10) Derivation of the Motion Tracking Equation

- Brightness Constancy : A scence point keeps the same brightness as it moves slightly between times "t" and "t+Δt".

$$I(x, y, t) = I(x+u, y+v, t+\Delta t)$$

where,

$(u, v) \rightarrow$ Small displacement.

- Assuming u, v are small distances, Lets expand an RHS.

$$I(x+u, y+v, t+\Delta t) \approx I(x, y, t) + I_x u + I_y v + I_t \Delta t$$

$$I_x = \frac{dI}{dx}, \quad I_y = \frac{dI}{dy}, \quad I_t = \frac{dI}{dt}$$

Substitute this into brightness constancy.

$$I(x, y, t) \approx I(x, y, t) + I_x u + I_y v + I_t \Delta t$$

$$I_x u + I_y v + I_t \Delta t \approx 0$$

- for a unit time frame, $\Delta t = 1$

$$I_x u + I_y v + I_t = 0$$

$$u = \frac{-S_{xt} S_{yy} + S_{xy} S_{yt}}{\Delta}, \quad v = \frac{-S_{yt} S_{xx} + S_{xy} S_{xt}}{\Delta}$$

In [1]:
```
# Cell 0: setup
import os, shutil, time, glob, math
from pathlib import Path

import cv2
import numpy as np
import matplotlib.pyplot as plt

# Where to save things
BASE = Path("oflow_session")
FRAMES_DIR = BASE / "frames"
PATCHES_DIR = BASE / "patches"

for p in (FRAMES_DIR, PATCHES_DIR):
    p.mkdir(parents=True, exist_ok=True)

def imshow_rgb(img, title=None, size=6):
    plt.figure(figsize=(size, size))
    if img.ndim == 2:
        plt.imshow(img, cmap="gray")
    else:
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    if title: plt.title(title)
    plt.axis("off")
    plt.show()
```

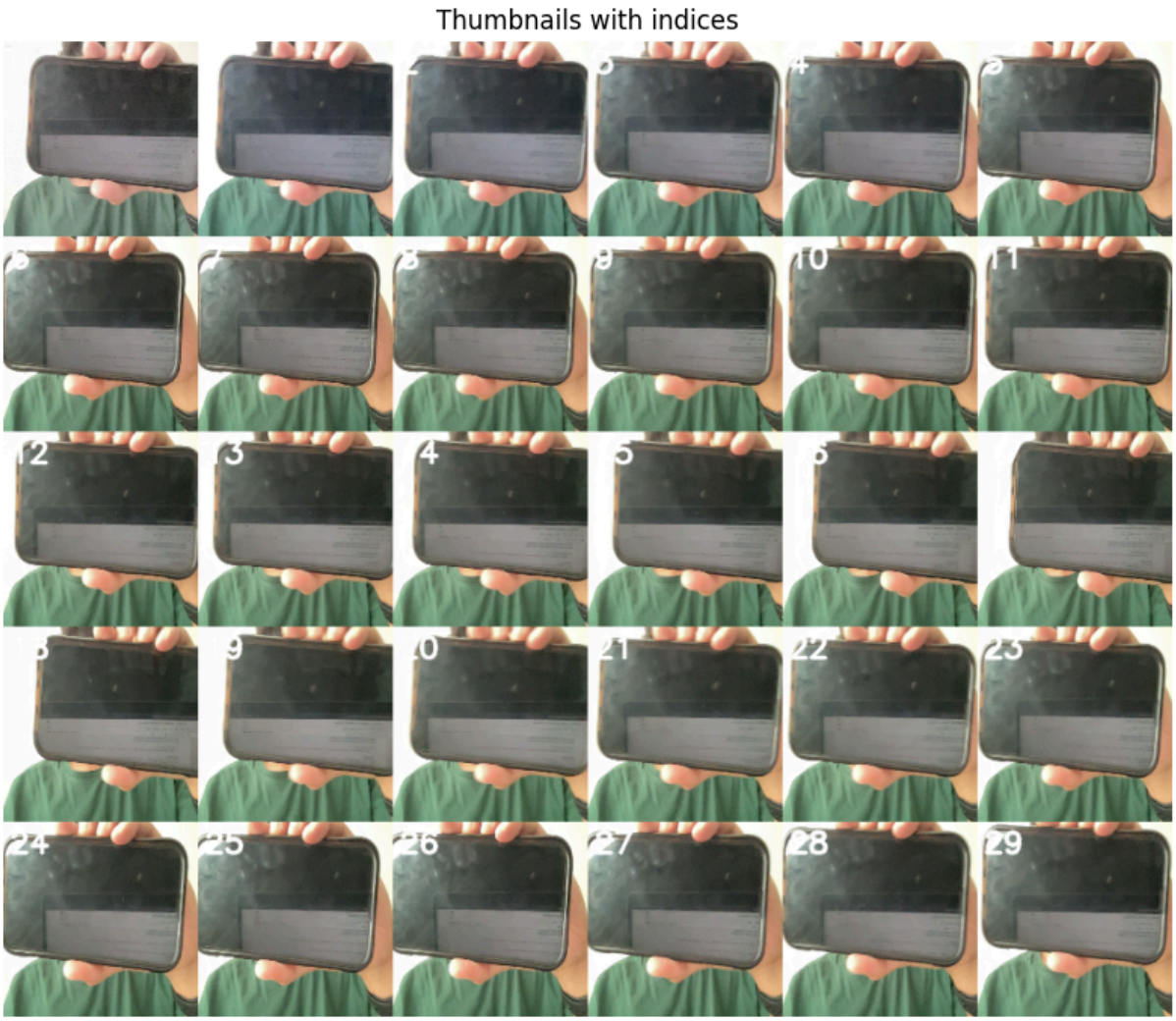
In [9]:
```
# Cell 1: capture ~30 frames to memory (webcam index 0 by default)
# If you don't have a webcam, skip this and use the "from video file" cell b

NUM_FRAMES = 30
CAM_INDEX = 1
CAP_WIDTH, CAP_HEIGHT = 640, 480

cap = cv2.VideoCapture(CAM_INDEX)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, CAP_WIDTH)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, CAP_HEIGHT)

if not cap.isOpened():
    raise RuntimeError("Could not open camera. If on a remote VM, use the 'f

frames_mem = []
print("[INFO] Capturing frames... (look at the camera / move the scene a bit
for i in range(NUM_FRAMES):
    ok, frame = cap.read()
    if not ok:
        print(f"[WARN] frame {i} not captured, retrying...")
        continue
    frames_mem.append(frame.copy())
    # Small delay (helps avoid duplicates)
    time.sleep(0.03)

cap.release()
print(f"[INFO] Captured {len(frames_mem)} frames to memory.")
```

```
[INFO] Capturing frames... (look at the camera / move the scene a bit)
[INFO] Captured 30 frames to memory.
```

In [10]:
```python
# Cell 2: save frames to disk
# This lets you re-open specific frames later even if you restart the kernel
# (We clear the folder first so indices are consistent.)

shutil.rmtree(FRAMES_DIR, ignore_errors=True)
FRAMES_DIR.mkdir(parents=True, exist_ok=True)

saved = 0
for i, f in enumerate(frames_mem):
    outp = FRAMES_DIR / f"frame_{i:03d}.png"
    cv2.imwrite(str(outp), f)
    saved += 1

print(f"[INFO] Saved {saved} frames to {FRAMES_DIR.resolve()}")
```

```
[INFO] Saved 30 frames to /Users/rishi/Desktop/Computer_Vision/Projects/Assi
gnmnet_5N6/Assignment5/oflow_session/frames
```

In [11]:
```python
# Cell 3: preview frames with indices to choose a pair
files = sorted(FRAMES_DIR.glob("frame_*.png"))
thumbs = []
for i, fp in enumerate(files):
    img = cv2.imread(str(fp))
    t = cv2.resize(img, (160,160))
    cv2.putText(t, str(i), (5,25), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255,255,2
    thumbs.append(t)

# Make a simple grid
cols = 6
rows = int(math.ceil(len(thumbs)/cols))
grid = []
for r in range(rows):
    row_imgs = []
    for c in range(cols):
        idx = r*cols + c
        if idx < len(thumbs):
            row_imgs.append(thumbs[idx])
        else:
            row_imgs.append(np.zeros_like(thumbs[0]))
    grid.append(np.hstack(row_imgs))
grid = np.vstack(grid) if grid else np.zeros((160,160,3), dtype=np.uint8)

imshow_rgb(grid, title="Thumbnails with indices", size=10)
```



In [12]:
```
# Cell 4: choose two consecutive frames, e.g., f and f+1
f = 10  # <-- change this index after preview
f1_path = FRAMES_DIR / f"frame_{f:03d}.png"
f2_path = FRAMES_DIR / f"frame_{f+1:03d}.png"

frame1 = cv2.imread(str(f1_path))
frame2 = cv2.imread(str(f2_path))

assert frame1 is not None and frame2 is not None, "Selected frames not found
imshow_rgb(frame1, f"Frame {f}")
imshow_rgb(frame2, f"Frame {f+1}")
```

In [17]:
```
# Cell 5: crop a small patch from both frames by giving ROI
# Pick a textured area (corners/edges), e.g., x=200, y=150, w=40, h=40
x, y, w, h = 200, 350, 40, 40   # <-- change these

patch1 = frame1[y:y+h, x:x+w].copy()
patch2 = frame2[y:y+h, x:x+w].copy()

# Save & show
shutil.rmtree(PATCHES_DIR, ignore_errors=True)
```

```
PATCHES_DIR.mkdir(parents=True, exist_ok=True)

cv2.imwrite(str(PATCHES_DIR / f"patch_f{f}.png"), patch1)
cv2.imwrite(str(PATCHES_DIR / f"patch_f{f+1}.png"), patch2)
side = np.hstack([patch1, patch2])
cv2.imwrite(str(PATCHES_DIR / "patch_side_by_side.png"), side)

imshow_rgb(side, title=f"Patch (left: frame {f}, right: frame {f+1})", size=
print(f"[INFO] Saved patches in {PATCHES_DIR.resolve()}")
```


```
[INFO] Saved patches in /Users/rishi/Desktop/Computer_Vision/Projects/Assign
mnet_5N6/Assignment5/oflow_session/patches
```

In [18]:
```
# Cell 6: compute Ix, Iy on frame t; It = I(t+1) - I(t), then build A, b.

# Convert to grayscale float32
g1 = cv2.cvtColor(patch1, cv2.COLOR_BGR2GRAY).astype(np.float32)
g2 = cv2.cvtColor(patch2, cv2.COLOR_BGR2GRAY).astype(np.float32)

# Sobel spatial gradients on frame t (g1)
Ix = cv2.Sobel(g1, cv2.CV_32F, 1, 0, ksize=3) / 8.0  # mild normalization
Iy = cv2.Sobel(g1, cv2.CV_32F, 0, 1, ksize=3) / 8.0

# Temporal gradient
It = g2 - g1

# Build least-squares system A * [u v]^T = b where b = -It
A = np.column_stack([Ix.ravel(), Iy.ravel()])
b = -It.ravel()

print("Patch size:", g1.shape)
print("A shape:", A.shape, "b shape:", b.shape)
```

```
Patch size: (40, 40)
A shape: (1600, 2) b shape: (1600,)
```

In [19]:
```
# Cell 7: solve (A^T A) [u v]^T = A^T b  via np.linalg.lstsq

# Optional: exclude near-zero gradient pixels to reduce noise
mag = np.hypot(A[:,0], A[:,1])
mask = mag > np.percentile(mag, 20)  # drop the flattest 20% pixels
A_use = A[mask]
b_use = b[mask]
```

```
uv, residuals, rank, svals = np.linalg.lstsq(A_use, b_use, rcond=None)
u_est, v_est = uv

ATA = A_use.T @ A_use
ATb = A_use.T @ b_use
cond_num = np.linalg.cond(ATA)

rss = float(((A_use @ uv) - b_use).T @ ((A_use @ uv) - b_use))

print(f"Estimated motion: u = {u_est:.4f}, v = {v_est:.4f}")
print(f"(A^T A):\n{ATA}")
print(f"A^T b: {ATb}")
print(f"Residual sum of squares: {rss:.4f}")
print(f"Condition number of (A^T A): {cond_num:.2f}  (lower is better; corne
```

```
Estimated motion: u = 2.2281, v = 0.0227
(A^T A):
[[29347.844  -7862.1875]
 [-7862.1875 16916.969 ]]
A^T b: [ 65210.   -17133.25]
Residual sum of squares: 222704.7500
Condition number of (A^T A): 2.53  (lower is better; corners >> edges >> fla
ts)
```

In [ ]:

- $\Delta = S_{xx} S_{yy} - S_{xy}^2$

$S_{xx} = \sum I_x^2 = 29347.844$

$S_{xy} = \sum I_x I_y = -7862.1875$

$S_{yy} = \sum I_y^2 = 16916.969$

$S_{xt} = \sum I_x I_t = -65210$

$S_{yt} = \sum I_y I_t = 17133.25$

$\Delta = 1103155548.49 - 61852973.61 \approx 434662574.88$

- $u = \dfrac{1103155548.49 - 134704823.98}{434662574.88} = \dfrac{968450724.51}{434662574.88}$

$u = \boxed{2.2281}$

- $V = \dfrac{-502823948.213 + 512693246.875}{434662574.88} = \dfrac{9869298.662}{434662574.88}$

$V = \boxed{0.0227}$