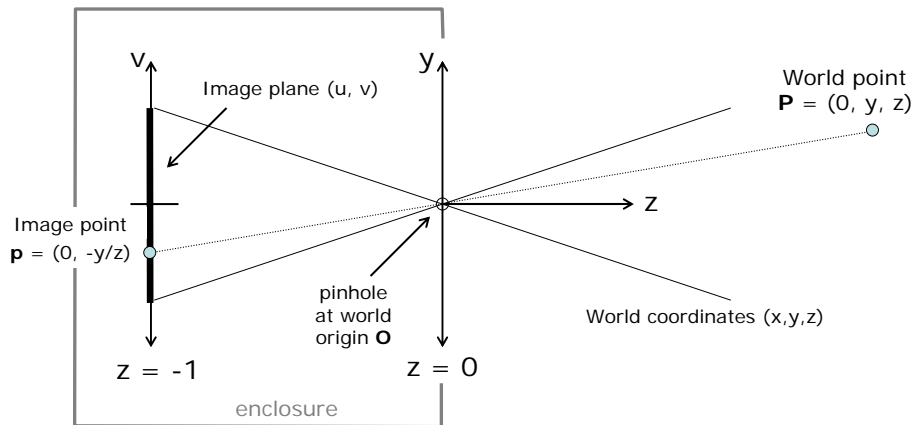
- Locate bricks
  - In which coordinate system?
  - Estimate range and bearing – how?

## Gauge distance from *apparent size*?

- Yes – under what assumption?



## Pinhole Camera Model (physical)

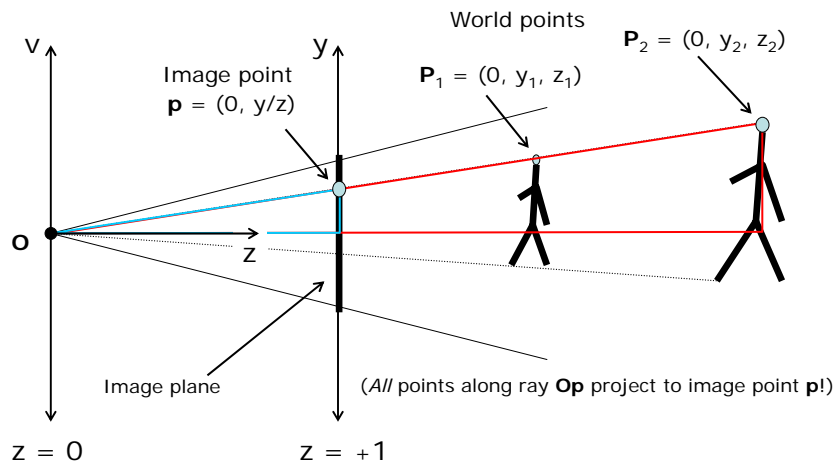


Notes:

- Diagram is drawn in the plane  $x=0$
- Image-space  $u$ -axis points out of diagram
- World-space  $x$ -axis points out of diagram
- Both coordinate systems are left-handed

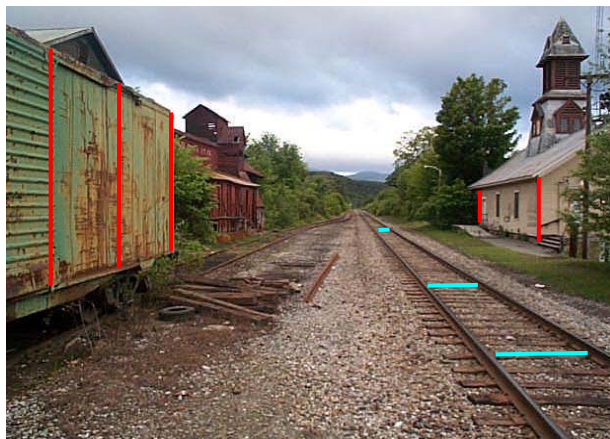
# Pinhole Camera Model (virtual)

("Virtual" image plane placed 1 unit in front of pinhole; no inversion)



## Perspective: Apparent Size

- Apparent object size decreases with *depth* (perp. distance from camera image plane)



## Perspective: Apparent Size

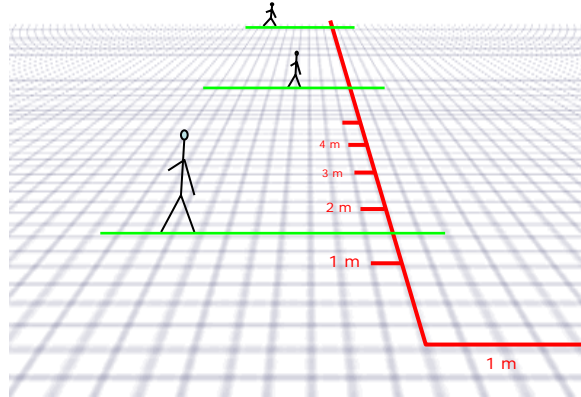


## What assumptions yield depth?



# Ground plane assumption

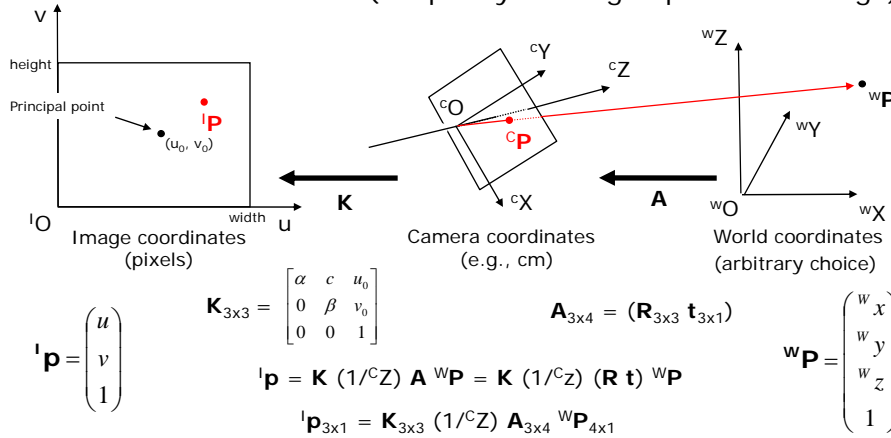
- Requires additional “metric” information
  - Think of this as a constraint on camera, world structure
- Plane in scene, with two independent marked lengths
  - Can measure distance to, or size of, objects on the plane



– ... but where do the marked lengths come from?

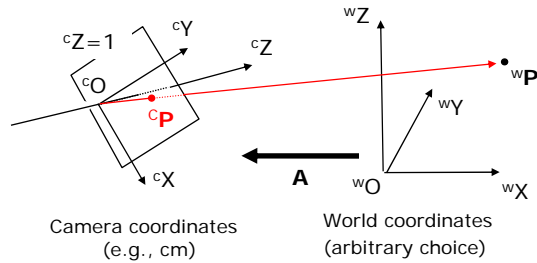
# Camera Calibration

- Maps 3D world points  ${}^wP$  to 2D image plane  ${}^iP$
- Map can be factored into two operations:
  - *Extrinsic* (rigid-body) calibration (situates camera in world)
  - *Intrinsic* calibration (warps rays through optics onto image)



# World-to-Camera Transform

- Relabels world-space points w.r.t. camera body
  - *Extrinsic* (rigid-body) calibration (situates camera in world)



$${}^c\mathbf{P} = \begin{pmatrix} c_x \\ c_y \\ 1 \end{pmatrix}$$

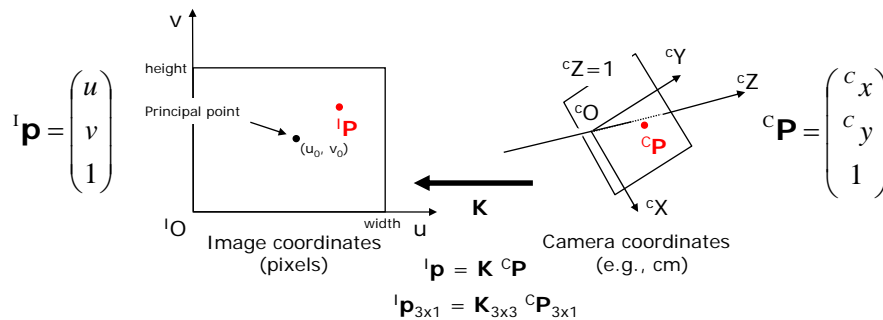
$$\begin{aligned} \mathbf{A}_{3 \times 4} &= (\mathbf{R}_{3 \times 3} \ \mathbf{t}_{3 \times 1}) \\ {}^c\mathbf{P} &= (1/c_z) (\mathbf{R} \ \mathbf{t}) \ \mathbf{wP} \\ {}^c\mathbf{P}_{3 \times 1} &= (1/c_z) \mathbf{A}_{3 \times 4} \ \mathbf{wP}_{4 \times 1} \end{aligned}$$

$$\mathbf{wP} = \begin{pmatrix} w_x \\ w_y \\ w_z \\ 1 \end{pmatrix}$$

Note effect of division by  $c_z$ : no scaling necessary!

# Camera-to-Image Transform

- Maps 2D camera points to 2D image plane
  - Models ray path through camera optics and body to CCD



$${}^i\mathbf{p} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

$${}^c\mathbf{P} = \begin{pmatrix} c_x \\ c_y \\ 1 \end{pmatrix}$$

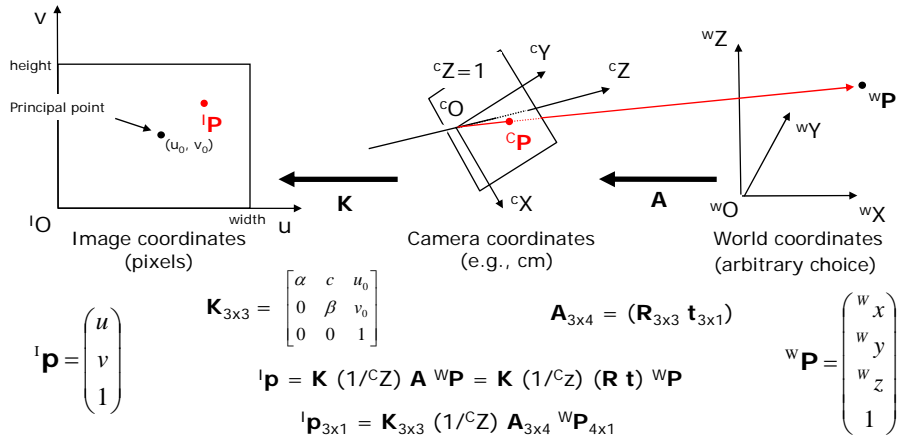
$$\begin{aligned} {}^i\mathbf{p} &= \mathbf{K} \ {}^c\mathbf{P} \\ {}^i\mathbf{p}_{3 \times 1} &= \mathbf{K}_{3 \times 3} \ {}^c\mathbf{P}_{3 \times 1} \end{aligned}$$

Matrix  $\mathbf{K}$  captures the camera's *intrinsic parameters*:


$$\mathbf{K}_{3 \times 3} = \begin{bmatrix} \alpha & c & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

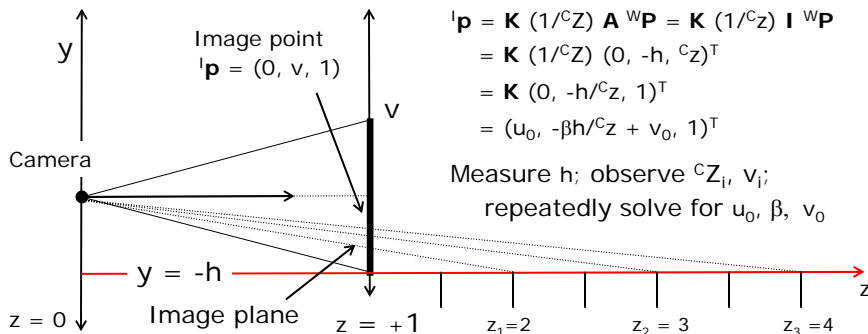
- $\alpha, \beta$ : horizontal, vertical scale factors (equal iff pixel elements are square)
- $u_0, v_0$ : principal point, i.e., point at which optical axis pierces image plane
- $c$ : image element (CCD) skew, usually  $\sim 0$

# End-to-End Transformation



## Example: Metric Ground Plane

- Make camera-frame and world-frame coincident  
Thus  $R = I_{3 \times 3}$ ,  $t = O_{3 \times 1}$ ,  $A_{4 \times 4} = (R \ t)$  as before
- Lay out a tape measure on line  $x = 0, y = -h$
- Mark off points at (e.g.) 50-cm intervals 
- What is the functional form of map  $u = f(w_x, w_y, w_z)$ ?





## Vision System Capabilities

- Material identification
  - Are there bricks in vicinity? If so, where?
- Motion freedom
  - What local motion freedom does robot have?
- Manipulation support
  - Is brick pose amenable to grasping/placement?
  - Is robot pose correct for grasping/placement?
- Localization
  - Where is robot, with respect to provided map?
  - Which way to home base? To a new region?
  - Has the robot been in this region before?

## Motion Freedom



- What can be inferred from image?

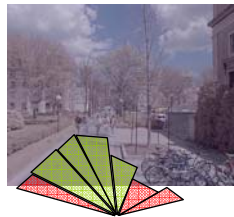
## Freespace Map



- Discretize bearing; classify surface type

## Freespace Map Ideas

- Use simple color classifier
  - Train on road, sidewalk, grass, leaves etc.
  - Training could be done offline, or in a start-of-mission calibration phase adapted from RSS II Lab 2



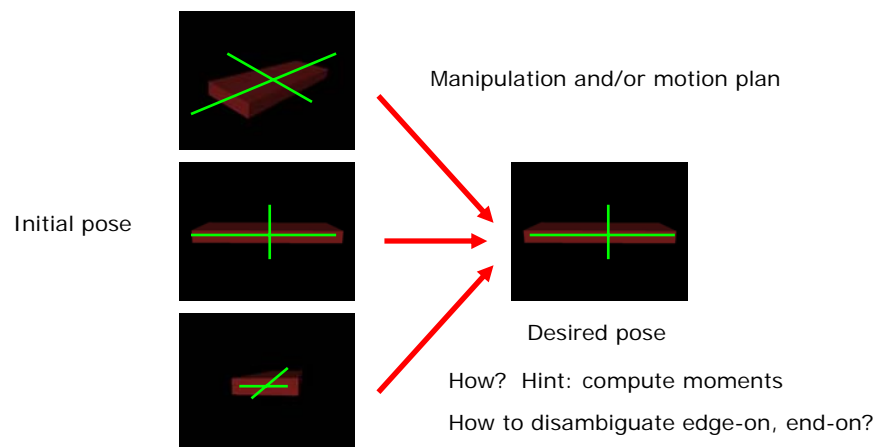
- For each wedge of disk, could report distance to nearest obstruction
- Careful: how will your code deal with varying lighting conditions?
- Finally: can fuse (or confirm) with laser data

## Vision System Capabilities

- Material identification
  - Are there bricks in vicinity? If so, where?
- Motion freedom
  - What local motion freedom does robot have?
- Manipulation support
  - Is brick pose amenable to grasping/placement?
  - Is robot pose correct for grasping/placement?
- Localization
  - Where is robot, with respect to provided map?
  - Which way to home base? To a new region?
  - Has the robot been in this region before?

## Manipulation Support

- Two options
  - Manipulate brick into appropriate grasp pose
  - Plan motion to approach the (fixed-pose) brick

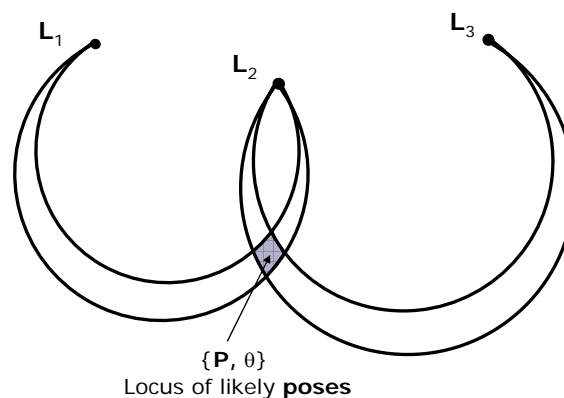


## Vision System Capabilities

- Material identification
  - Are there bricks in vicinity? If so, where?
- Motion freedom
  - What local motion freedom does robot have?
- Manipulation support
  - Is brick pose amenable to grasping/placement?
  - Is robot pose correct for grasping/placement?
- Localization
  - Where is robot, with respect to provided map?
  - Which way to home base? To a new region?
  - Has the robot been in this region before?

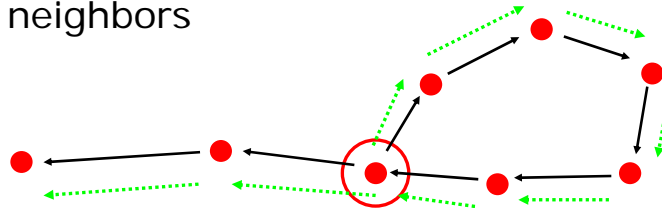
## Localization support

- Localization w.r.t. a known map
  - See localization lecture from RSS I
  - Features: curb cuts, vertical building edges
    - Map format not yet defined – one of your tasks



## Localization support

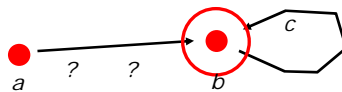
- Weaker localization model
  - Create (virtual) landmarks at intervals ●
  - Chain each landmark to predecessor ↖
  - Recognize when landmark is revisited ○
  - Record *direction* from landmark to its neighbors ↗



- ... Is this map *topological* or *metrical*?
- ... Does it support homing? Exploration?

## Visual basis for landmarks

- Desire a visual property that:
  - Is nearly invariant to *large* robot rotations
  - Is nearly invariant to *small* robot translations
  - Has a definable scalar distance  $d(b,c)$  [why?]



- Possible approaches:
  - Hue histograms (coarsely discretized)
  - Ordered hue lists (e.g., of vertical strips)
  - Skylines (must segment ground from sky)
  - Some hybrid of vision, laser data
  - Careful: think about *ambiguity* in scene

## Carmen Module APIs

- Vision module handles (processes) image stream
  - Must export more compact representation than images
- What representation(s) should module export?
  - Features? Distances? Landmarks? Directions? Maps?
- What questions should module answer?
  - Are there collectable blocks nearby? Where?
  - Has robot been here before? With what confidence?
  - Which direction(s) will get me closer to home?
  - Which direction(s) will explore new regions?
  - Which directions are physically possible for robot?
- What non-vision data should module accept?
  - Commands to establish a new visual landmark?
  - Notification of rotation in place? Translation?
- Spiral development
  - Put simple APIs in place, even if performance is stubbed
    - Get *someone else* to exercise them; revise appropriately

## Conclusion

- One plausible task decomposition
- Bottom-up operating scenario
- Related existing, new vision tools
- Functional view: what are APIs?
- Exhortation to spiral development