RANSAC Line Fitting

j.w.li

May 2025

1 Introduction

The following are some quick derivations meant to show how I got to the code. They are by no means perfect nor difficult even. We assume that the line we are fitting is in a 2D plane, as that's my personal use-case. Everywhere we assume that the following is true:

$$P = \left\{ \vec{p_i} \in \mathbb{R}^2 \,\middle|\, i = 0, \ldots, N-1 \right\}, \vec{p_i} = (x_i, y_i)^T, N \in \mathbb{N} \tag{1}$$

where P is the set of all points we fit over, $\vec{p_i}$ is one of those points and N is the amount of points we have.

We use the following indicator function for inliers:

$$I(a, b, \vec{p_i}, \delta) = \begin{cases} 1, & \text{if } \dfrac{\|ax_i + b - y_i\|}{\sqrt{a^2+1}} < \delta \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where we say δ is the threshold for how far a point may be. We then define the score for a line-fit as:

$$S(P, a, b, \delta) = \sum_{\vec{p_i}}^{P} I(a, b, \vec{p_i}, \delta) \tag{3}$$

For the purpose of evaluating on very small systems, we also evaluate every method on the following modified indicator function:

$$I'(a, b, \vec{p_i}, \delta) = \begin{cases} 1, & \text{if } \|ax_i + b - y_i\| < \delta \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where instead of the shortest Euclidean distance from a point to a line, we consider only the vertical deviation. In other words, the difference in the second dimension between the point $\vec{p_i}$ and the predicted point $ax_i + b$. For the derivations of the methods, however, this distinction does not matter.

2 Basic Algorithm

All algorithms follow the same basic principle. This basic RANSAC algorithm boilerplate code is outlined in Algorithm 1.

Algorithm 1: Basic RANSAC Algorithm

Input: Array P of size n, threshold δ, number of trials m
Output: Best fitting slope and intercept

$a_{best} \leftarrow 0$;
$b_{best} \leftarrow 0$;
$s_{best} \leftarrow 0$;
for $i \leftarrow 1$ **to** m **do**
    $(a, b) \leftarrow FitRandomLine(P)$;
    $s \leftarrow S(P, a, b, \delta)$;
    if $s > s_{best}$ **then**
        $a_{best} \leftarrow a$;
        $b_{best} \leftarrow b$;
        $s_{best} \leftarrow s$;
return $(a_{\text{best}}, b_{\text{best}})$;

where $FitRandomLine$ randomly fits a line using some data from the points P, and S is the scoring function as defined in Equation 3. As such the actual line fitting can be seen as a coroutine that is called by this RANSAC algorithm. The following sections will propose slightly different coroutines for the above given algorithm.

3 RANSAC Line Fitting using 2 Points

Classic RANSAC line fitting. We take 2 random points $\vec{p_i}, \vec{p_j}$ and calculate the line going through them.

$$a = \frac{y_i - y_j}{x_i - x_j}, b = y_i - ax_i$$

This can then be returned and used in the basic Algorithm 1.

4 RANSAC Line Fitting using 1 point through the origin

We can also do it by just using 1 point through the origin as our line. Then the line is defined by

$$a = \frac{y_i}{x_i}, b = 0$$

For our sampled point $\vec{p_i}$. This can then be returned and used in the basic Algorithm 1.

5 RANSAC Line Fitting using n Points with MSE

We can also use n points and use MSE to find a line. In this case we can take for example 3 or 5 points, and use it to estimate a line. This line can then be used upstream in the RANSAC algorithm.

We want to fit the line

$$y = ax$$

as we assume there is no slope. Then we know that the squared error for some slope a is the following for the samples $S \subseteq P$ that we choose.

$$\sum_{\vec{p_i}=(x_i,y_i)^T}^{S} (y_i - ax_i)^2$$

If we want to minimise this, we can just derive this w.r.t a

$$\frac{d}{da} \sum_{\vec{p_i}=(x_i,y_i)^T}^{S} (y_i - ax_i)^2 = \sum_{\vec{p_i}=(x_i,y_i)^T}^{S} -2x_i(y_i - ax_i)$$

After setting this to 0 we get

$$\begin{aligned}
\sum_{\vec{p_i}=(x_i,y_i)^T}^{S} -2x_i(y_i - ax_i) &= 0 \\
\sum_{\vec{p_i}=(x_i,y_i)^T}^{S} x_i(y_i - ax_i) &= 0 \\
\sum_{\vec{p_i}=(x_i,y_i)^T}^{S} x_i y_i - ax_i^2 &= \sum_{\vec{p_i}=(x_i,y_i)^T}^{S} ax_i^2 \\
\sum_{\vec{p_i}=(x_i,y_i)^T}^{S} x_i y_i &= a \sum_{\vec{p_i}=(x_i,y_i)^T}^{S} x_i^2 \\
\frac{\sum_{\vec{p_i}=(x_i,y_i)^T}^{S} x_i y_i}{\sum_{\vec{p_i}=(x_i,y_i)^T}^{S} x_i^2} &= a
\end{aligned}$$

It should be noted this is just equal to 2 inner products, and should be implemented when the hardware allows for as such. In our case, the RP2040 does not have SIMD registers and thus we ignore this.