UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Triennale in Ingegneria Informatica

La visione artificiale in robotica per l'interpretazione dello spazio

Relatore:

Andrea Munafò

Candidato:

Alessio de Dato

ANNO ACCADEMICO 2024/2025

ABSTRACT

In ambito di visione artificiale per la robotica, la proiezione prospettica è cruciale per interpretare lo spazio a partire da immagini.

Questa tesi analizza il modello pinhole dalla formulazione teorica all'implementazione e verifica sperimentale.

Ho sviluppato il software chiamato: "CoordCode", il quale è una GUI in Python/Tkinter per proiettare insiemi di punti 3D sul piano immagine, controllando i parametri intrinseci (focale, principal point) e visualizzando in parallelo le viste 3D/2D con esportazione numerica.

Le simulazioni su figure sintetiche (cubi, piramidi, icosaedro) mostrano che l'aumento della focale genera un'omotetia radiale centrata nel principal point, mentre la variazione di Cx/Cy produce una traslazione rigida delle proiezioni; emerge inoltre la non-iniettività della proiezione centrale (perdita di profondità), con sovrapposizione dei punti allineati sull'asse ottico.

Questi esiti confermano le previsioni analitiche e offrono una base didattica/operativa per comprendere l'impatto degli intrinseci e per impostare la calibrazione di sistemi di visione robotica.

Limiti e sviluppi futuri includono la modellazione di distorsioni ottiche, rolling shutter e l'estensione ad acquisizioni reali per la validazione metrica.

Indice:

Capitolo 1: Introduzione e obiettivi della tesi

In questa tesi ci proponiamo di analizzare in profondità il modello pinhole di proiezione prospettica, partendo dalla sua formulazione teorica fino alla sua implementazione pratica e all'analisi di come le variazioni dei parametri intrinseci influenzino l'immagine ottenuta. Nel dettaglio:

1. **Definizione e fondamenti teorici:** Illustreremo il modello pinhole, ne ricaveremo le equazioni principali e presenteremo un esempio numerico di proiezione di un punto 3D sul piano immagine.
2. **Simulazione pratica:** Realizzeremo un Python/Matlab per simulare la proiezione di un insieme di punti nello spazio, analizzando passo per passo impostazione dei dati, esecuzione e interpretazione dei risultati.
3. **Sensibilità ai parametri intrinseci:** Studieremo teoricamente come variazioni di lunghezza focale e centro principale modifichino la proiezione e lo verificheremo sperimentalmente attraverso ulteriori simulazioni.

Gli obiettivi principali di questo lavoro sono quindi:

- Comprendere in modo rigoroso il funzionamento del modello pinhole.
- Testare un ambiente di simulazione che permetta di mettere immediatamente in pratica i concetti teorici.
- Valutare l'impatto dei parametri intrinseci, concludendo indicazioni utili per la calibrazione di sistemi di visione robotica.

1.1: Problema della Computer Vision in Robotica

La Computer Vision in robotica si occupa di dotare i sistemi automatizzati della capacità di "vedere" e interpretare l'ambiente circostante a partire da immagini acquisite da sensori ottici. In pratica, un robot deve:

- Rilevare e segmentare oggetti nella scena, per riconoscere pezzi, ostacoli o target di manipolazione.
- Ricostruire la geometria 3D dell'ambiente, utile per la navigazione e la pianificazione di traiettorie.
- Stimare la propria posa (localizzazione e orientamento) rispetto a marcatori o caratteristiche note.
- Calibrare i parametri intrinseci ed estrinseci (posizione e orientamento della camera) per garantire accuratezza metrica delle misure.

Risolvere questi problemi è essenziale per applicazioni che vanno dalla manipolazione di oggetti in linee di produzione all'esplorazione autonoma in ambienti non strutturati fino ad arrivare ad applicazioni in campo medico.

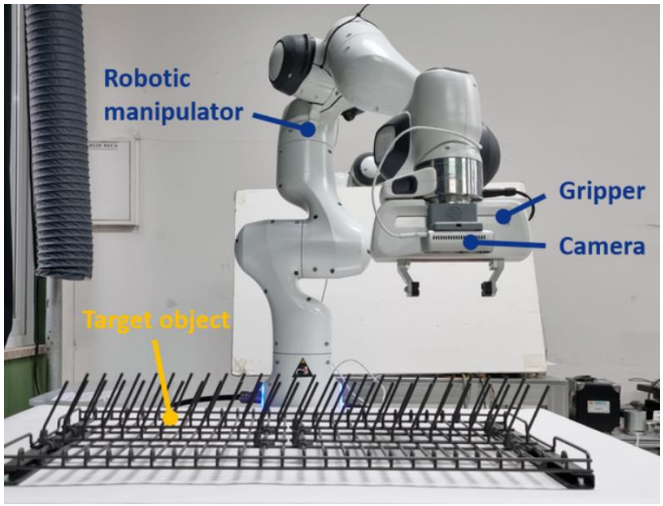
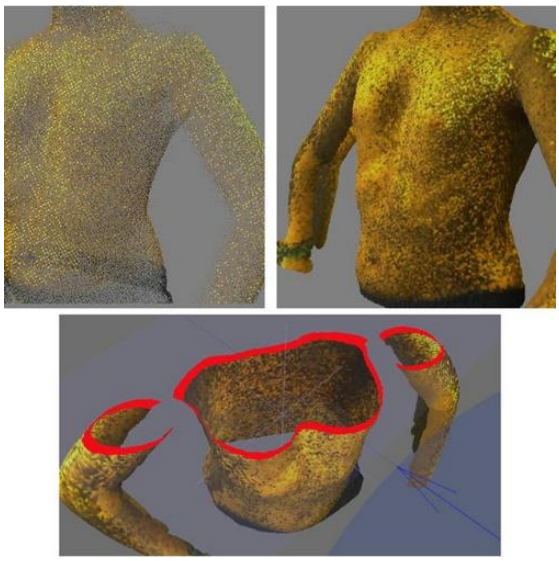


Figura 1: Braccio robotico con telecamera fissata alla pinza, comune nei task di manipolazione guidata dalla visione.

Figura 2: Nuvola di punti 3D ricostruita di un busto umano a partire da più immagini, esempio di Structure-from-Motion (SfM), si basa esplicitamente sul modello pinhole come modello di imaging di riferimento.



Capitolo 2: Il Modello Pinhole

Il modello pinhole idealizza la camera come un dispositivo privo di aberrazioni ottiche: ogni punto dello spazio è “proiettato” linearmente sul piano immagine attraverso un unico centro di proiezione. Questa semplicità matematica è alla base di quasi tutti gli algoritmi di visione artificiale e calibrazione fotografica.

Tale modello ha origini antichissime:

- Prima documentazione (V sec. a.C.)
  Il filosofo cinese Mozi (470–390 a.C.) fu tra i primi a descrivere in termini teorici il fenomeno di un'apertura che proietta su una parete oscura un’immagine capovolta della scena esterna.
- Osservazioni in Grecia (IV sec. a.C.)
  Aristotele (384–322 a.C.) menzionò l’effetto della “camera oscura” durante l’osservazione delle eclissi solari: i raggi di luce che filtravano tra le foglie degli alberi formavano sul terreno immagini capovolte del Sole.
- Prima trattazione scientifica (XI sec. d.C.)
  Il matematico e fisico arabo Alhazen realizzò, tra il 1012 e il 1021, esperimenti sistematici con la “stanza buia”, fornendo la prima descrizione geometrica e quantitativa del dispositivo che chiamiamo oggi “pinhole camera”.
- Modello pinhole in chiave moderna
  Il modello pinhole, pur avendo origini antiche, entra ufficialmente in uso nella visione artificiale a partire dai primi anni ’70, quando viene adottato per la calibrazione fotogrammetrica tramite il metodo del Direct Linear Transformation. Il lavoro fondativo è quello di Abdel-Aziz & Karara (1971), che propongono la DLT per mappare coordinate 3D in coordinate di immagine in close-range photogrammetry.
  Nel 1987 Roger Y. Tsai pubblica “A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses”, in cui utilizza esplicitamente il modello pinhole per calibrare camere montate su robot e per la guida di veicoli automatici, quello di Tsai è considerato il primo metodo sistematico di calibrazione “robot-oriented” basato sul modello pinhole, ed è tuttora alla base di gran parte delle tecniche di visione robotica.

2.1: Descrizione del Modello

Il modello si basa su due elementi geometrici fondamentali: un centro di proiezione, ossia un foro infinitamente piccolo attraverso cui passano tutti i raggi luminosi, e un piano immagine posizionato a distanza focale f da quel centro. In questa configurazione, ogni punto $P = (x_1, x_2, x_3)$ nello spazio si proietta in un punto $Q = (y_1, y_2)$ del piano immagine tramite un'unica retta che collega P al foro.

2.2: Equazioni principali del Modello

Dal punto di vista geometrico, si considera un sistema di riferimento 3D con origine nel foro e assi ortogonali (X_1, X_2, X_3) con rispettivamente: "X_1" coordinata del punto lungo l'asse orizzontale della camera, "X_2" coordinata del punto lungo l'asse verticale della camera e "X_3" è detto asse ottico e punta verso la scena.

Il piano immagine è parallelo ad X_1X_2 e interseca l'asse ottico in corrispondenza del punto "$-f$" (oppure, una formulazione equivalente più pratica, in "$+f$", se si introduce un'immagine virtuale).

La distanza focale "f" rappresenta il parametro intrinseco che scala le coordinate normalizzate.

Matematicamente, si ottiene la proiezione prospettica attraverso la similarità di triangoli, tramite la figura vista lungo l'asse X_2 si ricava:

$$\boldsymbol{y_1} = -\frac{f x_1}{x_3}, \boldsymbol{y_2} = -\frac{f x_2}{x_3}$$

Tramite la quale, eliminando la rotazione di 180° intrinseca al modello reale, diventa:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Questa relazione descrive in modo compatto come la profondità x_3 "comprime" le coordinate orizzontali e verticali in funzione di f.

Nel modello pinhole "elementare" abbiamo appena visto la proiezione di un punto $P = (X, Y, Z)$, coordinate non omogenee, sul piano immagine.

Per poter includere in un'unica espressione sia la **rotazione "**R" che la **traslazione "**t" della camera rispetto a un sistema di riferimento globale, è molto comodo "allargare" le nostre coordinate aggiungendo una componente in più, pari a 1.

Possiamo definire quindi: vettore delle coordinate omogenee di un punto nello spazio 3D (*X*) e vettore delle coordinate omogenee di un punto immagine (*x*):

$$\boldsymbol{X} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \boldsymbol{x} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Nel caso del vettore delle coordinate omogenee di un punto nello spazio 3D:

- $(X \quad Y \quad Z)$ sono le coordinate cartesiane del punto nel sistema di riferimento globale.
- La componente "1" consente di rappresentare traslazioni e proiezioni mediante moltiplicazioni matriciali uniformi, esattamente come avviene passando al sistema di coordinate della camera tramite $[R|t]$.

Nel caso del vettore delle coordinate omogenee di un punto immagine:

- $(u \quad v)$ sono le coordinate del punto sul sensore (solitamente espresse in pixel).
- La componente "1" serve a permettere le trasformazioni proiettive (moltiplicazioni matriciali) in modo uniforme.

Supponiamo ora di avere un punto nel "mondo" (ad esempio nel sistema di riferimento di un banco di lavoro) con coordinate: (X, Y, Z).

Immaginiamo ora che la camera, invece, si trova in una certa posizione e con una certa orientazione rispetto a quel sistema.

Sappiamo che:

- R è una matrice 3×3 che ruota il punto dal sistema mondo al sistema camera.
- t è un vettore 3×1 che trasla il punto per tener conto dello spostamento dell'origine.

Abbiamo dunque una matrice $[R|t]$ di dimensione 3×4:

$$[\boldsymbol{R}|\boldsymbol{t}] = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}$$

La quale moltiplicandola per "X" otteniamo le coordinate del punto espresso nel sistema di riferimento della camera:

$$\boldsymbol{CoordCam} = \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Equivalente a scrivere in maniera esplicita il sistema:

$$\begin{cases} \boldsymbol{X_c} = r_{11}X + r_{12}Y + r_{13}Z + t_x \\ \boldsymbol{Y_c} = r_{21}X + r_{22}Y + r_{23}Z + t_y \\ \boldsymbol{Z_c} = r_{31}X + r_{32}Y + r_{33}Z + t_z \end{cases}$$

A questo punto, individuate $(X_c \quad Y_c \quad Z_c)$ nel sistema della camera, vogliamo proiettarlo sul piano immagine tenendo conto però di alcuni fattori:

- f: la lunghezza focale, che scala le coordinate normalizzate in unità di pixel sul sensore.
- (c_x, c_y): il centro dell'immagine (principal point), ossia lo spostamento del punto ottico rispetto al vertice superiore sinistro del sensore.

Per fare ciò abbiamo bisogno di utilizzare la "**matrice dei parametri intrinseci**" (K) la quale ci permette di tradurre le coordinate metriche sul sensore, nelle coordinate discrete in pixel:

$$\boldsymbol{K} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Dove abbiamo che:

- f_x, f_y: sono le lunghezze focali espresse in pixel lungo gli assi orizzontale e verticale.
- s: è il parametro di skew, che misura l'eventuale non ortogonalità tra riga e colonna del sensore.
- c_x, c_y: è il principal point, ossia le coordinate (in pixel) del punto in cui l'asse ottico incide sul sensore. Spesso questo punto non coincide con il centro geometrico del

sensore, quindi, le coordinate, tengono conto di un'eventuale traslazione dell'origine del sistema di coordinate dell'immagine.

Moltiplicando K per il vettore omogeneo: $\left(\frac{X_c}{Z_c}, \frac{Y_c}{Z_c}, 1\right)$, otteniamo le coordinate in pixel $(u, v, 1)$.

Riassumendo quindi il tutto in un'unica espressione, chiamata: **"equazione di proiezione della camera"**, si ha che:

$$\boldsymbol{x \backsim K[R|t]X}$$

Dove:

- x: indica le coordinate del punto proiettato sul piano immagine, espresse in forma omogenea.
- K: matrice dei parametri intrinseci, che scala e trasla le coordinate normalizzate in coordinate pixel.
- $[R|t]$: matrice estrinseca, che Ruota e Trasla le coordinate dal sistema mondo al sistema camera.
- X: vettore omogeneo $(X, Y, Z, 1)^T$ del punto nel sistema di riferimento globale.Digitare l'equazione qui.
- $\sim$: indica che, a valle della moltiplicazione, dobbiamo dividere per la terza componente (**normalizzazione omogenea**) per tornare a coordinate 2D effettive.

$$(\boldsymbol{u, v}) = \left(\frac{x}{w}, \frac{y}{w}\right) \qquad se\ (\boldsymbol{x, y, w})^{\boldsymbol{T}} = K[R|t]X$$

  - Con w: terza componente del vettore omogeneo risultante dalla proiezione, ed è proprio il fattore di scala che equivale alla profondità del punto rispetto alla camera.

Con questo formalismo, si ha un'unica matrice 3×4 che incapsula posizione, orientazione e proiezione sul sensore: è il cuore di qualsiasi algoritmo di visione artificiale che lavora in contesti reali o robotici.

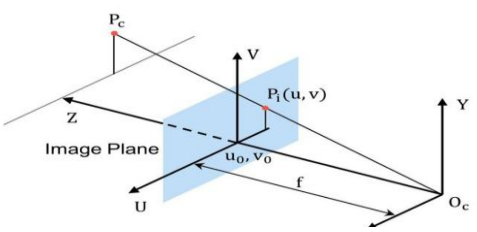


Figura 3: schema vettoriale del modello pinhole, con centro di proiezione O_c piano immagine a distanza focale f coordinate immagine (u_0, v_0) e proiezione sul punto P_c

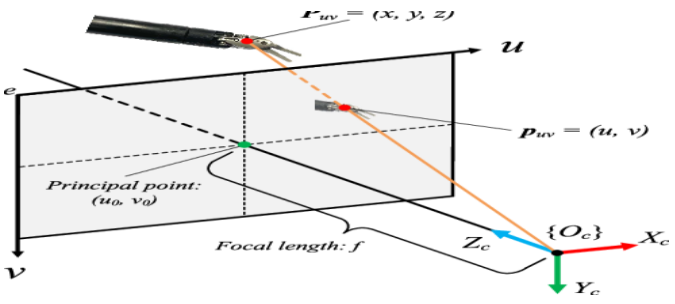


Figura 4: implementazione reale su un braccio robotico con telecamera, che mostra la proiezione del punto $P_{uv} = (x, y, z)$ nel piano immagine in (u, v) attraverso il foro stenopeico.

2.3: Esempio di un punto 3D proiettato nel piano immagine

Per dare un'idea più chiara del concetto espresso fino ad ora in maniera teorica, andiamo a simulare il meccanismo a livello di calcoli reali.

Dati:

- Scegliamo un punto 3D nel sistema mondo:

$$\boldsymbol{P} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix} \rightarrow distanza\ in\ metri$$

- Come parametri estrinseci (scelta semplificata per l'esempio), prendiamo:

$$\boldsymbol{R} = \boldsymbol{I}_3 \ \rightarrow \ I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$e\ prendiamo\ \boldsymbol{t} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

- Come parametri intrinseci invece usiamo:

$$\boldsymbol{f} = 800\ ed\ anche\ \left(\boldsymbol{c_x}, \boldsymbol{c_y}\right) = (320{,}240)\ \rightarrow distanse\ in\ pixel$$

***Nota**: grazie a tale scelta abbiamo che $f_x = f_y = f$ ed abbiamo anche che Skew è $s = 0$.*

Per procedere, bisogna in primis fare una rototraslazione nel sistema camera:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = [R|t] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = I_3 \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix}$$

A questo punto troviamo le coordinate normalizzate:

$$\boldsymbol{x} = \frac{X_c}{Z_c} = \frac{1}{5} = 0{,}2$$

$$\boldsymbol{y} = \frac{Y_c}{Z_c} = \frac{2}{5} = 0{,}4$$

A questo punto possiamo passare sul piano immagine:

$$\boldsymbol{u} = fx + c_x\ , \boldsymbol{v} = fy + c_y$$

Quindi:

$$\boldsymbol{u} = 800 \cdot 0{,}2 + 320 = 480$$
$$\boldsymbol{v} = 800 \cdot 0{,}4 + 240 = 560$$

Abbiamo dunque concluso che, il punto $P = (1{,}2{,}5)$, mappa sul sensore alle coordinate:

$$\boldsymbol{p} = (\boldsymbol{u}, \boldsymbol{v}) = (480{,}560) \rightarrow in\ pixel$$

Vedendo così in maniera pratica tutti i passaggi: dalla definizione del punto 3D e dei parametri camera, fino al calcolo delle coordinate immagine in pixel.

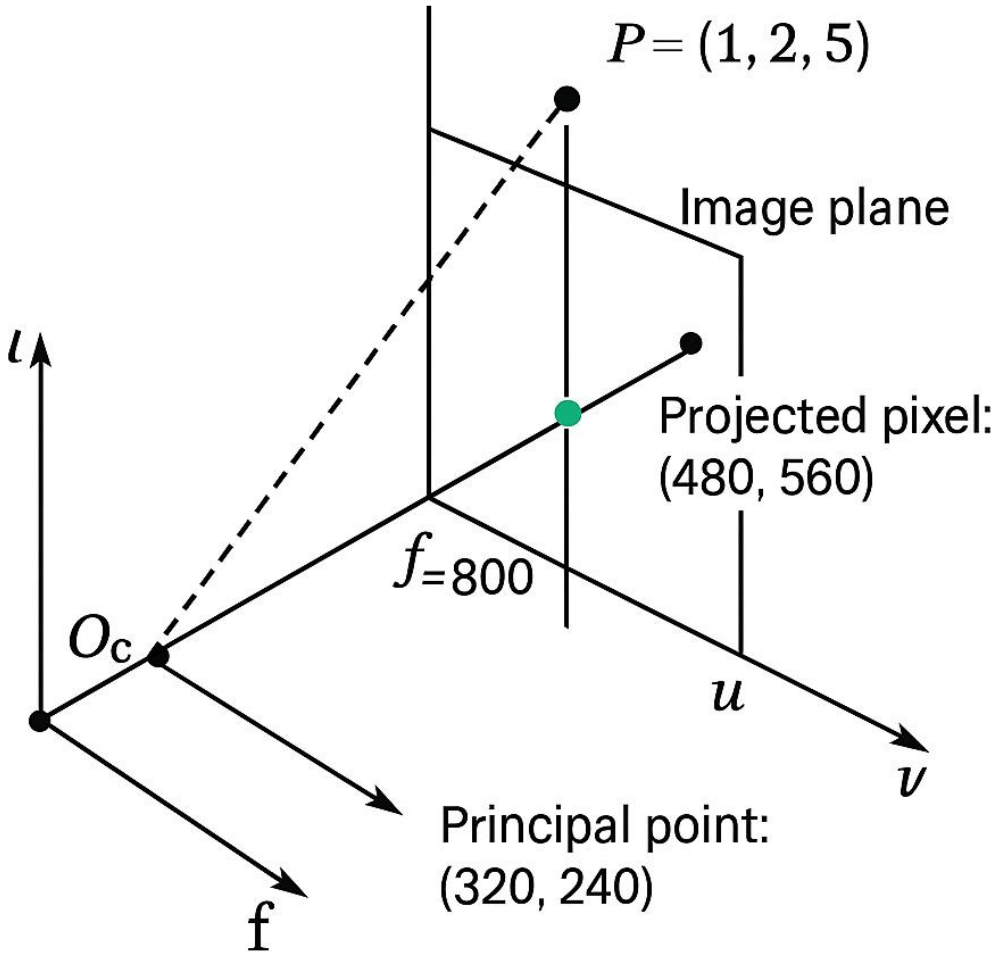


Figura 5: Diagramma vettoriale dell'esempio, il punto 3D → $P = (1{,}2{,}5)$ nel sistema camera $\{O_c\}$ si proietta sul piano immagine, che ha distanza focale $f = 800$, nel pixel di coordinate → $p = (480{,}560)$.

Capitolo 3: Simulazione della proiezione di alcuni punti 3D su un piano immagine

In questo capitolo presento l'ambiente per la simulazione da me sviluppato in Python, denominato **CoordCode**, che consente di proiettare insiemi di punti tridimensionali (X, Y, Z) sul piano immagine (u, v) secondo il **modello pinhole**.

La simulazione implementa la relazione:

$$\boldsymbol{u} = f\frac{X}{Z} + c_x$$

$$\boldsymbol{v} = f\frac{Y}{Z} + c_y$$

ed offre una doppia visualizzazione: la nuvola 3D (in piano tridimensionale) e la corrispondente proiezione 2D (con rappresentazione nel piano immagine).
Si hanno inoltre, strumenti di ispezione (zoom e pan), funzioni di import/export dei dati e possibilità di collegare i punti immessi nel grafico, in modo da dare un'idea più concreta della figura analizzata. Queste funzionalità sono descritte nella testata del sorgente CoordCode.py e all'interno della struttura nella classe principale dell'applicazione.

3.1: Spiegazione della simulazione

La simulazione è organizzata come segue:

- Nella schermata principale, appena eseguito il software, si inseriscono i Parametri Intrinseci, (f, c_x, c_y).
- Dopo di che si passa nella schermata per l'inserimento delle coordinate.
- Si forniscono, uno alla volta, i punti $P = (X, Y, Z)$ con $Z > 0$ in quanto il punto si deve trovare davanti alla camera.
- A questo punto, ogni punto viene proiettato in tempo reale sulla vista 2D e registrato nella tabella insieme alle coordinate calcolate automaticamente (u, v). La dipendenza inversa dalla profondità Z fa sì che, a parità di (X, Y) i punti più lontani (Z maggiore) risultino più vicini al principal point (c_x, c_y) nel piano immagine.
- La GUI permette di osservare direttamente questo effetto, infatti, l'algoritmo di proiezione è implementato nella funzione dedicata della classe applicativa.

3.2: Programma simulativo (*CoordCode*): Architettura Generale

Il programma è una GUI in Python/Tkinter organizzata in una classe principale (ApplicazioneCoordCode) che gestisce:

- Stato dell'applicazione (intrinseci, liste dei punti 3D e delle proiezioni 2D. collegamenti manuali)
- Costruzione delle due pagine dell'interfaccia.
- Disegno delle viste 2D/3D con Matplotlib.
- Importazione ed esportazione su file di testo.

Queste responsabilità sono elencate all'inizio del sorgente e nei commenti di sezione.

3.2.1: Schermata iniziale

All'avvio viene presentata una pagina introduttiva con il titolo CoordCode, una breve descrizione d'uso e due campi sequenziali: la focale f $(in\ pixel)$ e il principal point (c_x, c_y), inseriti come valori separati da virgola.

L'inserimento di f abilita il secondo campo e, alla conferma di (c_x, c_y), l'applicazione passa alla schermata operativa.

La costruzione della pagina, il bind del tasto Invio e la validazione dei valori sono implementati nei metodi che creano la pagina 1 e che eseguono i controlli sugli input.

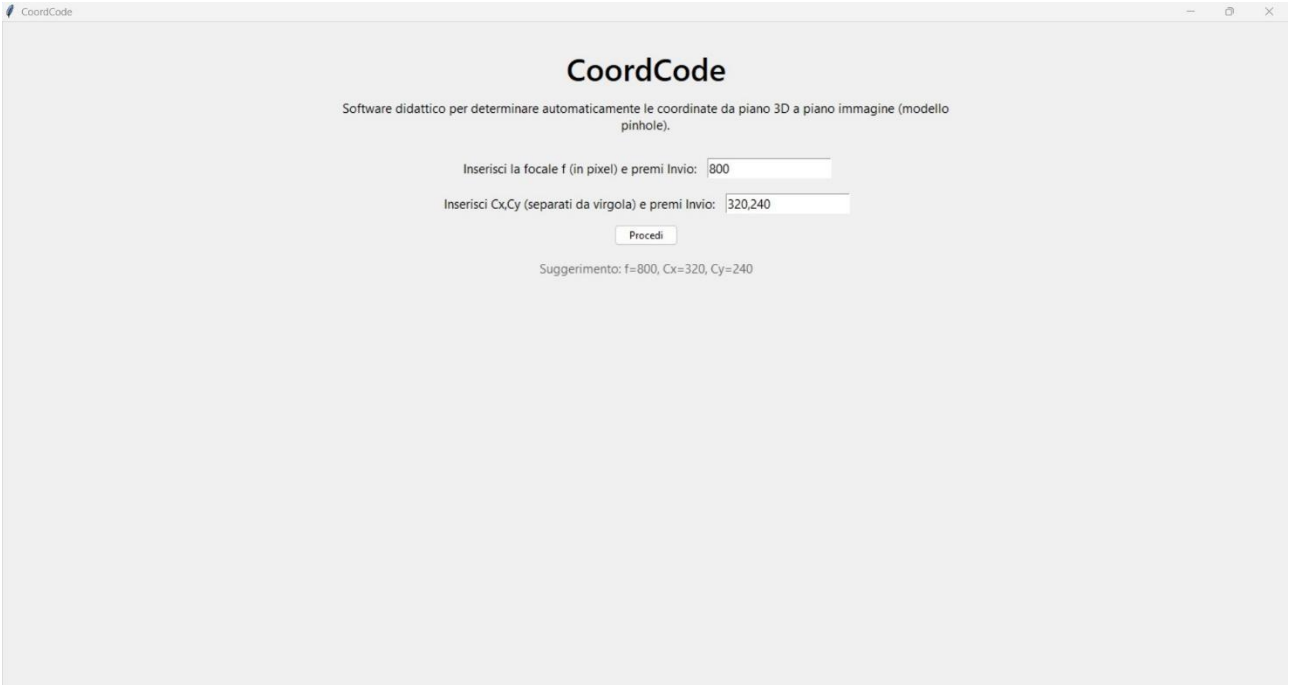


Figura 6: Schermata iniziale, è richiesto l'inserimento della focale e le coordinate di camera.

3.2.2: Schermata operativa (inserimento e visualizzazione)

La seconda pagina divide la finestra in due colonne:

- **Colonna sinistra:** campo di inserimento del punto (X, Y, Z), tabella con le colonne $\#, X, Y, Z, u, v$, pulsanti "Esporta txt…", "Importa txt…" e "Reset". L'aggiornamento della tabella avviene all'aggiunta di ciascun punto; l'intestazione e la creazione del widget Treeview con le colonne sopra indicate sono configurate a codice.

- **Colonna destra:** pannello Visualizzazione con scelta della Vista 2D (piano immagine) o Vista 3D (piano 3D) tramite radio button; opzioni Collega punti in ordine, Chiudi poligono e Mostra spigoli manuali tramite checkbox; box per inserire e gestire collegamenti manuali tra punti mediante una coppia di indici (i, j). La vista 2D mostra anche il principal point come segno "+"; entrambe le viste includono la toolbar di Matplotlib per zoom/pan.

La commutazione fra viste e il relativo ridisegno sono gestiti da metodi dedicati, con autoscale automatico dell'inquadratura; la toolbar viene istanziata per ciascuna canvas.

Qui riportato vi sono alcuni estratti di maggiore importanza del codice implementativo, e un esempio di come risulta il file di testo generato dall'esportazione:

- **Funzione di proiezione (modello Pinhole)**

  È il cuore matematico: calcola le coordinate immagine a partire da un punto 3D.

```
def proietta_punto(self, x, y, z):
    if z <= 0:
        raise ValueError("Z deve essere > 0 "
                         "(il punto deve trovarsi davanti alla camera).")
    u = self.focale * (x / z) + self.cx
    v = self.focale * (y / z) + self.cy
    return u, v
```

- **Inserimento punto 3D e aggiornamento tabella**

  Gestisce input utente, calcola la proiezione e aggiorna GUI e grafico.

```
def aggiungi_punto(self, event=None):
    valori = self.entry_punto.get().split(",")
    x, y, z = map(float, valori)
    u, v = self.proietta_punto(x, y, z)
    self.punti3d.append((x, y, z))
    self.punti2d.append((u, v))
    self.tree.insert("", "end",
        values=(len(self.punti3d), f"{x:.2f}", f"{y:.2f}",
                f"{z:.2f}", f"{u:.2f}", f"{v:.2f}"))
    self.ridisegna_corrente()
```

- **Scelta della vista e opzioni di collegamento**

  Definisce radio button e check box che controllano la visualizzazione.

```
ttk.Radiobutton(opzioni_vista, text="Vista 2D (piano immagine)",
    variable=self.modalita_vista, value="2D",
    command=self.cambia_vista).pack(anchor="w")

ttk.Radiobutton(opzioni_vista, text="Vista 3D (piano 3D)",
    variable=self.modalita_vista, value="3D",
    command=self.cambia_vista).pack(anchor="w")

ttk.Checkbutton(opzioni_linee, text="Collega punti in ordine",
    variable=self.collega_in_ordine_var,
    command=lambda: self.ridisegna_corrente(True)).pack(anchor="w")
```

- **Esportazione su file di testo**

  Scrive i parametri e i punti su file “.txt”, permettendo di salvare la sessione.

```
def esporta_txt(self):
    file = filedialog.asksaveasfilename(defaultextension=".txt")
    if not file: return
    with open(file, "w") as f:
        f.write("[Camera]\n")
        f.write(f"f={self.focale}, cx={self.cx}, cy={self.cy}\n\n")
        f.write("[Punti]\n")
        for i, ((x, y, z), (u, v)) in \
            enumerate(zip(self.punti3d, self.punti2d), start=1):
            f.write(f"{i}: {x},{y},{z} -> {u},{v}\n")
```

  E questo è un esempio di come appare la paginazione nel documento “.txt”:

```
==================== COORDCODE — ESPORTAZIONE DATI ====================
Descrizione: punti 3D, proiezioni (u,v) sul piano immagine e collegamenti definiti dall'utente.
Nota: i punti sono elencati nell'ordine di inserimento; le coordinate u,v sono in pixel.

[Camera]
  f  = 800        # focale in pixel
  cx = 320        # coordinata u del punto principale
  cy = 240        # coordinata v del punto principale

[Punti]
  # indice | X | Y | Z || u | v
   1)  X=200       Y=200       Z=2000       ==>  u=400.0000   v=320.0000
   2)  X=200       Y=-200      Z=2000       ==>  u=400.0000   v=160.0000
   3)  X=-200      Y=200       Z=2000       ==>  u=240.0000   v=320.0000
   4)  X=-200      Y=-200      Z=2000       ==>  u=240.0000   v=160.0000
   5)  X=200       Y=200       Z=2500       ==>  u=384.0000   v=304.0000
   6)  X=200       Y=-200      Z=2500       ==>  u=384.0000   v=176.0000
   7)  X=-200      Y=200       Z=2500       ==>  u=256.0000   v=304.0000
   8)  X=-200      Y=-200      Z=2500       ==>  u=256.0000   v=176.0000
   9)  X=100       Y=100       Z=2150       ==>  u=357.2093   v=277.2093
  10)  X=-100      Y=-100      Z=2150       ==>  u=282.7907   v=202.7907
  11)  X=-100      Y=100       Z=2150       ==>  u=282.7907   v=277.2093
  12)  X=100       Y=-100      Z=2150       ==>  u=357.2093   v=202.7907
  13)  X=-100      Y=-100      Z=2350       ==>  u=285.2174   v=205.2174
  14)  X=100       Y=100       Z=2350       ==>  u=354.7826   v=274.7826
  15)  X=-100      Y=100       Z=2350       ==>  u=285.2174   v=274.7826
  16)  X=100       Y=-100      Z=2350       ==>  u=354.7826   v=205.2174

[SpigoliManuali]
  # elenco di coppie (i, j) che collegano i punti con indici i e j
  (5, 7)
  (7, 8)
  (3, 7)
  (5, 6)
  (6, 8)
  (2, 6)
  (1, 3)
  (1, 2)
  (1, 5)
  (2, 4)
  (3, 4)
  (4, 8)
  (13, 15)
  (13, 16)
  (14, 15)
  (14, 16)
  (10, 11)
  (10, 12)
  (9, 12)
  (9, 11)
  (11, 15)
  (9, 14)
  (12, 16)
  (10, 13)
  (8, 13)
  (7, 15)
  (5, 14)
  (1, 9)
  (2, 12)
  (4, 10)
========================== FINE ESPORTAZIONE ==========================
```

3.3: Esempi pratici

In questa sezione presento alcune configurazioni di punti 3D, generate per verificare visivamente la coerenza tra vista tridimensionale e proiezione sul piano immagine. In tutti i casi, i parametri intrinseci sono:

$$\boldsymbol{f} = 800, \boldsymbol{c_x} = 320, \boldsymbol{c_y} = 240 \ (\textit{unità in pixel})$$

Le figure che seguono sono ottenute con CoordCode, impiegando le opzioni di collegamento automatico e/o gli spigoli manuali quando necessario (si veda il pannello dedicato nella GUI).

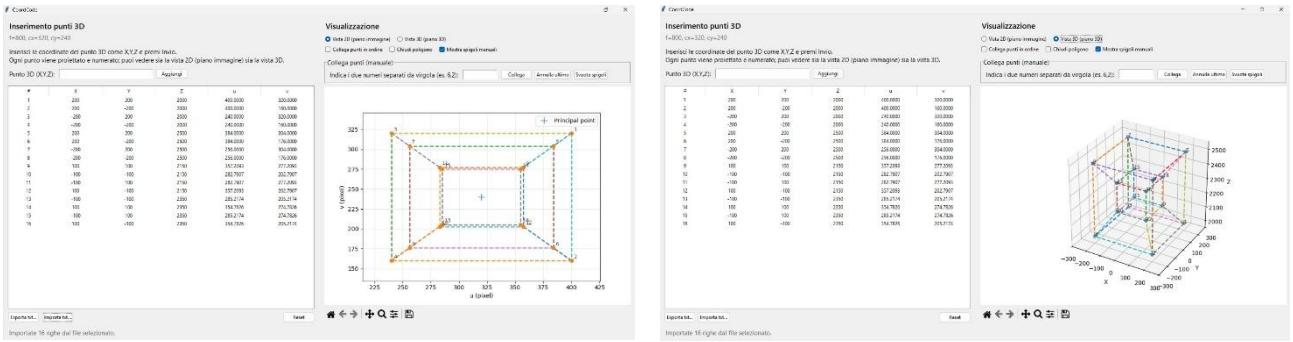


Figura 7.1: *"Doppio cubo": a sinistra visione 2D in piano Immagine, due cubi a profondità distinte proiettano due "cornici" concentriche: il cubo più lontano ha ingombro minore nell'immagine. A destra visione 3D in spazio Reale, la vista tridimensionale conferma la diversa quota Z dei due solidi e la coerenza con la proiezione osservata.*

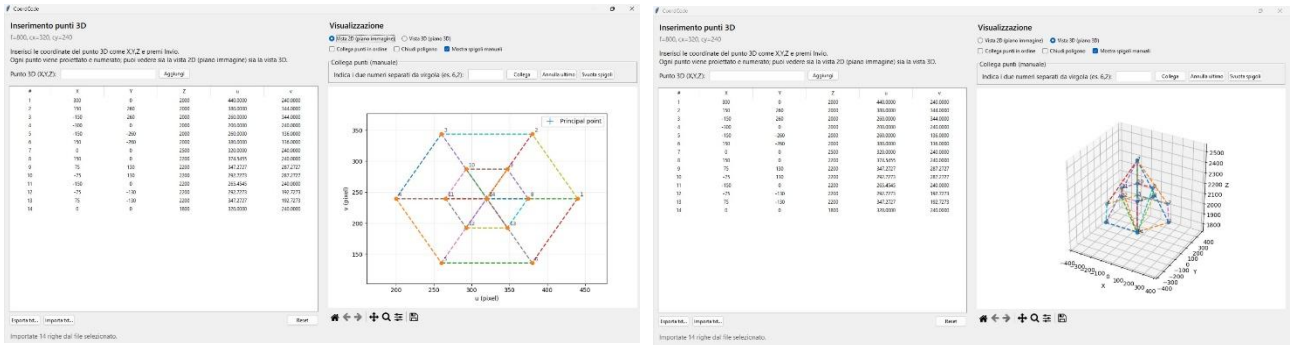


Figura 7.2: *"Doppia piramide": a sinistra visione 2D in piano Immagine, la figura mostra le proiezioni* (u, v) *dei vertici di due piramidi con quote Z differenti: la piramide più distante risulta contratta verso* (c_x, c_y)*, coerentemente con il fattore 1/Z nella proiezione Pinhole. A destra visione 3D in spazio Reale, evidenzia le altezze e la disposizione dei vertici; i collegamenti riproducono quelli usati nella 2D.*

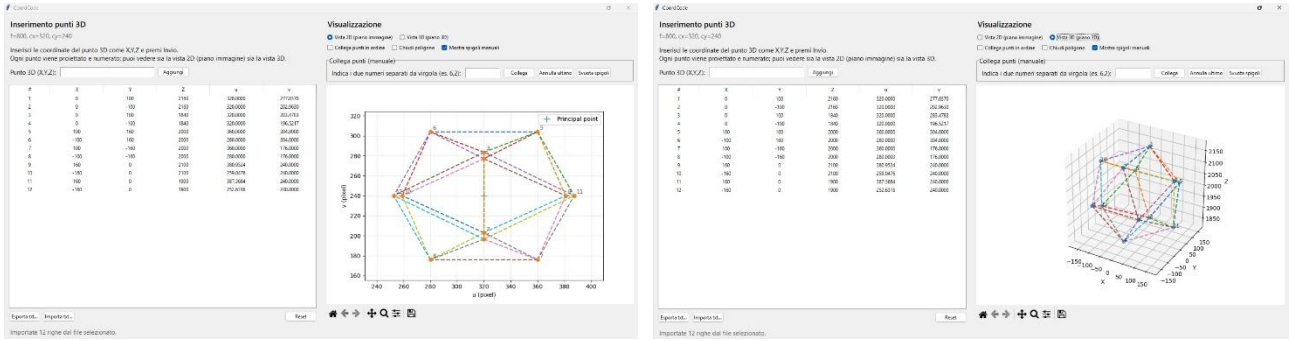


Figura 7.3: *"Icosaedro": a sinistra visione 2D in piano Immagine, la proiezione di un poliedro regolare mette in evidenza la prospettiva: lati non paralleli e variazioni di scala legate alla profondità. A destra visione 3D in spazio Reale, la geometria dei vertici e la connettività risultano pienamente leggibili nella vista 3D e rispecchiano i legami tracciati sul piano immagine.*

3.4: Spiegazione del risultato

Gli esperimenti mostrano che la pipeline implementata da CoordCode restituisce, per ogni punto 3D, una proiezione 2D coerente con il modello pinhole.

La trasformazione:

$$(X, Y, Z) \rightarrow (u, v)$$

dipende linearmente da X/Z e Y/Z e incorpora la traslazione del principal point (c_x, c_y).

L'effetto più evidente è la **dipendenza inversa dalla profondità**: a parità di (X, Y), la proiezione si avvicina a (c_x, c_y) all'aumentare di Z; questo comportamento è verificabile direttamente nella GUI passando dalla vista 3D alla corrispondente 2D.

La realizzazione pratica sfrutta la funzione di proiezione dell'applicazione e i metodi di ridisegno delle viste con autoscale e strumenti integrati di ispezione.

3.4.1: Perdita di informazione e ambiguità della proiezione.

I test condotti con *CoordCode* mettono in evidenza una perdita di informazione intrinseca della proiezione prospettica: la mappa $(X, Y, Z) \rightarrow (u, v)$ non è iniettiva.

In particolare, per ogni $\lambda > 0$ i punti (X, Y, Z) e $(\lambda X, \lambda Y, \lambda Z)$ producono la stessa proiezione, poiché:

$$\boldsymbol{u} = f\frac{X}{Z} + c_x = f\frac{\lambda X}{\lambda Y} + c_x$$

$$\boldsymbol{v} = f\frac{Y}{Z} + c_y = f\frac{\lambda Y}{\lambda Z} + c_y$$

Ne consegue che tutti i punti allineati sulla medesima semiretta uscente dal centro ottico risultano sovrapposti in 2D e dunque indistinguibili sul piano immagine. Inoltre, la riduzione di dimensionalità da $\mathbb{R}^3$ a $\mathbb{R}^2$ comporta la perdita della coordinata di profondità: la tridimensionalità della figura non è più direttamente accessibile, rendendo più complessa la comprensione della struttura a partire da una singola vista.

4: Analisi e test su come cambiano le proiezioni al variare dei parametri intrinseci

In questo capitolo studio in modo teorico-sperimentale l'effetto dei soli parametri intrinseci che l'utente può impostare nel software CoordCode: la Focale (f), il Principal Point (C_x, C_y). Il software, come spiegato ampiamente nel capitolo 3, implementa la proiezione Pinhole, secondo le equazioni:

$$u = f\frac{X}{Z} + C_x$$

$$v = f\frac{Y}{Z} + C_y$$

Entrambe con $(Z > 0)$, come definito nella funzione "*proietta_punto"* del codice.

4.1: Analisi teorica

Sia $P = (X, Y, Z)$ un punto espresso nel sistema Camera.

Scriviamo il vettore proiettato, rispetto al principal point come:

$$\begin{pmatrix} u - C_x \\ v - C_y \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix}$$

Ne discendono le seguenti proprietà, direttamente derivate dall'equazione implementata.

- **Variazione di f (scala radiale)**

  Per un punto fissato:

  $$\frac{\partial u}{\partial f} = \frac{X}{Z}$$

  $$\frac{\partial v}{\partial f} = \frac{Y}{Z}$$

  Un incremento di f dilata la distanza tra $\|u - C_x, v - C_y\|$ in modo proporzionale a $\sqrt{X^2 + Y^2}$ tutto diviso per Z.

  In particolare, per insiemi con Z costante, la proiezione subisce una omotetia centrata in (C_x, C_y) di fattore $\frac{f\ nuovo}{f\ vecchio}$, che è una conseguenza diretta della forma :

  $$(u - C_x, v - C_y) = \left(\frac{f}{Z}\right)(X, Y)$$

- **Ambiguità prospettica (perdita della profondità)**
  Per ogni $\lambda > 0$:

  $$(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})\ e\ (\boldsymbol{\lambda X}, \boldsymbol{\lambda Y}, \boldsymbol{\lambda Z}) \text{ hanno la stessa proiezione su } (\boldsymbol{u}, \boldsymbol{v})$$

  Tutti i punti allineati sulla stessa semiretta che esce dal centro ottico risultano quindi indistinguibili in 2D.
  Questa non iniettività è intrinseca alla proiezione prospettica, come mostrato nel capitolo precedente.

- **Variazione di (C_x, C_y) (traslazione rigida)**
  Si ha:

$$\frac{\partial u}{\partial C_x} = 1$$
$$\frac{\partial v}{\partial C_y} = 1$$

mentre tutte le altre derivate rispetto a (C_x, C_y) sono nulle.

Qualunque variazione di (C_x, C_y) produce solo una traslazione di tutto l'insieme $\{(u, v)\}$ nel piano immagine, senza alterarne scala o forma.

4.2: Simulazione pratica

Per la verifica sperimentale uso lo stesso dataset 3D del Capitolo 3 e vario un solo parametro alla volta (f, C_x, C_y), mantenendo costanti gli altri.

Nel software creato, le routine che gestiscono input, calcolo e ridisegno sono:
- "*aggiungi_punto"* per il parsing e la chiamata della funzione "*proietta_punto"*.

- *ridisegnza_2d* e *ridisegna_3d* per la visualizzazione assieme all'utilizzo della toolbar per l'ispezione

- **Simulazione 1:** Prendendo il caso della figura dell'Icosaedro, una buona simulazione per la Scala Radiale è quella di fissare (C_x, C_y) e confrontare tre valori $f_1 < f < f_2$ con f valore standard (800 pixel).
  La teoria prevede un allontanamento progressivo dei punti da (C_x, C_y) in 2D, con fattore di scala $\approx \frac{f_i}{f}$ a parità di Z.

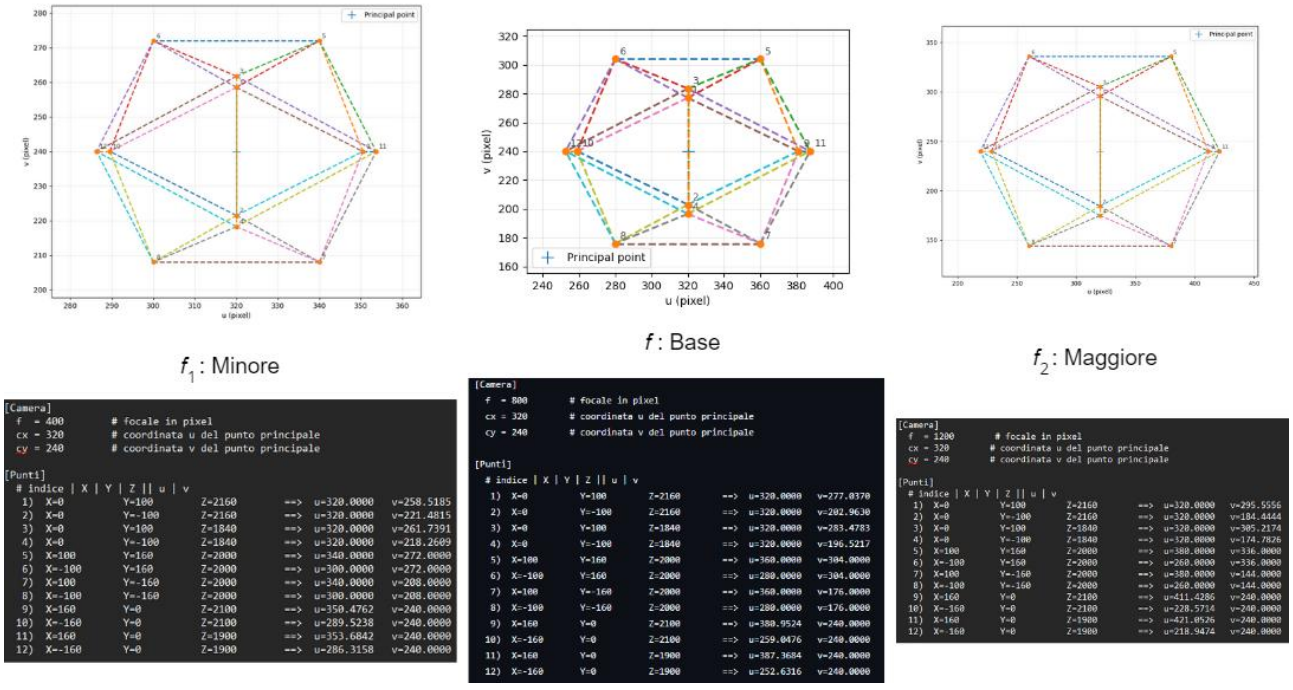


Figura 8: Effetto della variazione per la focale fissato il centro. Incrementando la focale, le proiezioni di dilatano radialmente attorno al pricipal point. A parità di $(X, Y; Z)$, la distanza $\|u - C_x, v - C_y\|$, cresce in proporzione alla focale stessa.

- **Simulazione 2:** Prendendo il caso della figura del Doppio Cubo, si fissa la focale (f) e si trasla (C_x, C_y), prima aumentando C_x e in variando C_y e dopo viceversa, che equivale ad aggiungere un fattore $+\Delta$ peima a u ed in seguito a v.
  La teoria, prevede lo stesso pattern proiettato per entrambi, ma rigidamente traslato di un vettore $(\Delta, 0)$ oppure $(0, \Delta)$.

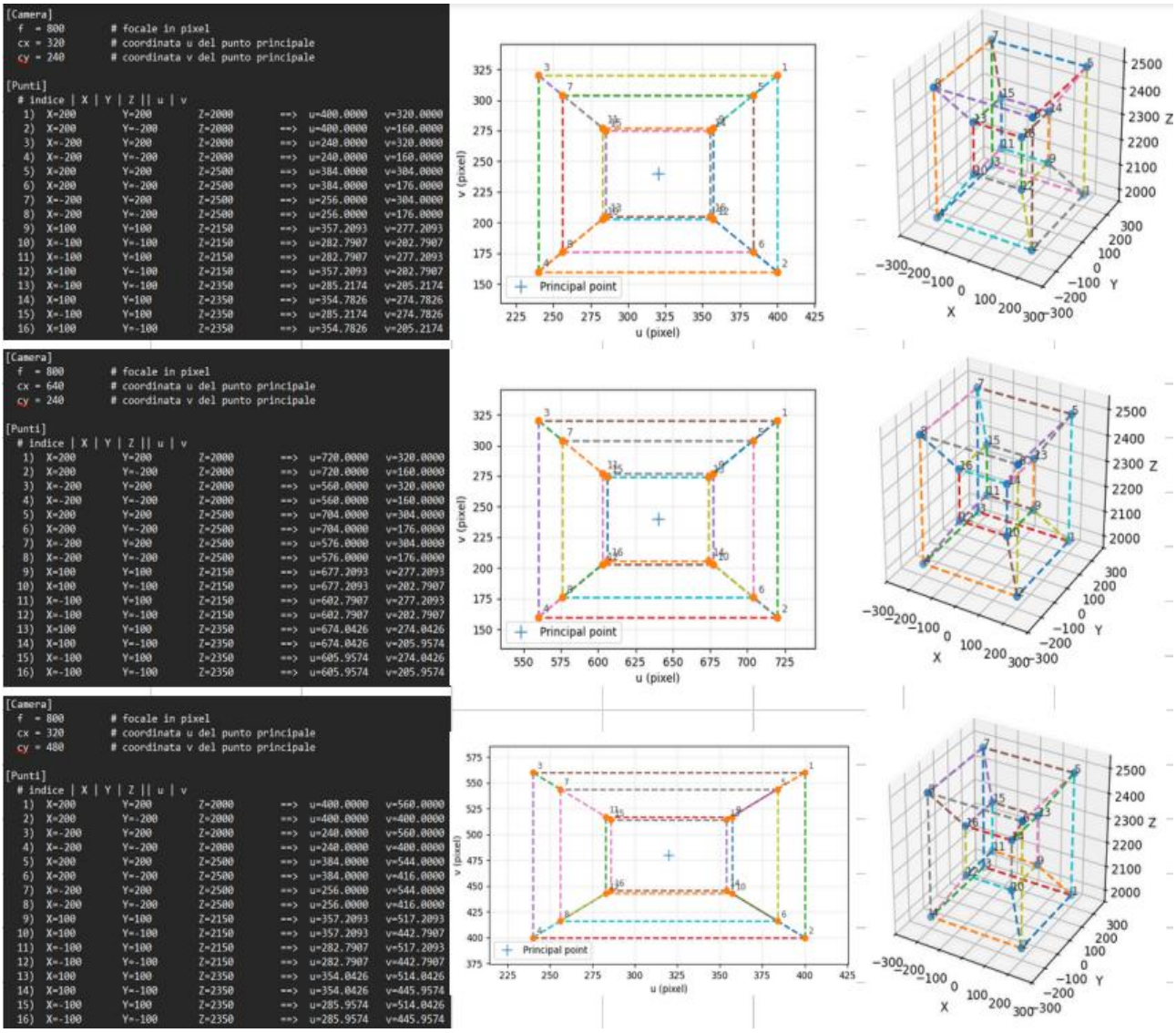


Figura 9: Abbiamo riportato tre casistiche diverse
1) Caso Base: Abbiamo $(C_x = 320, C_y = 240)$.
2) Caso di C_x *: Aumentata rispetto al caso base, portandolo a 640, lasciando invarita* C_y.
3) Caso di C_y *: Aumentata rispetto al caso base, portandolo a 480, lasciando invariata* C_x.

4.3: Osservazioni sui risultati

I risultati sperimentali ottenuti con *CoordCode* coincidono con le previsioni analitiche:

- L'aumento di f realizza una omotetia centrata in (C_x, C_y) delle proiezioni 2D.
- La modifica di $(C_x\ o\ C_y)$, agisce come traslazione rigida dell'intero insieme (u, v).
- Indipendentemente dai valori scelti, la proiezione resta non iniettiva lungo le semirette uscenti dal centro ottico, con sovrapposizione in 2D di punti collineari in 3D.

Queste conclusioni derivano direttamente dall'equazione implementata nel codice e dalla verifica visuale in vista 2D/3D.

5: Conclusioni

Questo lavoro ha sviluppato e verificato, in modo coerente e riproducibile, la pipeline di proiezione prospettica basata sul modello Pinhole e ne ha analizzato gli effetti al variare dei parametri intrinseci effettivamente impostabili in *CoordCode* (la lunghezza focale e il principal point).

Il quadro teorico adottato è la relazione:

$$u = f\frac{X}{Z} + C_x \; con \; (Z > 0)$$

$$v = f\frac{Y}{Z} + C_y \; con \; (Z > 0)$$

che, nella sua essenzialità, isola i contributi di scala radiale (governati dal fuoco), e di traslazione (governati dal principal point) in assenza di Skew e distorsioni.

L'implementazione software ha consentito di mettere alla prova questi elementi su insiemi di punti 3D sintetici, con una duplice resa grafica (vista 3D e proiezione 2D) e con tracciabilità numerica tramite esportazione dei dati.

Dal punto di vista sperimentale, i risultati confermano puntualmente le previsioni del modello. A parità di scena, l'aumento della focale produce un'omotetia centrata in (C_x, C_y): le distanze $\|u - C_x, v - C_y\|$ crescono in proporzione al fattore di variazione di fuoco.

È un comportamento direttamente leggibile dalla forma: $\left(u - C_x, v - C_y\right) = \frac{f}{Z}(X, Y)$ e osservabile in tutte le configurazioni testate: gli stessi vertici, proiettati con fuoco minore o maggiore, si dispongono su "corone" di raggio coerente con il rapporto di scala.

Analogamente, lo spostamento del principal point agisce come traslazione rigida dell'intero insieme (u, v): modificare C_x (rispettivamente C_y) trasla uniformemente le coordinate lungo l'asse u (rispettivamente v), senza introdurre deformazioni né variazioni di scala. Questi due gradi di libertà, scala e offset, sono sufficienti, già da soli, a spiegare la quasi totalità delle differenze osservate nelle proiezioni all'interno del perimetro modellato.

Il progetto mette inoltre in evidenza una caratteristica strutturale della proiezione centrale: la non iniettività della mappa $(X, Y, Z) \rightarrow (u, v)$.

L'identità $(X, Y, Z) \sim (\lambda X, \lambda Y, \lambda Z) \; con \; (\lambda > 0)$ implica che tutti i punti giacenti sulla stessa semiretta uscente dal centro ottico collassano sulla stessa immagine (u, v).

Da qui discende una duplice conseguenza: da un lato, la proiezione è intrinsecamente perdita d'informazione (la profondità non è ricostruibile da una singola vista in assenza di ulteriori vincoli); dall'altro, configurazioni 3D anche molto diverse possono produrre pattern 2D indistinguibili, con una naturale ambiguità interpretativa. Nelle prove, questa ambiguità è stata percepibile come sovrapposizione di punti e come difficoltà di lettura della struttura spaziale quando la sola proiezione 2D è considerata isolatamente. La presenza della vista 3D accoppiata, introdotta nell'applicativo, ha proprio la funzione di mitigare questa perdita percettiva, rendendo esplicito il legame tra geometria nello spazio e resa sul sensore.

Sul piano metodologico, *CoordCode* ha dimostrato di essere un banco di prova utile perché:

- Rende immediata la relazione tra parametri intrinseci e immagine.
- Offre misurabilità tramite tabella numerica (X, Y, Z, u, v) ed esportazione testuale.
- Consente una verifica "visuale e numerica" delle ipotesi teoriche.

La scelta di limitare l'interazione a f e (C_x, C_y) ha volontariamente mantenuto il fuoco sull'essenziale, evitando interferenze con altri fenomeni (Skew, pixel non quadrati, distorsioni). In questo perimetro, la coerenza fra teoria, codice ed evidenza grafica è risultata chiara.

Rimangono, tuttavia, limiti e opportunità. Il modello Pinhole è ideale: non contempla distorsioni ottiche (radiali e tangenziali), Rolling Shutter, rumore di misura, quantizzazione o disallineamenti meccanici; non gestisce inoltre le componenti anisotrope della focale (f_x, f_y) né lo Skew (s).

Questi aspetti, benché trascurati per semplicità, sono spesso determinanti in impieghi reali, in particolare in ambito robotico, dove la precisione metrica e la ripetibilità sono requisiti essenziali. L'osservazione sperimentale della sola coppia (f, C_x, C_y) è quindi da leggersi come primo gradino di una catena di raffinamenti che tipicamente include calibrazione completa degli intrinseci, modellazione delle distorsioni e validazione su acquisizioni reali.

In conclusione, la tesi ha consolidato il ponte tra formulazione matematica e osservazione sperimentale: il modello pinhole, pur nella sua semplicità, spiega in modo trasparente gli effetti principali degli intrinseci; l'applicativo sviluppato ne ha offerto una dimostrazione operativa, traducendo equazioni in comportamenti misurabili e ripetibili. La chiarezza con cui si evidenziano scala radiale, traslazione e perdita di profondità rende questo strumento idoneo sia per la didattica sia come base di partenza per prototipi più completi.

6: Bibliografia:

- "Introductory Techniques for 3-D CV" – Trucco,Verri
- "Multiple View Geometry in Computer Vision" – Richard Hartley, Andrew Zisserman
- "Computer Vision: Algorithms and Applications" – Richard Szeliski
- "Probabilistic Robotics" – Sebastian Thrun, Wolfram Burgard, Dieter Fox
- "Sensor Systems: Fundamentals and Applications" – Clarence W. de Silva
- "Springer Handbook of Robotics" – Siciliano, Khatib
- "Robotica. Modellistica, pianificazione e controllo" – Bruno Siciliano

7: Sitografia:

- https://it.wikipedia.org/wiki/Stenoscopia
- https://it.wikipedia.org/wiki/Sistema_di_visione_artificiale
- https://github.com/ (utilizzati molteplici Repository)
- https://stackoverflow.com/questions (utilizzati molteplici codici)
- https://www.python.org/ (utilizzati molteplici codici)
- https://www.youtube.com/ (utilizzati molteplici canali e video)
- https://robotacademy.net.au/lesson/pinholes-and-lenses/
- https://www.researchgate.net/figure/A-pinhole-system-utilized-for-modeling-a-camera-of-the-robot-binocular-vision-system-It_fig3_331103524
- https://pubmed.ncbi.nlm.nih.gov/38748782/
- https://chatgpt.com/ (in supporto alla programmazione e per dubbi su vari argomenti)

Per visionare l'intero codice Python di "CoordCode", vedere i file .txt degli esempi citati nella tesi e per trovare tutto il materiale da me utilizzato per redigerla, potete visionare la Repository inerente al seguente link:

https://github.com/Deda-404/Tesi-Triennale-IngInf-UniPi

Oppure scannerizzando il seguente QR-Code: