Motion Generation for Flexible and Rigid graphs

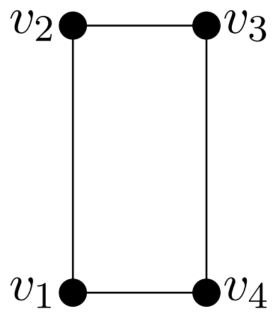
Sartaj Islam - 200050128

Contents

1 Problem

Given a graph (G, p) where p is a map of the vertices of G into 2-space having only one non-trivial degree of freedom. Generate and visualize the non-trivial motion of the graph.

2 Building Blocks



2.1 Rigidity Matrix

The rigidity matrix $M(G, p)$ of (G, p) is the matrix of coefficients of this system of equations. It is an $|E| \times d|V|$ matrix with rows indexed by E and sequences of d consecutive columns indexed by V. The entries in the row corresponding to an edge $e \in E$ and columns corresponding to a vertex $u \in V$ are given by the vector $p(u)-p(v)$ if $e = uv$ is incident to u and is the zero vector if e is not incident to u.

Example Let (G, p) be the 2-dimensional frameworks shown above, and let $p(v_i) = (x_i, y_i)$ for $1 \leq i \leq 4$. Then the rigidity matrix of (G, p) is the matrix $M(G, p)$ shown below.

$$\begin{pmatrix} x_1-x_2 & y_1-y_2 & x_2-x_1 & y_2-y_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_2-x_3 & y_2-y_3 & x_3-x_2 & y_3-y_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_3-x_4 & y_3-y_4 & x_4-x_3 & y_4-y_3 \\ x_1-x_4 & y_1-y_4 & 0 & 0 & 0 & 0 & x_4-x_1 & y_4-y_1 \end{pmatrix}$$

2.2 Null Vector

We use $Z(G, p)$ to denote the null space of the matrix $M(G, p)$ and refer to the vectors in $Z(G, p)$ as instantaneous motions of the framework (G, p).

The nullspace $Z(G, p)$ is a span of set of vectors z which is the solution of the given equation where M is the rigidity matrix $M(G, p)$.

$$Mz = 0$$

The vector $z = [z_{1x}, z_{1y}, ..., z_{4x}, z_{4y}]$ gives the instantaneous velocities of the vertices of the graph (G, p) such that the edge lengths $e \in E$ doesn't change. The velocity of vertex v_i is given by (z_{ix}, z_{iy}). The vectors in $Z(G, p)$ contains the vectors which describe the trivial motions of the graph like translation and rotation as well as the non-trivial ones, if the graph is flexible. To only have the non-trivial motions, we can fix an edge in the plane which removes both the trivial translation and rotation motion.

Example Fix the edge between v_1 and v_4 in the above graph. Then we want solutions of the form $z = [0, 0, z_{2x}, z_{2y}, z_{3x}, z_{3y}, 0, 0]$ from the equation $Mz = 0$. If we put the given vector in this equation we get $M'[z_{2x}, z_{2y}, z_{3x}, z_{3y}]^T = 0$ where M' is given below

$$\begin{pmatrix} x_2-x_1 & y_2-y_1 & 0 & 0 \\ x_2-x_3 & y_2-y_3 & x_3-x_2 & y_3-y_2 \\ 0 & 0 & x_3-x_4 & y_3-y_4 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

This M' is formed by masking out the columns corresponding to those vertices which were the part of the edge we wanted to fix. Hence, below are the steps given to find the non-trivial motion of the graph.

1. Choose an edge to fix. Let $(v_i, v_j) \in E$ be the edge.

2. Create a mask vector m of size $2|V|$ such that $m_{2i-1}, m_{2i}, m_{2j-1}, m_{2j}$ are set to 0 and rest are 1.
3. Get the null space Z' of M' such that $M' = M(:, m)$.
4. The null space Z of M can be created by unmasking the vectors $z' \in Z'$ using m. Hence, a vector z is in nullspace Z if $z(m) = z'$ and $z(1-m) = 0$ where $z' \in Z'$.

Note the number of solutions of $M'z' = 0$ gives the number of non-trivial degree of freedom of the graph. Zero solution means the given graph is rigid.

2.3 Angular Velocity

Once we get the nullspace Z' which is a span of a single vector z' in case of a graph having only one non-trivial degree of freedom we also have the nullspace Z which is also a span of single vector produced by unmasking z' to z.

To produce a visually stable motion, such that the velocities of the vertices do not arbitrarily change on every step, we need to choose an appropriate vector $z \in Z$. This can be achieved by having a constant angular velocity of any one of the edges adjacent to the fixed edge.

Once we have a stable velocity vector, we need to set the graph in motion, such that the edge length doesn't change. The new positions can be calculated as follows:

$$p'_{ix} = p_{ix} + z_{ix}dt$$
$$p'_{iy} = p_{iy} + z_{iy}dt$$

where p'_i is the new position, p_i is the old position and z_i is the velocity of the vertex v_i. Here, dt is the infinitesimally small time interval. Updating the positions this way will not ensure the length of edges to remain constant as it is impossible to take infinitesimally small steps of $z_{ix}dt$ size. The step size needs to be atleast 1 pixel of display screen to make it visually moving.

Example Consider the edge $(v_1, v_2) \in E$ in the above diagram. Let the initial length of this edge be e_{12} and this same edge was chosen to move with constant angular velocity ω. After first step, the length of edge (v_1, v_2) will change to $e_{12}\sqrt{1 + (\omega dt)^2} > e_{12}$. In the next step, this will increase further and the error will keep on accumulating, which tends to increase the length of (v_1, v_2).

To solve this problem, angular velocity helps again. The angular velocity ω_{ij} of an edge $(v_i, v_j) \in E$ can be calculated as:

$$\omega_{ij} = \frac{z_{jy} - z_{iy}}{p_{jx} - p_{ix}} - \frac{z_{jx} - z_{ix}}{p_{jy} - p_{iy}}$$

and the angle θ_{ij} made with the x-axis is given by:

$$\theta_{ij} = \tan^{-1}\left(\frac{p_{jy} - p_{iy}}{p_{jx} - p_{ix}}\right)$$

To calculate the new position p'_j of vertex v_j, let us assume that, the new position is p'_i, one of it's adjacent vertex v_i is known and e_{ij} be the edge length of (v_i, v_j).

$$p'_{jx} = p'_{ix} + e_{ij}\cos(\theta_{ij} + \omega_{ij}dt)$$
$$p'_{jy} = p'_{iy} + e_{ij}\sin(\theta_{ij} + \omega_{ij}dt)$$

After this update, the edge length is still maintained to be equal to e_{12}.

2.4 Breadth First Search

Before calculating the new position p'_j of vertex v_j we assumed to know the new position p'_i of an adjacent vertex v_i. This assumption can be achieved through BFS algorithm. Since there is a fixed edge, the position of the vertices forming this fixed edge is always fixed. In the initial queue of BFS we can either have any of these vertices or both of them and proceed on the loop until the queue is empty.

Example In the above graph, let the fixed edge be (v_1, v_4). Hence the vertices v_1 and v_4 are fixed and their new positions are always equal to their initial positions. The state of the queue at each iteration of BFS is shown below

$$\{v_1, v_4\} \rightarrow \{v_4, v_2\} \rightarrow \{v_2, v_3\} \rightarrow \{v_3\} \rightarrow \{\}$$

This naive BFS has a problem again as it doesn't ensures the constant length of all the edges but only the ones which are in the BFS tree. In the example shown above, the edges in the BFS tree are $\{(v_1, v_4), (v_1, v_2), (v_4, v_3)\}$ and the edge (v_2, v_3) is left at every step of the graph's motion. This way it doesn't ensure the length of the (v_2, v_3) edge to remain constant. As a result, the length of this edge changes continuously at every step of graph's motion due to accumulation of floating point errors or the errors encountered during the division by zero in the above calculations.

To encounter this problem, we modify the calculation of new position p'_j of vertex v_j. Now instead of considering only one of the vertex adjacent to v_j whose new position is known, we take all the adjacent vertices whose new position is known. Let this set of vertices is denoted by S_j for vertex v_j. Now the new position p'_j is given by:

$$p'_{jx} = \frac{1}{|S_j|} \sum_{v_i \in S_j} p'_{ix} + e_{ij} \cos(\theta_{ij} + \omega_{ij} dt)$$
$$p'_{jy} = \frac{1}{|S_j|} \sum_{v_i \in S_j} p'_{iy} + e_{ij} \sin(\theta_{ij} + \omega_{ij} dt)$$

where the meaning of symbols e_{ij}, θ_{ij} and ω_{ij} is given above. This way, the edge lengths do not change significantly over time and can ensure constant length of all the edges in each step within floating point error limits.

3 Algorithm

3.1 Next Positions

To plot the animation of a flexible graph motion, we need to get the positions of all the vertices p at each timestamp. The following algorithm updates the current positions of the vertices in the graph which runs at every timestamp.

```
Algorithm 1: nextPositions
   Data: $V, E, p, e_f, v_r, s$
 1 $M \leftarrow \mathsf{rigidity}(V, E, p)$
 2 $v_{f_1}, v_{f_2} \leftarrow e_f$
 3 $m \leftarrow \mathsf{ones}(2 * |V|),\ m_{f_1 - 1} \leftarrow 0,\ m_{f_1} \leftarrow 0,\ m_{f_2 - 1} \leftarrow 0,\ m_{f_2} \leftarrow 0$
 4 $M' = M[:, m]$
 5 $z' \leftarrow \mathsf{nullVector}(M')$
 6 $z \leftarrow \mathsf{zeros}(2 * |V|),\ z[m] \leftarrow z'$
 7 $zdt \leftarrow z * s / \mathsf{angularVelocity}(z, p, v_{f_1}, v_r)$
 8 $p' \leftarrow \mathsf{zeros}(|V|, 2)$
 9 for $v_j, S_j \in \mathsf{bfsIterate}(V, E, e_f)$ do
10     for $v_i \in S_j$ do
11         $e_{ij} \leftarrow \mathsf{distance}(p_i, p_j)$
12         $\theta_{ij} \leftarrow \mathsf{tanInverse}(p_{jx} - p_{ix}, p_{jy} - p_{iy})$
13         $\omega_{ij} dt \leftarrow \mathsf{angularVelocity}(zdt, p, v_i, v_j)$
14         $p'_{jx} = p'_{jx} + (p'_{ix} + e_{ij} * \cos(\theta_{ij} + \omega_{ij} dt))$
15         $p'_{jy} = p'_{jy} + (p'_{iy} + e_{ij} * \sin(\theta_{ij} + \omega_{ij} dt))$
16     end
17     $p'_{jx} \leftarrow p'_{jx} / \mathsf{size}(S_j)$
18     $p'_{jy} \leftarrow p'_{jy} / \mathsf{size}(S_j)$
19 end
20 return $p'$
```

The inputs to the above function are V, E, p, e_f, v_r, s which are vertices, edges, current position of vertices, fixed edge, the adjacent vertex which is to be rotated with constant angular velocity and the step size by which this vertex moves at every frame respectively. Below are the description of the functions used in this algorithm:

- ones: vector of ones of given size.
- zeros: vector of zeroes of given size.

- nullVector: unique null vector of the input matrix.
- distance: distance between the input positions.
- tanInverse: inclination of the edge given the relative position of the vertices in the range $[0, 2\pi]$.
- angularVelocity: angular velocity of the edge formed by the two input vertices.
- size: size of the input array or set.

3.2 BFS Iterate

The bfsIterate function returns the traversal of the BFS algorithm along with the set of connected vertices which were already visited at the time of traversing a vertex for all the vertices. The pseudo code for the algorithm is given below:

Algorithm 2: bfsIterate

```
Data: $V, E, e_f$
$v_{f_1}, v_{f_2} \leftarrow e_f$
$adj \leftarrow$ emptySets($|V|$)
for $v_i, v_j \in E$ do
    append($adj_i, v_j$)
    append($adj_j, v_i$)
end
$q \leftarrow \{v_{f_1}, v_{f_2}\}$
$vis \leftarrow$ zeros($|V|$), $vis_{f_1} \leftarrow 1$, $vis_{f_2} \leftarrow 1$
$T \leftarrow \{\}$
while notEmpty($q$) do
    $v_i \leftarrow$ pop($q$)
    for $v_j \in adj_i$ do
        if $vis_j = 0$ then
            $vis_j \leftarrow 1$
            append($q, v_j$)
            $S_j \leftarrow \{\}$
            for $v_k \in adj_j$ do
                if $vis_k = 1$ then
                    append($S_j, v_k$)
                end
            end
            append($T, (v_j, S_j)$)
        end
    end
end
return $T$
```

The inputs to the above function are V, E, e_f which are vertices, edges and the fixed edge respectively. Below are the description of the functions used in this algorithm:

- emptySets: create array of empty sets of given size.
- append: append an element to the input container.
- notEmpty: checks if the input container is empty.
- pop: pops an element from the top of the container.

4 Modifications and Error Handling

4.1 Null Vector

In case of infinitesimally rigid graphs whose flexibility depends highly on the edge lengths, taking a step in the motion of graph would result in having a rigid graph due to minor change of edge length by floating point errors. As a result the nullspace of the masked rigidity matrix M' is empty.

To encounter this problem we can update the null vector function to produce a *near null vector* which returns a vector which when multiplied with the input matrix results in a vector which is very close to zero.

$$M'z' \approx 0$$

The following algorithm shows how to get this vector z':

Algorithm 3: nearNullVector

```
  Data: $M'$
1 $U, S, V \leftarrow \mathsf{svd}(M')$
2 $\lambda_{min} \leftarrow \mathsf{argmin}(S)$
3 $z' \leftarrow V_{\lambda_{min}}$
4 return $z'$
```

The svd function does the singular value decomposition of the matrix and the argmin function returns the index of the minimum value in the diagonal elements of the matrix S.

4.2 Divide by Zero

For a smooth visualisation of motion of graph it is important to handle the floating point overflow errors in all the places using division operation. In the division of value a by value b, the absolute value of b shouldn't be less than some small threshold ϵ. Also maintaining the sign similar to the sign of b.

$$a/b = \begin{cases} a/b & \text{if } |b| > \epsilon \\ a/\epsilon & \text{if } b \geq 0 \\ -a/\epsilon & \text{if } b < 0 \end{cases}$$

5 References

1. https://webspace.maths.qmul.ac.uk/b.jackson/levicoFINAL.pdf
2. https://www.youtube.com/watch?v=k2jKCJ8fhj0