Detecting a Ball (Sphere) Using Camera Geometry

Using Known Camera Height, Tilt, FOV, and Ball Diameter

Introduction

This document outlines a method for detecting a spherical ball in an image captured by a tilted camera at a known height. We assume:

- The ball has a known real-world diameter D_{ball} (e.g., 70 mm).
- The camera is at height H above the ground (e.g., 85 mm), tilted downward by an angle θ.
- The camera has a horizontal and vertical field of view (FOV) of ϕ_h and ϕ_v, respectively (in degrees).
- The image resolution is $W \times H_{\text{img}}$, and the camera is modeled as a pinhole with effective focal lengths f_x, f_y.

Below, we derive the approximate distance from the camera to the ball using circle detection (`cv2.HoughCircles`) and account for partial ground occlusion where the ball contacts the floor. The associated Python code demonstrates how these formulas are implemented in real time.

1. Circle Detection

We first detect the ball's 2D projection in the image using methods such as:

$$\texttt{cv2.HoughCircles}(...) \quad \rightarrow \quad (u_{\text{ball}},\, v_{\text{ball}},\, r_{\text{pix}}),$$

where $(u_{\text{ball}},\, v_{\text{ball}})$ is the circle's center in pixel coordinates, and r_{pix} is the detected circle's pixel radius.

2. Initial Distance Estimate

Under a simpler pinhole model (no occlusion correction), if an object of physical diameter D_{obj} appears with diameter d_{pix} in pixels, we can approximate the distance:

$$D_{\text{est}} \;=\; \frac{f_{\text{avg}} \times D_{\text{obj}}}{d_{\text{pix}}},$$

where

- $d_{\text{pix}} = 2\, r_{\text{pix}}$,
- f_{avg} is an average focal length in pixels, often computed as

$$f_x \;=\; \frac{W/2}{\tan\!\left(\frac{\phi_h}{2}\right)}, \quad f_y \;=\; \frac{H_{\text{img}}/2}{\tan\!\left(\frac{\phi_v}{2}\right)}, \quad f_{\text{avg}} \;=\; \frac{f_x + f_y}{2}.$$

For a sphere of diameter D_{ball}, d_{pix} is the measured diameter in the image.

3. Ground Occlusion Correction

If the ball rests on the ground, part of it is hidden by the contact point with the floor, making the measured circle smaller than the true circle. The Python code uses a function

$$\texttt{project\_y}(Y_{\text{world}},\, Z_{\text{world}},\, f_y,\, c_y)$$

to compute the vertical image coordinate of a 3D point $(0,\, Y_{\text{world}},\, Z_{\text{world}})$ in a camera coordinate system tilted by θ.

3.1 Spherical Radius vs. Contact Point

Let $R_{\text{ball}} = \frac{D_{\text{ball}}}{2}$ be the ball's real radius.

- The *center* of the ball is at $Y_{\text{world}} = R_{\text{ball}}$ above the ground.
- The *contact* with the ground occurs at $Y_{\text{world}} = 0$.

We initially estimate $Z_{\text{world}} = D_{\text{est}}$ and compute where the camera sees $Y_{\text{world}} = R_{\text{ball}}$ and $Y_{\text{world}} = 0$ in the image:

$$v_{\text{center\_est}} \;=\; \texttt{project\_y}\big(R_{\text{ball}},\, D_{\text{est}},\, f_y,\, c_y\big),$$
$$v_{\text{contact\_est}} \;=\; \texttt{project\_y}\big(0,\, D_{\text{est}},\, f_y,\, c_y\big).$$

The difference

$$\Delta \;=\; v_{\text{contact\_est}} \;-\; v_{\text{center\_est}}$$

represents the vertical offset (in pixels) between the ball's center and the ground contact point.

3.2 Corrected Apparent Diameter

In a perfect side view, if no occlusion occurred, the ball's full radius in the image could be approximated by

$$r_{\text{full}} = \sqrt{\Big(\frac{d_{\text{pix}}}{2}\Big)^2 + \Delta^2}.$$

Thus, the *full* (unoccluded) diameter is $d_{\text{full}} = 2\, r_{\text{full}}$. The code then refines the distance:

$$D_{\text{corr}} \;=\; 2 \;\times\; \frac{f_{\text{avg}} \;\times\; D_{\text{ball}}}{d_{\text{full}}},$$

introducing a factor of 2 to compensate for the partial occlusion. This new D_{corr} is used for a more accurate distance estimate.

4. Final 3D Position

Once D_{corr} is computed, we treat the ball's center as lying at

$$Z_{\text{world}} = D_{\text{corr}} \cos(\alpha_{\text{eff}}), \quad X_{\text{world}} = D_{\text{corr}} \sin(\alpha_{\text{eff}}),$$

where

- $\alpha = \arctan\big(\frac{u_{\text{ball}} - c_x}{f_x}\big)$ is the horizontal angle,

- $\beta = \arctan\left(\frac{v_{\text{center\_corr}} - c_y}{f_y}\right)$ is the additional pitch from the camera center to the ball center,
- $\alpha_{\text{eff}} = \theta + \beta$ is the total pitch angle from the horizontal reference.

Hence, in a 2D plane where the camera is the origin, X_{world} is the lateral offset and Z_{world} is the forward distance (in mm). The ball's vertical position relative to the ground can also be approximated by

$$\text{VerticalEst} = D_{\text{corr}} \times \sin(\alpha_{\text{eff}}).$$

5. Summary of Key Steps in the Code

1. **Capture frame**, detect circle $(u_{\text{ball}}, v_{\text{ball}}, r_{\text{pix}})$ via `cv2.HoughCircles`.

2. **Initial distance**:
$$D_{\text{est}} = \frac{f_{\text{avg}} \times D_{\text{ball}}}{(2\, r_{\text{pix}})}.$$

3. **Project ball center** and **contact** to the image using `project_y`.

4. **Compute occlusion offset** Δ and refine diameter d_{full}.

5. **Refine distance** D_{corr} and **re-evaluate** the ball center's pitch angle β.

6. **Compute 2D ground coordinates** $(X_{\text{world}}, Z_{\text{world}})$ and optional vertical position.

6. Caveats and Practical Considerations

- **Lens Distortion:** Wide-angle lenses may cause significant distortion that requires calibration for more accurate geometry.
- **Noise and Circle Detection:** `cv2.HoughCircles` parameters (`dp`, `param1`, `param2`, `minRadius`, etc.) must be tuned for the environment.
- **Partial Occlusion Approximations:** The approach here models the ball-floor contact as a simple 2D projection. Very large tilt angles or non-flat surfaces can introduce errors.
- **Real-Time Constraints:** Computing these corrections each frame is still efficient enough for many robotics tasks, but frame rates vary based on resolution and Pi performance.

Conclusion

By treating a detected circle in the image as part of a *sphere* resting on a flat plane, we can achieve more accurate distance estimates than naive pinhole formulas would give. The core ideas:

1. Use the known ball diameter and measured circle size to estimate distance.
2. Account for camera tilt and partial occlusion with a simple trigonometric correction.

3. Derive final (X, Z) coordinates in millimeters from the camera origin.

This pipeline can guide a mobile robot to locate, track, or interact with spherical objects using only a single downward-tilted camera.