Projectile Motion and Target Detection

Gia Dvalishvili

Kutaisi International University

Introduction to Numerical Programming

Problem Statement

- Calculate initial conditions (velocity v_0 and angle θ) to hit fixed targets.
- Detect targets in an input image using OpenCV.
- Use the shooting method to solve for v_0 and θ.
- Visualize the results with animations.

Circle (Ball) Detection

Code Snippet:

```
def detect_targets(image):
    """Detect circular targets in the image using OpenCV's HoughCircles."""
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) if len(image.shape) == 3 else image
    blurred = cv2.GaussianBlur(gray, (9, 9), 2)
    circles = cv2.HoughCircles(
        blurred, cv2.HOUGH_GRADIENT, dp=1, minDist=50,
        param1=50, param2=30, minRadius=20, maxRadius=100
    )
    return [(c[0], c[1]) for c in np.round(circles[0, :]).astype(int)] if circles is not None else []
```

Shooting Method for Trajectory Calculation (Part 1)

Code Snippet:

```python
def shooting_method_2d(target, origin=(0.0, 0.0), g=9.81,
                       v0_range=(1.0, 100.0), theta_range_right=(5, 80),
                       theta_range_left=(100, 175), steps_v0=200,
                       steps_theta=200):
    """Find initial velocity and angle to hit the target."""
    x_t, y_t = target
    delta_x = x_t - origin[0]
    delta_y = y_t - origin[1]
    theta_min, theta_max = theta_range_right if delta_x >= 0 else theta_range_left
```

Shooting Method for Trajectory Calculation (Part 2)

Code Snippet:

```
    best_v0, best_theta_deg, min_dist = None, None, float('inf')
    for v0 in np.linspace(*v0_range, steps_v0):
        for theta_deg in np.linspace(theta_min, theta_max, steps_theta):
            theta = math.radians(theta_deg)
            cos_theta, sin_theta = math.cos(theta), math.sin(theta)
            if abs(cos_theta) < 1e-6:
                continue
            t_impact = delta_x / (v0 * cos_theta)
            if t_impact <= 0:
                continue
            y_pred = origin[1] + v0 * sin_theta * t_impact - 0.5 * g * t_impact**2
            if abs(y_pred - y_t) < min_dist:
                min_dist, best_v0, best_theta_deg = abs(y_pred - y_t), v0, theta_deg
    return (best_v0, best_theta_deg) if min_dist <= 0.5 else (None, None)
```

Animation Visualization

Code Snippet:

```python
class Animator:
    def __init__(self, fig, ax, trajectories, origin, interval=30):
        """Initialize the Animator class."""
        self.fig, self.ax, self.trajectories = fig, ax, trajectories
        self.origin, self.interval = origin, interval
        self.projectile_marker, = ax.plot([], [], 'bo', markersize=6)
        self.trajectory_line, = ax.plot([], [], 'g-', linewidth=2)
        self.gen = self.frame_generator()
        self.anim = FuncAnimation(
            fig, self.update, frames=self.gen, init_func=self.init,
            interval=self.interval, blit=True, repeat=False
        )
    def frame_generator(self):
        for traj in self.trajectories:
            for x, y in zip(traj[0], traj[1]):
                yield x, y
```

Results and Challenges

Results:

- Successful detection and targeting of objects in clean, noise-free images.
- Accurate trajectory calculations for moderate targets.

Challenges:

- Difficulty detecting small or indistinct targets in noisy images.
- Extreme trajectories (e.g., high targets) can be computationally expensive.

Future Improvements

- Optimize the shooting method for faster convergence.
- Enhance target detection for complex and noisy environments.
- Incorporate realistic factors like air resistance into trajectory calculations.