Planning Tracking Motions for an Intelligent Virtual Camera

by

Tsai-Yen Li and Tzong-Hann Yu

{li,s8410}@cs.nccu.edu.tw

Computer Science Department

National Chengchi University

Taipei, Taiwan, R.O.C.

Motivations

- An auto-navigation system for virtual environments:
  - specifying locations of interests by clicking on a 2D layout map

- The problems:
  - Tour path planning
  - Camera motion planning
  - Humanoid simulation

- Given known target path, to generate intelligent camera tracking motions to
  - avoid collisions with obstacles
  - always keep the target in sight
Related Work

- **Graphics:**
  - Visibility computation for geometry culling
    - Teller 91], [Teller 97], etc.
  - Tracking fixed point in image space
    - Gleich 92], etc.
  - Film directing with Cinematographic idioms
    - He 96], etc.

- **Robotics:**
  - Sensor placement problem
    - Briggs 96], etc.
  - Pursuit-evasion problem
    - Guibas 97], [Gonzalez-Banos 97], etc.
  - Intelligent Observer problem
    - Becker 95], [Lavalle 96], etc.

Problem Formulation: View Model

Target configuration: 
\( q' = (x', y', \theta') \)

Viewpoint configuration:
\( q'' = (x'', y'', \theta'') \)

Composite space:
\( (x', y', \theta', x'', y'', \theta'') \)

Configuration-Time space:
\( (t, x', y', \theta') \)
Problem Formulation:
Planning Space Parameterization

Camera

Target

View Model Definitions:
View Distance ($l$) and Tracking Direction ($\phi$)

Page 3
**View Model Definitions: View Angle ($\phi$)**

- **Target**
- **Camera**

---

**Search Space for the Planning Problem**

- **Equivalent space:** $CT (t, x^v, y^v, \theta^v) \Rightarrow CT' (t, \phi, l, \varphi)$
- **Simplification:** fixing view angle ($\varphi$) $\Rightarrow CT'' (t, \phi, l)$
- **Relaxing view angle ($\varphi$) after a feasible path is found.**
Best-First Planning Algorithm

procedure BFP {
mark all the configurations in CT as unvisited;
INSERT(q_i, OPEN); mark q_i as visited;
SUCCESS ← false;
while (!EMPTY(OPEN) and !SUCCESS) {
    q ← FIRST(OPEN);
    for (every q’ ∈ NEIGHBOR(q)) {
        mark q’ visited;
        if (LEGAL(q’)) {
            PARENT(q’) ← q;
            INSERT(q’, OPEN);
        }
        if (GOAL(q’)) SUCCESS ← true;
    }
}
if (SUCCESS)
    return the path by tracking back to q_i
}

Search Criteria for BFP

- **Planning time** (t): highest priority
  - return the first found path
- **Tracking Direction** (ϕ): subjective criterion
  - a range centered behind the target
- **View Distance** (l): subjective criterion
  - percentage of the human figure in the rendered image
- **Overall Movement** (d): subjective criterion
  - avoid motion sickness and speed up graphics
- **View Angle** (j): lazy movement in postprocessing
  - avoid frequent rotation/scene changes
Planning Criteria: Cost Functions

\[ f(t, \phi, l, dir) = w_1 * f_1(t) + w_2 * f_2(\phi) + w_3 * f_3(l) + w_4 * f_4(\phi, l, dir) \]

- \( f_1(t) = t_e - t \): cost function for the time difference to the ending slices
- \( f_2(\phi) = |\phi - \phi_0|\): cost function for the tracking direction
- \( f_3(l) = |l - l_0|\): cost function for the view distance
- \( f_4(\phi, l, dir) = \text{dist}(p(\phi, l, 0) - p(\phi, l, dir))\): cost function for the Euclidean distance moved from its parent configuration,

\( w_i \): normalized weights (except for \( w_1 \)) for individual cost functions,
\( t \): current time, \( t_e \): is the ending time
\( \phi \): current tracking direction, \( \phi_0 \): is a neutral tracking direction
\( l \): distance between the viewpoint and the target, \( l_0 \): is a neutral view distance
\( dir \): an integer indicating the direction where the current configuration was created,
\( p \): returns the previous position of the viewpoint for the given approaching direction,
\( \text{dist} \): returns the distance between two positions.

Implementations and Experiments

- **Grid search space**: stack of 2D bitmaps indexed by time (with appropriate parameterization)
- **Collision detection**: a line segment with obstacles (with the help of linear-time C-space construction)
- **Path smoothing**: replacing by paths with smaller overall costs
- **Programming**: written in Java
- **Experimental settings**:
  - planning time measured on a Pentium II 233MHz PC
  - workspace: 128x128 grid, **rotational increment**: 3 degrees
  - **parameter ranges**: \( \phi \): ±110, \( l \): (10, 60), \( \phi \): ±15
Experimental Examples: Target Path

Example 1: 257 steps
Example 2: 515 steps

Generated by a Holonomic Path Planner

Experimental Results: An Example

Parameter Set #1
\[ w_1(\phi) = 100.0 \]
\[ w_2 = 0.0 \]
\[ w_3 = 0.0 \]
planning time = \(0.56\) sec

Parameter Set #2
\[ w_1 = 0.0 \]
\[ w_2(l) = 100.0 \]
\[ w_3 = 0.0 \]
planning time = \(0.39\) sec

Parameter Set #3
\[ w_1 = 0.0 \]
\[ w_2 = 0.0 \]
\[ w_3(dist) = 100.0 \]
planning time = \(2.59\) sec

Camera Tracking Motions
Experimental Results: Another Example

Camera Tracking Motions

Parameter Set #1
\[ w_1 \ (\phi) = 100.0 \]
\[ w_2 = 0.0 \]
\[ w_3 = 0.0 \]
planning time = 1.86 sec

Parameter Set #2
\[ w_1 = 0.0 \]
\[ w_2 (l) = 100.0 \]
\[ w_3 = 0.0 \]
planning time = 2.07 sec

Parameter Set #3
\[ w_1 = 0.0 \]
\[ w_2 = 0.0 \]
\[ w_3 (dist) = 100.0 \]
planning time = 5.00 sec

Prefer Good Tracking Direction (\(\phi\))
Prefer Good View Distance (\(l\))
## Conclusion and Future Work

- **Proposing a planning approach for tracking moving target with an intelligent virtual camera**
  - finding a feasible path **quickly** for interactive applications
  - finding a **good** path according to user-specified criteria

- **Future extensions:**
  - integration in an **auto-navigation system** for virtual factories
  - incorporating Cinematographic idioms for **automatic preference selection**
  - **blending costs** between keyframes of different parameter sets
  - developing more **efficient collision detection** routines
  - handling **3D environments** with full camera motions