QP Solver

Key idea is to write the problem into an constrained quadratic programming formulation, like:

Constrained quadratic programming (QP) formulation:

$$\min \begin{bmatrix}\mathbf{p}_1\\ \vdots \\ \mathbf{p}_M\end{bmatrix}^T \begin{bmatrix}\mathbf{Q}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M\end{bmatrix} \begin{bmatrix}\mathbf{p}_1\\ \vdots \\ \mathbf{p}_M\end{bmatrix}$$

$$\text{s.t. } \mathbf{A}_{\text{eq}} \begin{bmatrix}\mathbf{p}_1\\ \vdots \\ \mathbf{p}_M\end{bmatrix} = \mathbf{d}_{eq}$$

Derivative constraints and continuity constraints are put into A_{eq} and d_{eq}. Then it is solved by the QP solver in MATLAB.

Closed-form Solver

The problem of finding an optimal polynomial can be transformed into finding optimal v, a at each waypoint, by the way of decision variable.

$$J = \begin{bmatrix}\mathbf{p}_1\\ \vdots \\ \mathbf{p}_M\end{bmatrix}^T \begin{bmatrix}\mathbf{Q}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M\end{bmatrix} \begin{bmatrix}\mathbf{p}_1\\ \vdots \\ \mathbf{p}_M\end{bmatrix} \quad + \quad \boldsymbol{M}_j \mathbf{p}_j = \mathbf{d}_j$$

$$\Downarrow$$

$$J = \begin{bmatrix}\mathbf{d}_1\\ \vdots \\ \mathbf{d}_M\end{bmatrix}^T \begin{bmatrix}\boldsymbol{M}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \boldsymbol{M}_M\end{bmatrix}^{-T} \begin{bmatrix}\mathbf{Q}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_M\end{bmatrix} \begin{bmatrix}\boldsymbol{M}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \boldsymbol{M}_M\end{bmatrix}^{-1} \begin{bmatrix}\mathbf{d}_1\\ \vdots \\ \mathbf{d}_M\end{bmatrix}$$

The next step is separating free and constrained variables in order to get a closed form solution.

$$\mathbf{C}^T \begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix} = \begin{bmatrix}\mathbf{d}_1\\ \vdots \\ \mathbf{d}_M\end{bmatrix} \quad \Longrightarrow \quad J = \begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}^T \underbrace{\mathbf{C}\boldsymbol{M}^{-T}\mathbf{Q}\boldsymbol{M}^{-1}\mathbf{C}^T}_{\mathbf{R}} \begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix} = \begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}^T \begin{bmatrix}\mathbf{R}_{FF} & \mathbf{R}_{FP}\\ \mathbf{R}_{PF} & \mathbf{R}_{PP}\end{bmatrix} \begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}$$

Finally, it is solving a optimal problem.

$$J = \mathbf{d}_F^T \mathbf{R}_{FF} \mathbf{d}_F + \mathbf{d}_F^T \mathbf{R}_{FP} \mathbf{d}_P + \mathbf{d}_P^T \mathbf{R}_{PF} \mathbf{d}_F + \mathbf{d}_P^T \mathbf{R}_{PP} \mathbf{d}_P$$

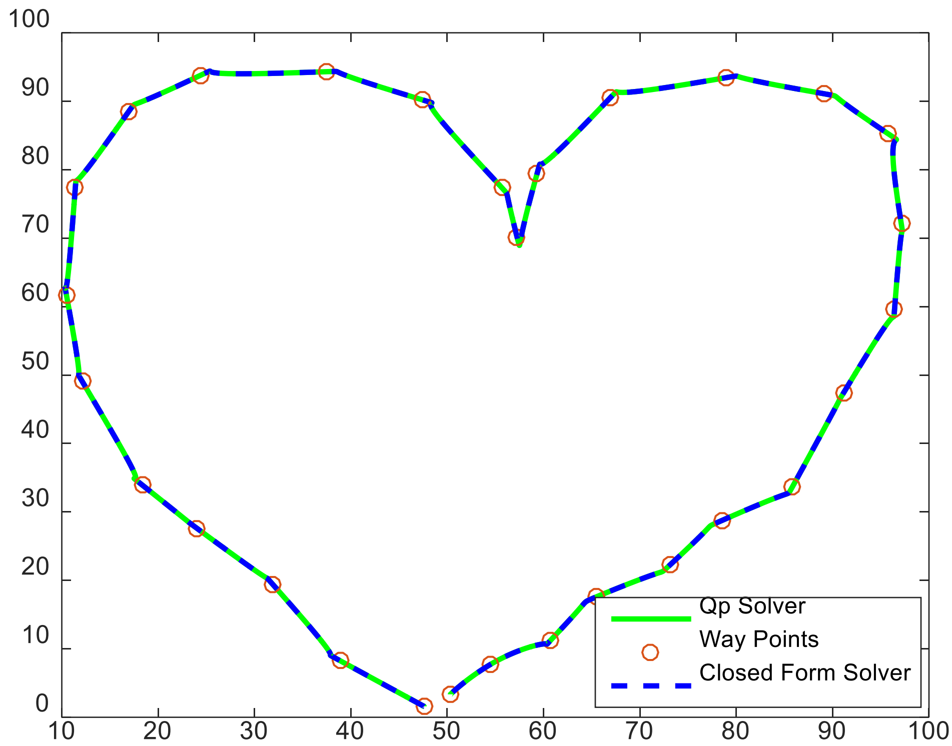
$$\mathbf{d}_P^* = -\mathbf{R}_{PP}^{-1} \mathbf{R}_{FP}^T \mathbf{d}_F$$

To understand it, we formulate a simple problem as three waypoints with only p and v two variables in each segment. Therefore, we have:

$$\mathrm{d}_\mathrm{P}^* = -\mathrm{R}_\mathrm{PP}^{-1} \mathrm{R}_\mathrm{FP} \mathrm{d}_\mathrm{F}$$

Simulation Results and Analysis

Minimum snap trajectory generated by QP solver and closed-form solution are shown in the following picture.



Analysis of two solutions:

QP solver	Expensive in iteration for a numerical solution
Closed-form solution	Expensive in matrix operation

Implementation Notes

The difficulty lies in making the matrix correct. Patience is important!