VINS-mono詳細解讀

2018-04-15 / VIEWS: 133

VINS-mono詳細解讀

極品巧克力

前言

Vins-mono是香港科技大學開源的一個VIO算法，https://github.com/HKUST-Aerial-Robotics/VINS-Mono，是用緊耦合方法實現的，通過單目+IMU恢復出尺度，效果非常棒。

感謝他們開源，我從中學到了非常多的知識。源碼總共有15000多行，我在通讀完進程之後，結合參考文獻，把進程背後的算法理論都推導了一遍，總結成了本文，與各位分享。

本文目標讀者：對vins-mono有一定了解的SLAM算法工程師。由於進程裏有非常多的實現細節，建議讀者在讀本文前，先讀一遍vins-mono的進程。

1.特徵點跟蹤

首先用cv::goodFeaturesToTrack在第一幀圖像上面找最強的150個特徵點，非極大值抑制半徑為30。新的特徵點都有自己的新的對應的id。然後在下一幀過來時，對這些特徵點用光流法進行跟蹤，在下一幀上找匹配點。然後對前後幀中這些匹配點進行校正。先對特徵點進行畸變校正，再投影到以原點為球心，半徑為1的球面上，再延伸到深度歸一化平面上，獲得最終校正後的位置。對於每對匹配點，基於校正後的位置，用F矩陣加ransac來篩選。然後再在匹配上的特徵點之外的區域，用cv::goodFeaturesToTrack搜索最強的新的特徵點，把特徵點數量補上150個。

最後，把剩下的這些特徵點，把圖像點投影回深度歸一化平面上，再畸變校正，再投影到球面上，再延伸到深度歸一化平面上，得到校正後的位置。把校正後的位置發送出去。

特徵點跟蹤和匹配，就是前一幀到這一幀的，一幀幀繼承下去。或者生成新的特徵點。

2.初始化

2.1外參中的旋轉

用機器人手眼標定的方法，計算出外參中的旋轉。

$$q_{b_{k+1}}^{b_k} \otimes q_c^b = q_c^b \otimes q_{c_{k+1}}^{c_k}$$
$$\Rightarrow \left[Q_1\left(q_{b_{k+1}}^{b_k}\right) - Q_2\left(q_{c_{k+1}}^{c_k}\right)\right] q_c^b = Q_{k+1}^k q_c^b = 0$$

其中

$$q_{b_{k+1}}^{b_k}$$

是陀螺儀預積分得到的，

$$q_{c_{k+1}}^{c_k}$$

是用8點法對前後幀對應的特徵點進行計算得到的。詳細見《Monocular Visual-Inertial State Estimation With Online Initialization and Camera-IMU Extrinsic Calibration》。

2.2 SFM

先在關鍵幀窗口裏面，找到第l幀，第l幀與最後一幀有足夠的像素位移，並且能用8點法算出旋轉和位移。以l幀的姿態為世界座標系。先從l幀開始與最後一幀進行三角定位，再用pnp估計出下一幀的位姿，下一幀再與最後一幀三角定

位得出更多的三維點。重複到倒數第二幀。從l幀開始往第一幀，逐漸幀pnp，再與第l幀進行三角定位得到更多的三維點。每幀pnp時的位姿初值都用上一個關鍵幀的的位姿。剩下的那些還沒有被三角定位的特徵點，通過它被觀察到的第一幀和最後一幀進行三角定位。

固定住l幀的位置和姿態，固定住最後一幀的位置。因為這時候的圖像位姿和點的位置都不太準，所以用ceres統一一起優化圖像位姿和三維點位置，優化重投影誤差。優化的測量值是，特徵點在每幀中被觀察到的位置，可以轉成重投影誤差約束。有關的自變量是，每幀圖像的位姿，特徵點的三維座標。

優化完成之後，即用ceres優化出這些關鍵幀的位姿和地圖點後，再用pnp算出在這段時間區域內的所有圖像的位姿。每個圖像的計算都用下一個關鍵幀的位姿來當pnp的初值。

$$\begin{bmatrix} P_0^w \\ q_0^w \\ P_1^w \\ P_1^w \\ P_0^w \end{bmatrix} = \begin{bmatrix} \frac{P_0^0}{P_0^0(3)} - \hat{P}_0^0 \\ \frac{P_0^1}{P_0^1(3)} - \hat{P}_0^1 \end{bmatrix}$$

$$P_0^0 = \left(R\left(q_0^w\right)\right)^T P_0^w - \left(R\left(q_0^w\right)\right)^T p_0^w$$

$$P_0^1 = \left(R\left(q_1^w\right)\right)^T P_0^w - \left(R\left(q_1^w\right)\right)^T p_1^w$$

進程裏面沒有求雅克比，而是用自動求導的方法。

2.3 計算陀螺儀的偏移

在2.1中已經根據連續圖像的相對旋轉算出相機和IMU間的外參旋轉了，現在要再根據上一節2.2中的SFM算出來的各幀圖像的相對旋轉來計算出陀螺儀的偏移。

就是根據前後幀之間的根據陀螺儀預積分出來的旋轉

$$q_{\text{intergrate},ij}$$

與基於SFM圖像算出來的旋轉轉換到IMU座標系的相對旋轉

$$q_{ij}$$

之間的矢量差的兩倍。

在進程裏面，每次算出的圖像的姿態

$$R_{c_i}^{c_0}$$

，都會轉換成

$$R_{b_i}^{c_0} = R_{c_i}^{c_0} R_b^c$$

。然後在計算相對IMU的姿態時，就用

$$R_{b_k}^{b_i} = \left(R_{b_i}^{c_0}\right)^T R_{b_k}^{c_0}$$

。

這裏是採用了近似計算的方法。其實就是把角度的殘差轉換成了角軸差的形式。詳見《從角軸到四元數微分方程》和《on-manifold詳細解讀》。

$$q_{residual} = q_{\text{intergrate,ji}} \otimes q_{ij} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\theta_x \\ \frac{1}{2}\theta_y \\ \frac{1}{2}\theta_z \end{bmatrix} \Rightarrow \theta_{residual} = 2\begin{bmatrix} q_{residual,x} \\ q_{residual,y} \\ q_{residual,z} \end{bmatrix}$$

這個

$$\theta_{residual}$$

關於陀螺儀的偏移求導，得到雅克比矩陣。然後再根據高斯牛頓法算出陀螺儀偏移。

$$\begin{bmatrix} \frac{\partial \theta_1}{\partial b_g} \\ \vdots \end{bmatrix} \begin{bmatrix} b_g \end{bmatrix} = \begin{bmatrix} \theta_{residual,1} \\ \vdots \end{bmatrix}$$

算出陀螺儀的偏移以後，對於每一幅圖像，利用緩存的每一個IMU數據，重新計算這一段時間的每幀圖像對應的預積分，雅克比矩陣和協方差矩陣。

雅克比矩陣的初值是單位陣，協方差矩陣的初值為零。雅克比矩陣每次都乘以一個狀態轉移矩陣。協方差矩陣每次都左右乘狀態轉移矩陣，再加上一個噪聲矩陣，噪聲矩陣就是加速度和陀螺儀噪聲所形成的噪聲協方差矩陣。

2.4速度，重力和尺度對齊

從2.3之後，認為陀螺儀是準確的，每幀圖像的姿態也都是準確的。

速度，重力和尺度對齊，其實就是，每一幀對後一幀的位置，速度的預測值，與當前值的誤差。當前值

$$p_{b_k}^{b_0}$$

的位置，是SFM位置

$$p_{c_k}^{c_0}$$

通過外參

$$T_c^b$$

計算過來，給的。與這有關的自變量是每幀的速度，重力，尺度。速度，重力，尺度，給的初值都是零。

以第一幀和第二幀為例。

$$\begin{bmatrix} v_{b_0}^{b_0} \\ v_{b_1}^{b_0} \\ \vdots \\ g^{b_0} \\ s \end{bmatrix} = \begin{bmatrix} sp_{b_0}^{b_0} + v_{b_0}^{b_0}\Delta t + R_{b_0}^{b_0}\alpha_1^0 - \frac{1}{2}g^{b_0}\Delta t^2 - sp_{b_1}^{b_0} \\ v_{b_0}^{b_0} + R_{b_0}^{b_0}\beta_1^0 - g^{b_0}\Delta t - v_{b_1}^{b_0} \\ sp_{b_1}^{b_0} + v_{b_1}^{b_0}\Delta t + R_{b_1}^{b_0}\alpha_2^1 - \frac{1}{2}g^{b_0}\Delta t^2 - sp_{b_2}^{b_0} \\ v_{b_1}^{b_0} + R_{b_1}^{b_0}\beta_2^1 - g^{b_0}\Delta t - v_{b_2}^{b_1} \end{bmatrix}$$

因為速度，重力尺度的初值全部都是零，所以第一步，這裏殘差的計算可以簡化成如下。

$$residual = \begin{bmatrix} R_{b_0}^{b_0}\alpha_1^0 \\ R_{b_0}^{b_0}\beta_1^0 \\ R_{b_1}^{b_0}\alpha_2^1 \\ R_{b_1}^{b_0}\beta_2^1 \end{bmatrix}$$

但雅克比的計算，還是得用原來的表達式算。

$$\begin{bmatrix} \Delta t & 0 & \cdots & -\frac{1}{2}\Delta t^2 & p_{b_0}^{b_0} - p_{b_1}^{b_0} \\ I & -I & \cdots & -\Delta t & 0 \\ 0 & \Delta t & \cdots & -\frac{1}{2}\Delta t^2 & p_{b_1}^{b_0} - p_{b_2}^{b_0} \\ 0 & I & \cdots & -\Delta t & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} v_{b_0}^{b_0} \\ v_{b_1}^{b_0} \\ \vdots \\ g^{b_0} \\ s \end{bmatrix} = \begin{bmatrix} sp_{b_0}^{b_0} + v_{b_0}^{b_0}\Delta t + R_{b_0}^{b_0}\alpha_1^0 - \frac{1}{2}g^{b_0}\Delta t^2 - sp_{b_1}^{b_0} \\ v_{b_0}^{b_0} + R_{b_0}^{b_0}\beta_1^0 - g^{b_0}\Delta t - v_{b_1}^{b_0} \\ sp_{b_1}^{b_0} + v_{b_1}^{b_0}\Delta t + R_{b_1}^{b_0}\alpha_2^1 - \frac{1}{2}g^{b_0}\Delta t^2 - sp_{b_2}^{b_0} \\ v_{b_1}^{b_0} + R_{b_1}^{b_0}\beta_2^1 - g^{b_0}\Delta t - v_{b_2}^{b_1} \\ \vdots \end{bmatrix}$$

雖然給的初值是零，但因為雅克比矩陣裏面的元素都是與自變量無關的常數，表示這是線性的，所以只用高斯牛頓法計算一次就可以了。只要數據足夠，就能算出比較準確的值。

其中，當前值

$$p_{b_z}^{b_0}$$

的位置，是SFM位置

$$p_{c_z}^{c_0}$$

通過外參

$$T_c^b$$

計算過來，給的。機器人手眼標定。

$$T_c^b T_{c_k}^{c_0} \left(T_c^b\right)^{-1}$$

$$\begin{bmatrix} t_{b_k}^{b_0} \\ 1 \end{bmatrix} = \begin{bmatrix} R_c^b & t_c^b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{c_k}^{c_0} & t_{c_k}^{c_0} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \left(R_c^b\right)^T & -\left(R_c^b\right)^T t_c^b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_c^b R_{c_k}^{c_0} & R_c^b t_{c_k}^{c_0} + t_c^b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \left(R_c^b\right)^T & -\left(R_c^b\right)^T t_c^b \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} R_c^b R_{c_k}^{c_0} \left(R_c^b\right)^T & -R_c^b R_{c_k}^{c_0} \left(R_c^b\right)^T t_c^b + R_c^b t_{c_k}^{c_0} + t_c^b \\ 0 & 1 \end{bmatrix}$$

$$t_{b_k}^{b_0} = -R_c^b R_{c_k}^{c_0} \left(R_c^b\right)^T t_c^b + R_c^b t_{c_k}^{c_0} + t_c^b = R_c^b \left(t_{c_k}^{c_0} - R_{c_k}^{c_0} \left(R_c^b\right)^T t_c^b\right) + t_c^b$$

$$\begin{aligned} p_{b_0}^{b_0} - p_{b_1}^{b_0} &= R_c^b \left(t_{c_0}^{c_0} - R_{c_0}^{c_0} \left(R_c^b\right)^T t_c^b\right) + t_c^b - R_c^b \left(t_{c_1}^{c_0} - R_{c_1}^{c_0} \left(R_c^b\right)^T t_c^b\right) - t_c^b \\ &= R_c^b \left(t_{c_0}^{c_0} - R_{c_0}^{c_0} \left(R_c^b\right)^T t_c^b - t_{c_1}^{c_0} + R_{c_1}^{c_0} \left(R_c^b\right)^T t_c^b\right) \\ &= R_c^b \left(t_{c_0}^{c_0} - t_{c_1}^{c_0} + \left(R_{c_1}^{c_0} - R_{c_0}^{c_0}\right) \left(R_c^b\right)^T t_c^b\right) \\ &= R_c^b \left(t_{c_0}^{c_0} - t_{c_1}^{c_0} - \left(R_{c_0}^{c_0} - R_{c_1}^{c_0}\right) \left(R_c^b\right)^T t_c^b\right) \end{aligned}$$

當然，進程裏面實際上是把殘差轉換成了

$$v_{b_k}^{c_0}$$

這樣的奇怪的座標系下，結果就會像《technical report》裏面的公式18那樣子，全部都乘以

$$q_{c_0}^{b_z}$$

。但是，我覺得像我上面那樣的用機器人手眼標定的方法來表示，會比較方便容易易懂直觀明瞭，更不容易在寫表達式的時候出錯。

雖然給的初值是零，雖然只用高斯牛頓法計算一次，但只要數據足夠，也能算出比較準確的值。每增加一幀，就用高斯牛頓法全部計算一次，速度重力尺度的初值全部都是零。因為反正都可以用高斯牛頓法一次算出結果，所以就不用繼承之前的值來優化。

直到優化後的重力接近9.8，比如重力模在[8.8,10.8]區間內。這個方法挺好的，不用先驗知識就把重力優化到9.8，結果令人信服。

進程裏面，會把關於尺度求導得到的雅克比除以100，這就意味着，尺度這個變量對殘差的影響力減弱了100倍。最終為了能消去殘差，優化後的尺度會比實際的大100倍。得到後，要再除以100。這麼做的目的，應該是要讓尺度的精度更高。

每次都直接把這一塊的雅克比矩陣和對應的殘差，轉換成H矩陣的形式，直接加到H矩陣上。為什麼要這樣做呢？為什麼不把全部的雅克比矩陣算好之後再一次性地轉換成H矩陣。因為雅克比矩陣太巨大了，而且非常稀疏，裏面很多元素都是零。所以，可以直接根據雅克比矩陣的計算表達式知道哪些位置是非零的，然後非零的位置對應相乘加到入H矩陣中對應的位置，即節省存儲空間，又能加快計算。

2.5 進一步優化重力

在2.4計算出來一個相對準確的值的基礎上，還要再加個約束，即重力的模為9.8。可以有兩個方法。

第一個方法是，直接在2.4的殘差約束裏面增加一個，

$$\left\|g^{b_0}\right\| - 9.8$$

。

$$\begin{bmatrix} \Delta t & 0 & \cdots & -\frac{1}{2}\Delta t^2 & p_{b_0}^{b_0} - p_{b_1}^{b_0} \\ I & -I & \cdots & -\Delta t & 0 \\ 0 & \Delta t & \cdots & -\frac{1}{2}\Delta t^2 & p_{b_1}^{b_0} - p_{b_2}^{b_0} \\ 0 & I & \cdots & -\Delta t & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{\partial\left(\left\|g^{b_0}\right\|-9.8\right)}{\partial g^{b_0}} & 0 \end{bmatrix} \begin{bmatrix} v_{b_0}^{b_0} \\ v_{b_1}^{b_0} \\ \vdots \\ g^{b_0} \\ s \end{bmatrix} = \begin{bmatrix} sp_{b_0}^{b_0} + v_{b_0}^{b_0}\Delta t + R_{b_0}^{b_0}\alpha_1^0 - \frac{1}{2}g^{b_0}\Delta t^2 - sp_{b_1}^{b_0} \\ v_{b_0}^{b_0} + R_{b_0}^{b_0}\beta_1^0 - g^{b_0}\Delta t - v_{b_1}^{b_0} \\ sp_{b_1}^{b_0} + v_{b_1}^{b_0}\Delta t + R_{b_1}^{b_0}\alpha_2^1 - \frac{1}{2}g^{b_0}\Delta t^2 - sp_{b_2}^{b_0} \\ v_{b_1}^{b_0} + R_{b_1}^{b_0}\beta_2^1 - g^{b_0}\Delta t - v_{b_2}^{b_0} \\ \vdots \text{(cnblogs.com/ilekoaiq)} \\ \left\|g^{b_0}\right\| - 9.8 \end{bmatrix}$$

$$\frac{\partial\left(\left\|g^{b_0}\right\|-9.8\right)}{\partial g^{b_0}} = \frac{\partial\left\|g^{b_0}\right\|}{\partial g^{b_0}} = \frac{\partial\left(\left(x^2+y^2+z^2\right)^{\frac{1}{2}}\right)}{\partial[x \quad y \quad z]} = \left(x^2+y^2+z^2\right)^{-\frac{1}{2}}[x \quad y \quad z]$$

但是，如果這樣的話，雅克比矩陣裏面就有了與自變量有關的元素，就不是線性的了。這樣子，計算起來就會比較麻煩，可能需要迭代很多次。

所以，採用方法二。

對模的方向進行優化。2.4的優化僅僅是提供一個較好的重力的方向。

直接給重力一個初值，模的大小就是9.8，方向的初值是2.4中優化後的重力的方向。對這個重力，在切線上調整它的方向。

$$b_1 = \left\|\bar{g}^{b_0} \times [1,0,0]\right\|$$
$$b_2 = \left\|\bar{g}^{b_0} \times b_1\right\|$$

$$\begin{bmatrix} -\frac{1}{2}b_1\Delta t^2 & -\frac{1}{2}b_2\Delta t^2 & p_{b_0}^{b_0} - p_{b_1}^{b_0} \\ -b_1\Delta t & -b_2\Delta t & 0 \\ -\frac{1}{2}b_1\Delta t^2 & -\frac{1}{2}b_2\Delta t^2 & p_{b_1}^{b_0} - p_{b_2}^{b_0} \\ -b_1\Delta t & -b_2\Delta t & 0 \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} v_{b_0}^{b_0} \\ v_{b_1}^{b_0} \\ \vdots \\ lx \\ ly \\ s \end{bmatrix} = \begin{bmatrix} sp_{b_0}^{b_0} + v_{b_0}^{b_0}\Delta t + R_{b_0}^{b_0}\alpha_1^0 - \frac{1}{2}\left(g^{b_0} + lx \cdot b_1 + ly \cdot b_2\right)\Delta t^2 - sp_{b_1}^{b_0} \\ v_{b_0}^{b_0} + R_{b_0}^{b_0}\beta_1^0 - \left(g^{b_0} + lx \cdot b_1 + ly \cdot b_2\right)\Delta t - v_{b_1}^{b_0} \\ sp_{b_1}^{b_0} + v_{b_1}^{b_0}\Delta t + R_{b_1}^{b_0}\alpha_2^1 - \frac{1}{2}\left(g^{b_0} + lx \cdot b_1 + ly \cdot b_2\right)\Delta t^2 - sp_{b_2}^{b_0} \\ v_{b_1}^{b_0} + R_{b_1}^{b_0}\beta_2^1 - \left(g^{b_0} + lx \cdot b_1 + ly \cdot b_2\right)\Delta t - v_{b_2}^{b_0} \\ \vdots \end{bmatrix}$$

因為是線性的，所以自變量的初值直接全部設為零。直接一步高斯牛頓法，算出最優值。用

$$g^{b_0}{}_{new} = \frac{g^{b_0} + lx \cdot b_1 + ly \cdot b_2}{\left\|g^{b_0} + lx \cdot b_1 + ly \cdot b_2\right\|} * 9.8$$

來更新

$$g^{b_0}$$

。

重複上述步驟四次，即

$$g^{b_0}$$

更新四次後，就認為這些變量，速度，重力，尺度都已經優化到一個很優的值了。這時候，尺度應該是大於零的。

手動在切線上調整，然後再控制模。其實，就是由自動的非線性優化，改成了手動的線性優化，用來控制迭代次數。

2.6優化地圖

用奇異值分解的方法，對每一個特徵點都優化重投影誤差最小化，優化出它在被觀察到的第一幀的相機座標系下的位置，然後只把深度拿過來用。設它們在被觀察到的第一幀的相機座標系下的齊次座標為

$$P$$

，其它觀察幀相對觀察第一幀的位姿為

$$T$$

。已知它在每幀的投影點，最小化它在每幀的重投影誤差。相當於是尋找空間中的一點，與，每幀的光心從投影點發出的射線，距離最小。

與2.2SFM時算出的地圖點的區別在於，2.2SFM時算出的地圖點是ceres優化出來的關鍵幀及其對應的地圖點。而這裏，是固定這段時間的所有的關鍵幀，這些關鍵幀的姿態在2.3後期算出陀螺儀偏移後，已經調整過了。奇異值分解，最小化重投影誤差，算出地圖點的位置。只取在被觀察到的第一幀的深度。因為在這時候，各個圖像的位姿已經相對比較準確了，所以與2.2中的cere圖像位姿和地圖點全部優化不同，這裏只需要優化地圖點就可以了。

$$T=\begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$$

$$P_2=T_{21}P_1$$

$$\begin{bmatrix} (P_2(1)/P_2(4))/(P_2(3)/P_2(4)) \\ (P_2(2)/P_2(4))/(P_2(3)/P_2(4)) \end{bmatrix}=\begin{bmatrix} P_2(1)/P_2(3) \\ P_2(2)/P_2(3) \end{bmatrix}=\begin{bmatrix} u_2 \\ v_2 \end{bmatrix}$$

$$\Rightarrow\begin{bmatrix} T_1P_1/T_3P_1 \\ T_2P_1/T_3P_1 \end{bmatrix}=\begin{bmatrix} u_2 \\ v_2 \end{bmatrix}\Rightarrow\begin{bmatrix} T_1P_1 \\ T_2P_1 \end{bmatrix}=\begin{bmatrix} u_2\cdot T_3P_1 \\ v_2\cdot T_3P_1 \end{bmatrix}\Rightarrow\begin{bmatrix} T_1P_1-u_2\cdot T_3P_1 \\ T_2P_1-v_2\cdot T_3P_1 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow\begin{bmatrix} T_1^{1,1}-u_1\cdot T_3^{1,1} \\ T_1^{1,1}-v_1\cdot T_3^{1,1} \\ T_1^{1,2}-u_1\cdot T_3^{1,2} \\ T_1^{1,2}-v_1\cdot T_3^{1,2} \\ \vdots \end{bmatrix}P_1=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix},\text{s.t.(cnblogs.com/ilekoaiq)}T_3P_1\neq 0$$

在算奇異值分解時，還要增加

$$P_1$$

模為1的約束。齊次座標

$$P$$

為一個四維的矢量，這樣子，通過增加模為1的限制，就可以用奇異值分解來優化，這就是這裏用齊次座標的好處。（在進程實際運行的時候，上面方程的左邊，還除以了歸一化的模，但不會影響計算結果）

基於2.3中的陀螺儀的偏移，重新計算每相鄰兩個關鍵幀之前的相對姿態，算的是關鍵幀之間的預積分。用的是這一段時間緩存的。IMU數據，用中點插值的方法。

然後用2.5中優化出來的尺度，計算出窗口裏面的每個關鍵幀的新的位置，它們相對於第一個關鍵幀的位姿，把第一個關鍵幀當作世界座標系。用2.5中優化出來的速度賦值給每一個對應的關鍵幀。用尺度來調整每一個特徵點的深度。

然後根據重力，計算出當前的世界座標系相對於水平座標系的旋轉，即第一個關鍵幀相對於水平座標系的旋轉。然後，把所有關鍵幀的位置，姿態和速度都轉到水平座標系上。

3.正常的跟蹤

每新進來一張圖片，上面有跟蹤出來的特徵點。

對於f_manager中的feature列表中的那些還沒有深度的特徵點，用奇異值分解計算出一個它的座標，使得它在它被觀察到的每幀圖像上的重投影誤差最小。它的座標用在它被觀察到的第一幀圖像的相機座標系的深度表示。因為它還有它在被觀察到的每幀圖像上的歸一化座標。

然後用cere來優化。結合各個關鍵幀的位姿，各個相機的外參，邊緣化的信息，與預積分的誤差，每個特徵點的重投影誤差，迴環閉環誤差。進行優化。

然後滑動窗口。判斷邊緣化的條件是，是否進來一個新的關鍵幀。

如果有邊緣化的話，則把窗口中最前面的一個關鍵幀滑掉。然後把第一次被這個關鍵幀觀察到的特徵點，都轉移到新的第0個關鍵幀上。如果沒有邊緣化的話，則把之前最新的這一幀用進來的最新的這一幀替換掉。

3.1判斷是否是關鍵幀

窗口的大小默認是10。

每當進來一個新的圖像幀的時候，首先判斷它與窗口裏面存儲的之前的那一幀的的相對位移，就是與第10幀的特徵點的相對位置，用特徵點的相對位移來表示。

如果特徵點的平均相對位移大於某一個閾值，就認為是一個新的關鍵幀。就把這個新的關鍵幀壓入到窗口裏面，就是壓入到第10個位置，然後其它的關鍵幀都往前移動。第一個位置的關鍵幀被移出去，邊緣化。

如果不是新的關鍵幀，就把之前的第10幀邊緣化掉，這個新的一幀替換成為第10幀。

總之，無論是哪種情況，這個新的一幀肯定都會成為窗口裏面的第10幀。

邊緣化，是在優化之後才進行的，而且最新的這幀上面觀察到的新的特徵點並不參與優化。所以，優化的時候，是包括最新的這一幀的11幀的姿態，以及前10幀的特徵點在每一幀的投影，包括它們在最新這幀的投影點。所以para_Pose之類的待優化變量的數組長度是11。

3.2創建地圖點

對於f_manager中的feature列表中的那些還沒有深度的特徵點，如果它被之前2幀以上的關鍵幀觀察到過，用奇異值分解計算出一個它的座標，使得它在它被觀察到的每幀圖像上的重投影誤差最小。它的座標用在它被觀察到的第一幀圖像的相機座標系的深度表示。因為它還有它在被觀察到的每幀圖像上的歸一化座標。

如果以後要把VINS改用深度相機的話，可以在這裏修改。1.信任深度相機，這個點被觀察到的第一次的位置就是準確值，直接加入地圖點。2.怕深度相機有誤差，所以加判斷，這個點在它被觀察到的連續兩幀裏面，在世界座標系中的三維座標不能相差太大，如果相差太大，就把它的第一幀的觀察記錄刪掉。如果相差不大，就取兩個三維座標的平均值，作為該點的三維位置，加入地圖點。3.還可以借鑑ORBSLAM裏面的篩選地圖點的方法，該點需要在連續幾幀中被觀察到，並且這連續幾幀，觀察到的它在世界座標系中的三維座標，不能相差太大，就認為它是一個好的點，加入地圖點。之後的cere優化中，就不再優化地圖點，可以極大地加快優化速度。

3.3cere優化

要優化的目標是各幀的位置，姿態，速度，偏移，以及相機的外參，以及每一個特徵點在它被觀察到的第一幀的深度。即，要優化的自變量是。

$$\chi = \begin{bmatrix} p_{b_0}^w \\ v_{b_0}^w \\ q_{b_0}^w \\ b_{a,b_0} \\ b_{g,b_0} \\ p_{b_1}^w \\ v_{b_1}^w \\ q_{b_1}^w \\ b_{a,b_1} \\ b_{g,b_1} \\ \vdots \\ p_{b_n}^w \\ v_{b_n}^w \\ q_{b_n}^w \\ b_{a,b_n} \\ b_{g,b_n} \\ p_c^b \\ q_c^b \\ \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{m-1} \\ \lambda_m \end{bmatrix}$$

要擬合的目標是，之前邊緣化後的先驗值，前後幀之間的IMU的預積分值，再加上前後幀之間的偏移的差的約束，每一個特徵點的重投影位置。

$$\hat{y} = \begin{bmatrix} p^{w}_{prior,b_0} \\ q^{w}_{prior,b_0} \\ v^{w}_{prior,b_0} \\ b_{prior,a,b_0} \\ b_{prior,g,b_0} \\ p^{w}_{prior,b_1} \\ q^{w}_{prior,b_1} \\ v^{w}_{prior,b_1} \\ b_{prior,a,b_1} \\ b_{prior,g,b_1} \\ \vdots \\ p^{w}_{prior,b_n} \\ q^{w}_{prior,b_n} \\ v^{w}_{prior,b_n} \\ b_{prior,a,b_n} \\ b_{prior,g,b_n} \\ \alpha^{b_0}_{b_1} \\ \beta^{b_0}_{b_1} \\ \gamma^{b_0}_{b_1} \\ \vdots \\ \alpha^{b_{n-1}}_{b_n} \\ \beta^{b_{n-1}}_{b_n} \\ \gamma^{b_{n-1}}_{b_n} \\ \hat{\mathbf{P}}^{c_0}_{0} \\ \hat{\mathbf{P}}^{c_1}_{0} \\ \hat{\mathbf{P}}^{c_2}_{0} \\ \vdots \\ \hat{\mathbf{P}}^{c_0}_{1} \\ \hat{\mathbf{P}}^{c_1}_{1} \\ \hat{\mathbf{P}}^{c_2}_{1} \\ \vdots \\ \hat{\mathbf{P}}^{c_n}_{m} \end{bmatrix}$$

3.3.1與先驗值的殘差

其中，prior代表先驗值。

$$\alpha$$

代表前後幀之間的加速度計的二次積分，

$$\beta$$

代表前後幀之間的加速度計的一次積分。

$$\gamma$$

代表前後幀的陀螺儀的一次積分。

另外，要再加上前後幀之間的偏移的差的約束。

所以，在當前的自變量的情況下，與目標的殘差為residual，表示如下。

$$
residual = \begin{bmatrix}
p_{b_0}^{w} - p_{prior,b_0}^{w} \\
q_{b_0}^{w} - q_{prior,b_0}^{w} \\
v_{b_0}^{w} - v_{prior,b_0}^{w} \\
b_{a,b_0} - b_{prior,a,b_0} \\
b_{g,b_0} - b_{prior,g,b_0} \\
\vdots \text{(cnblogs.com/ilekoaiq)} \\
q_{w}^{b_0}\left(p_{b1}^{w} - p_{b0}^{w} + \frac{1}{2}g^{w}\Delta t^{2}\right) - q_{w}^{b_0}v_{b_0}^{w}\Delta t - \alpha_1^0 \\
q_{w}^{b_0}\left(v_{b_1}^{w} + g^{w}\Delta t\right) - q_{w}^{b_0}v_{b_0}^{w} - \beta_1^0 \\
2\left[\left(q_{b_1}^{w}\right)^{-1} \otimes q_{b_0}^{w} \otimes \gamma_{b_1}^{b_0}\right]_{xyz} \\
b_{a,b_1} - b_{a,b_0} \\
b_{g,b_1} - b_{g,b_0} \\
\vdots \\
\hat{\mathbf{P}}_1^{f_0} - \mathbf{P}_1^{f_0} \\
\vdots
\end{bmatrix}
$$

其中，與先驗值的殘差，為了計算方便，可以再轉換到之前的舒爾補之後的殘差空間裏。用第一狀態雅克比。因為在上一次邊緣化之前，已經優化之後的殘差，舒爾補之後的residual，已經是最小二乘結果，它反饋到自變量上面是接近於零（因為各個方向的量都抵消了）。所以，如果這時候，如果對自變量調整了dx，則與之前的舒爾補後的值的殘差會增加J*dx。

其實，在這裏，不僅僅是要表示與之前的先驗值的差，而是要表示在之前的誤差的基礎上面疊加上來的誤差。因為之前的誤差是最小二乘結果，而不是全零。

```
//每個關鍵幀的位姿與先驗值的不同，會造成殘差項增大
Eigen::Map<Eigen::VectorXd>(residuals, n) = marginalization_info->linearized_residuals +marginalization_info->linearized_jacobians * dx;
```

所以，第一部分的雅克比就是之前邊緣化後的雅克比，誤差就是之前的誤差再加上新的誤差。

第一部分，要擬合的約束目標是，每幀與之前邊緣化後的位姿、速度和偏移不能相差太大。殘差用，之前的邊緣化後的殘差，加上，現在的自變量與之前邊緣化後的自變量的相差的值乘以第一狀態雅克比。與之相關的自變量是，每幀的位姿，速度和偏移，相機與IMU的相對位姿。殘差關於要優化的自變量求導，得到雅克比矩陣。

$$
J\begin{bmatrix}
p_{b_0}^w \\ q_{b_0}^w \\ v_{b_0}^w \\ b_{a,b_0} \\ b_{g,b_0} \\ p_{b_1}^w \\ q_{b_1}^w \\ v_{b_1}^w \\ b_{a,b_1} \\ b_{g,b_1} \\ \vdots \\ p_{b_n}^w \\ q_{b_n}^w \\ v_{b_n}^w \\ b_{a,b_n} \\ b_{g,b_n} \\ p_c^b \\ q_c^b \\ \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{m-1} \\ \lambda_m
\end{bmatrix} =
\begin{bmatrix}
p_{b_0}^w - p_{prior,b_0}^w \\
q_{b_0}^w - q_{prior,b_0}^w \\
v_{b_0}^w - v_{prior,b_0}^w \\
b_{a,b_0} - b_{prior,a,b_0} \\
b_{g,b_0} - b_{prior,g,b_0} \\
\vdots \\
q_w^{b_0}\left(p_{b1}^w - p_{b0}^w + \frac{1}{2}g^w\Delta t^2 - v_{b_0}^w\Delta t\right) - \alpha_1^0 \\
q_w^{b_0}\left(v_{b_1}^w + g^w\Delta t - v_{b_0}^w\right) - \beta_1^0 \\
2\left[\left(q_{b_1}^w\right)^{-1} \otimes q_{b_0}^w \otimes \gamma_{b_1}^{b_0}\right]_{xyz} \\
b_{a,b_1} - b_{a,b_0} \\
b_{g,b_1} - b_{g,b_0} \\
\vdots \text{(cnblogs.com/ilekoaiq)} \\
\hat{\mathrm{P}}_1^{f_0} - \mathrm{P}_1^{f_0} \\
\vdots
\end{bmatrix}
$$

3.3.2 與預積分的殘差

為了簡化表示，可以把

$$Ax = b$$

，裏面的A和b矩陣按行分成一塊一塊的來表示。

從第二部分開始。要擬合的測量值目標是，IMU的加速度計和陀螺儀的預積分。可以轉換成實際值與預測值的位姿差來約束。另外還要加上前後偏移接近的約束。與這有關的自變量是，前後幀的位姿，速度，偏移。

其實就是，前一幀基於IMU對後一幀的位置，姿態，速度，偏移，的預測值，與當前值的，誤差。

每次迭代之後，重新算誤差（需要人為寫出算誤差的進程）。都會用最新的偏移與之前的偏移的差，重新計算兩幀之間的

$$\alpha, \beta, \gamma$$

，因為先前預積分的時候已經算好了預積分的值關於偏移的導數。所以，這裏直接根據偏移差乘以導數，就能一下調節預積分的值。明明已經有了預積分關於偏移的雅克比了，為什麼這裏要這樣子把偏移單獨拿出來算新的預積分，而不是偏移和預積分一起直接優化呢？每迭代一次都要調整一次，為什麼不多優化幾次，最後再累加呢？因為當新的偏移與舊的偏移相差較大的時候，就不再使用之前預積分關於偏移的雅克比了，需要repropagate，重新計算新的預積分值和雅克比。但進程裏把，判斷偏移相差較大後重新傳播這一塊註釋了。預積分關於偏移的雅克比矩陣的計算為，狀態矢量轉移或相加，則對應的雅克比矩陣也轉移或相加，詳見《on-manifold詳細解讀》。

非線性迭代，每次迭代後，根據殘差和雅克比調整自變量的值，再根據調整後的自變量計算出新的殘差，再計算出新的雅克比，如此循環。這樣子，因為有殘差裏面的預積分關於偏移的導數，每次迭代後，調節自變量裏面的偏移的值後，再計算新的殘差時，就方便了。

$$
\begin{bmatrix}
-\frac{\partial P}{\partial b_a} & -\frac{\partial P}{\partial b_g} & R\left(q_w^{b_k}\right) & 0 & 0 & 0 & 0 \\
0 & \frac{\partial R}{\partial b_g} & \frac{\partial R}{\partial R} & 0 & 0 & 0 & 0 \\
-\frac{\partial V}{\partial b_a} & -\frac{\partial V}{\partial b_g} & 0 & 0 & R\left(q_w^{b_k}\right) & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 1 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
p_{b_0}^w \\ q_{b_0}^w \\ v_{b_0}^w \\ b_{a,b_0} \\ b_{g,b_0} \\ p_{b_1}^w \\ q_{b_1}^w \\ v_{b_1}^w \\ b_{a,b_1} \\ b_{g,b_1} \\ \vdots \\ p_{b_n}^w \\ q_{b_n}^w \\ v_{b_n}^w \\ b_{a,b_n} \\ b_{g,b_n}
\end{bmatrix}
=
\begin{bmatrix}
q_w^{b_k}\left(p_{b_1}^w - p_{b_0}^w + \frac{1}{2}g^w\Delta t^2 - v_{b_0}^w\Delta t\right) - \iint_{t\in[0,1]}\gamma_{b_i}^{b_k}\left(\hat{a}_i - b_{a_i}\right)dt^2 \\
2\left[\left(q_{b_1}^w\right)^{-1}\otimes q_{b_0}^w\otimes\int_{t\in[0,1]}\gamma_{b_i}^{b_k}\otimes\begin{bmatrix}0\\ \frac{1}{2}\left(\hat{\omega}_i - b_{g_i}\right)\end{bmatrix}dt\right]_{xyz} \\
q_w^{b_k}\left(v_{b_1}^w + g^w\Delta t - v_{b_0}^w\right) - \int_{t\in[0,1]}\gamma_{b_i}^{b_k}\left(\hat{a}_i - b_{a_i}\right)dt \\
b_{a,b_1} - b_{a,b_0} \\
b_{g,b_1} - b_{g,b_0} \\
\vdots
\end{bmatrix}
$$

3.3.3最小化重投影誤差

然後，第三部分，要擬合的測量值目標是特徵點在各幀圖像上的測量位置。可以轉換成重投影誤差來約束。與這有關的自變量是，該特徵點被觀察到的第一幀和被觀察到的另外一幀的位置和姿態，相機和IMU之間的相對位姿，特徵點在第一幀的深度。

其實就是，對每一個地圖點的預測的投影點與實際的投影點的誤差。

全展開殘差項可以表示為，

$$
\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} * \left(\pi_c^{-1}\left(\begin{bmatrix}\hat{u}_l^{c_j}\\ \hat{v}_l^{c_j}\end{bmatrix}\right) - \frac{q_b^c\left(q_w^{b_j}\left(q_{b_i}^w\left(q_c^b\frac{1}{\lambda_l}\pi_c^{-1}\left(\begin{bmatrix}u_l^{c_i}\\ v_l^{c_i}\end{bmatrix}\right)+p_c^b\right)+p_{b_i}^w\right)+p_w^{b_j}\right)+p_b^c}{\left\| q_b^c\left(q_w^{b_j}\left(q_{b_i}^w\left(q_c^b\frac{1}{\lambda_l}\pi_c^{-1}\left(\begin{bmatrix}u_l^{c_i}\\ v_l^{c_i}\end{bmatrix}\right)+p_c^b\right)+p_{b_i}^w\right)+p_w^{b_j}\right)+p_b^c \right\|} \right)
$$

其中，

$$
\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}
$$

是當前的原點到測量單位球面點的矢量

$$
a
$$

，在球面上的切向單位矢量。

$$
b_1 = \left\| a \times [1,0,0] \right\|
$$

$$
b_2 = \left\| a \times b_1 \right\|
$$

它與誤差矢量相乘，就相當於誤差矢量在這兩個切向矢量上的投影。因為

$$
a \cdot b = \|a\|\|b\|\cos\theta
$$

。

化簡後，可以表示為，

$$
P_l^{c_j} = q_b^c\left(q_w^{b_j}\left(q_{b_i}^w\left(q_c^b\frac{1}{\lambda_l}\pi_c^{-1}\left(\begin{bmatrix}u_l^{c_i}\\ v_l^{c_i}\end{bmatrix}\right)+p_c^b\right)+p_{b_i}^w\right)+p_w^{b_j}\right)+p_b^c
$$

$$residual = \left[\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} * \left(\pi_c^{-1} \left(\begin{bmatrix} \hat{u}_i^{c_j} \\ \hat{v}_i^{c_j} \end{bmatrix} \right) - \frac{P_i^{c_j}}{\left\| P_i^{c_j} \right\|} \right) \right]$$

$$\frac{\partial residual}{\partial P_i^{c_j}} \begin{bmatrix} \frac{\partial P_i^{c_j}}{\partial p_{b_i}^w} & \frac{\partial P_i^{c_j}}{\partial q_{b_i}^w} & \frac{\partial P_i^{c_j}}{\partial p_{b_j}^w} & \frac{\partial P_i^{c_j}}{\partial q_{b_j}^w} & \frac{\partial P_i^{c_j}}{\partial p_c^b} & \frac{\partial P_i^{c_j}}{\partial q_c^b} & \frac{\partial P_i^{c_j}}{\partial \lambda_i} \end{bmatrix} \begin{bmatrix} p_{b_i}^w \\ q_{b_i}^w \\ p_{b_j}^w \\ q_{b_j}^w \\ p_c^b \\ q_c^b \\ \lambda_i \end{bmatrix}$$

$$= \left[\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} * \left(\pi_c^{-1} \left(\begin{bmatrix} \hat{u}_i^{c_j} \\ \hat{v}_i^{c_j} \end{bmatrix} \right) - \frac{P_i^{c_j}}{\left\| P_i^{c_j} \right\|} \right) \right]$$

其中，鏈式求導，一步步傳導下去。

$$*(\text{cnblogs.com/ilekoaiq}) \frac{\partial \begin{bmatrix} \frac{x_1}{\sqrt{x_1^2+x_2^2+x_3^2}} & \frac{x_2}{\sqrt{x_1^2+x_2^2+x_3^2}} & \frac{x_3}{\sqrt{x_1^2+x_2^2+x_3^2}} \end{bmatrix}^T}{\partial \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}}$$

$$\begin{bmatrix} -x_1^2\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} & -x_1x_2\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} & -x_1x_3\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} \\ -x_1x_2\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} & \left(x_1^2+x_2^2+x_3^2\right)^{-\frac{1}{2}}-x_2^2\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} & -x_2x_3\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} \\ -x_1x_3\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} & -x_2x_3\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} & \left(x_1^2+x_2^2+x_3^2\right)^{-\frac{1}{2}}-x_3^2\left(x_1^2+x_2^2+x_3^2\right)^{-\frac{3}{2}} \end{bmatrix}$$

$$\frac{\partial P_i^{c_j}}{\partial p_{b_i}^w} = \frac{\partial \left(q_b^c q_w^{b_j} p_{b_i}^w \right)}{\partial p_{b_i}^w} = R\left(q_b^c q_w^{b_j} \right)$$

$$\frac{P_i^{c_j}}{q_{b_i}^w} = \frac{\partial \left(q_b^c q_w^{b_j} q_{b_i}^w \left(q_c^b \frac{1}{\lambda_i} \pi_c^{-1} \left(\begin{bmatrix} u_i^{c_i} \\ v_i^{c_i} \end{bmatrix} \right) + p_c^b \right) \right)}{\partial q_{b_i}^w} = \frac{\partial \left(q_b^c q_w^{b_j} q_{b_i}^w P_i^{b_i} \right)}{\partial q_{b_i}^w} = R\left(q_b^c q_w^{b_j} \right) R\left(q_{b_i}^w \right) \left(-\left(P_i^{b_i} \right)^\Lambda \right)$$

$$\frac{\partial P_i^{c_j}}{\partial p_{b_j}^w} = \frac{\partial \left(q_b^c p_w^{b_j} \right)}{\partial p_{b_j}^w} = \frac{\partial \left(R\left(q_b^c \right) \left(-\left(R\left(q_{b_j}^w \right) \right)^T p_{b_j}^w \right) \right)}{\partial p_{b_j}^w} = -R\left(q_b^c q_w^{b_j} \right)$$

$$\frac{\partial P_i^{c_j}}{\partial q_{b_j}^w} = \frac{\partial \left(q_b^c q_w^{b_j} q_{b_j}^w P_i^{b_i} \right)}{\partial q_{b_j}^w} = \frac{\partial \left(R\left(q_b^c \right) \left(R\left(q_{b_j}^w \right) \right)^T R\left(q_{b_i}^w \right) P_i^{b_i} \right)}{\partial q_{b_j}^w} = R\left(q_b^c \right) \left(P_i^{b_j} \right)^\Lambda$$

$$\frac{\partial P_i^{c_j}}{\partial p_c^b} = \frac{\partial \left(q_b^c q_w^{b_j} q_{b_i}^w p_c^b + p_b^c \right)}{\partial p_c^b} = R\left(q_b^c q_w^{b_j} q_{b_i}^w \right) - \left(R\left(q_c^b \right) \right)^T = R\left(q_c^b \right) \left(R\left(q_w^{b_j} q_{b_i}^w \right) - I \right)$$

$$= \frac{\partial \left(q_b^c q_w^{b_j} q_{b_i}^w q_c^b P_i^{c_i} \right)}{\partial q_c^b} = \frac{\partial \left(R\left(q_c^b \right)^T R\left(q_w^{b_j} q_{b_i}^w \right) R\left(q_c^b \right) P_i^{c_i} \right)}{\partial q_c^b} = \left(P_i^{c_j} \right)^\Lambda - R\left(q_c^b \right)^T R\left(q_w^{b_j} q_{b_i}^w \right) R\left(q_c^b \right) \left(P_i^{c_i} \right.$$

$$\frac{\partial P_l^{c_j}}{\partial \lambda_l}=\frac{\partial\left(q_b^c q_w^{b_j} q_{b_i}^w q_c^b \frac{1}{\lambda_l} p_l^{c_i}\right)}{\partial \lambda_l}=-\lambda_l^{-2} R\left(q_b^c q_w^{b_j} q_{b_i}^w q_c^b\right) p_l^{c_i}$$

因為在上述的雅克比矩陣裏面，都有自變量

$$\Delta x$$

了，所以它是非線性的，只能通過優化的方法一步步迭代。

3.3.4邊緣化

H*x=JT*residuals，H=JT*J。

基於舒爾補來邊緣化。

用奇異值分解的技巧來得到新的雅克比矩陣，作為先驗雅克比。由b反過去算殘差，作為先驗值，在下一次優化時使用。

其實，與《on-mainifold》裏面的舒爾補的方法，本質上是一樣的。

關於邊緣化，還可以再參考http://blog.csdn.net/heyijia0327/article/details/52822104。

3.3.5 迴環優化

如果窗口裏存在有迴環幀的對應幀的話，則先找到對應幀與迴環幀的匹配點id和位置，特徵點匹配用的是在一定範圍內匹配描述子。特徵點匹配完之後，然後只保留這些匹配上的id和特徵點位置，給front_pose，id用對應幀的id，特徵點位置用迴環幀的位置。其實，就是把對應幀的id賦給匹配上的迴環幀的特徵點。然後給front_pose.loop_pose對應幀的姿態作為初值，然後用窗口裏面的點的重投影誤差來優化front_pose.loop_pose，點的逆深度，點被觀察到的第一幀的位姿。

窗口優化之後，比較對應幀與front_pose.loop_pose的yaw角是否相差太大，如果相對yaw角大於30度或相對位置大於20的話，就認為這是錯誤的迴環，刪除這個迴環。因為這時候，這個對應幀還在窗口裏面，還沒有滑出去迴環閉環。所以，這樣也可以及時把DBow找出來的錯誤的迴環，刪掉。

其實，就是把迴環幀放到窗口裏面來優化，優化出迴環幀的位姿，然後再算出迴環幀和對應幀的相對位姿。然後把這個相對位姿，作為後面的迴環4自由度優化的測量值。具體是，優化出迴環幀在窗口裏面的位置後，算出對應幀相對於迴環幀的位姿。

新的關鍵幀在加入關鍵幀數據庫的時候，作為對應幀，在關鍵幀數據庫裏面去找回環幀。如果用DBOW找到迴環幀，使用cur_kf->findConnectionWithOldFrame(old_kf, measurements_old, measurements_old_norm, PnP_T_old, PnP_R_old, m_camera);。先用searchByDes 去找匹配點，然後用cv::findFundamentalMat(un_measurements, un_measurements_old, cv::FM_RANSAC, 5.0, 0.99, status);篩選匹配點，再用對應幀自帶的地圖點結合solvePnPRansac，得到迴環幀的位姿PnP_T_old。PnP_T_old作為迴環幀位姿loop_pose的初值，結合繼承過來的匹配點measurements_old_norm，再傳回當前窗口中優化，problem.AddResidualBlock(f, loss_function, para_Pose[start], retrive_data_vector[k].loop_pose, para_Ex_Pose[0], para_Feature[feature_index]);，得到優化後的loop_pose。如果優化後的loop_pose相對對應幀的位姿relative_yaw小於閾值，則認為迴環正確。而優化後的loop_pose相對迴環幀原來的位姿，為relocalize_r和relocalize_t。

在vins-mono的新版本中，添加加了relocalize_r、relocalize_t，其作用是，在大回環起作用的間隙，用relocalize_r、relocalize_t來對位姿進行及時的修正，以更好地保證輸出位姿的準確性，以及關鍵幀輸入到關鍵幀數據庫裏時的位姿的準確性。因為以前是要等迴環幀的對應幀滑出窗口，大回環優化後，才對這兩個位姿進行校正的，而現在可以更及時地修正這些位姿，如果有地方想要最快速地得到準確的位姿的話。new KeyFrame(estimator.Headers[WINDOW_SIZE - 2], vio_T_w_i, vio_R_w_i, cur_T, cur_R, KeyFrame_image, pattern_file);這裏面的vio_T_w_i是迴環優化時的計算前後關鍵幀的相對位置時用的，所以用的還是窗口中的位姿。而cur_T，也就是T_w_i，是表示校正後的位姿，在輸入的時候，會根據relocalize_r校正，大回環優化後，還再校正關鍵幀數據庫裏的所有的關鍵幀位姿。relocalize_r、relocalize_t也是很巧妙的方法，因為它們是根據迴環幀和對應幀的圖像的相對位姿算出來的迴環偏移，其實大回環優化之後得到的迴環偏移correct_t，和這個迴環偏移relocalize_t，應該相差不大。大回環主要是優化環中間的那些關鍵幀的位姿，大回環的主要目的在於修正關鍵幀圖PoseGraph。所以，relocalize_r、relocalize_t一開始就很接近最終大回環優化後的迴環偏移correct_t。relocalize_t是把迴環幀放到當前窗口下優化，算出來的與原來的偏移；correct_t是大回環優化後的對應幀位姿與它原來的位姿的偏移。correct_t準確但計算慢，relocalize_t計算快速且較準確。

4.迴環檢測

這裏的迴環檢測，是每3個關鍵幀檢測一幀，相當於是跳兩幀。這跟迴環檢測的速度，和實際關鍵幀生成的速度，對比有關。因為迴環檢測的速度總是慢於關鍵幀生成的速度，所以為了保持迴環檢測的關鍵幀不落後於時間，只能跳幀檢測。ORBSLAM裏面也是這樣，但ORBSLAM裏面的迴環檢測判斷標準是，一段時間內的關鍵幀都能匹配上回環，所以ORBSLAM的策略是拿一段時間的關鍵幀來進行檢測。ORBSLAM的迴環檢測進程，Sleep一段時間，在Sleep的這段時間，收集關鍵幀，然後開始工作，只針對收集的這些關鍵幀。工作時，不收集新的關鍵幀，都跳過。處理完這些關鍵幀後，又Sleep。

因為迴環檢測的速度總是慢於關鍵幀生成的速度，所以為了保持迴環檢測的關鍵幀不落後於時間，只能跳幀檢測。

窗口裏，每3個關鍵幀，送一個到關鍵幀數據庫裏。關鍵幀數據庫裏面的每一幀都跟之前的進行檢測，看是否有迴環。迴環檢測用的是DBow，這個關鍵幀上面的用FAST找出來的新的特徵點和它在之前被光流跟蹤到的特徵點，提取描述子，與歷史描述子匹配。這幀的描述子以及對應的特徵點，跟歷史上的描述子以及對應的特徵點進行匹配，得到匹配上的特徵點。

如果有迴環，就等這個對應幀從窗口裏滑出，再回環閉環。

找出迴環最早開始的幀，然後把這幀的位姿設為固定。

迴環閉環的約束條件是，與優化前的相對位姿不能差太大，每幀與前5幀的相對位姿。迴環幀和對應幀的相對位姿的權重是5。迴環閉環裏面，優化的都是4自由度的姿態，迴環幀與閉環幀，每幀與它的前5幀的相對姿態。

優化完後，在最後一個對應幀那裏，再把世界座標系也轉換一下，然後把剩下的關鍵幀都轉換一下。

迴環閉環優化部分的測量值是，迴環幀與對應幀的基於圖像算出的相對4自由度姿態，就是relative_t和relative_yaw，就是迴環幀的loop_pose在窗口中優化後，相對窗口中的對應幀的位姿。每幀圖像與它前5幀的相對4自由度姿態。約束為，預測值和測量值之間的差。與此有關的自變量是，每幀圖像的4自由度位姿。就像一個項鍊一樣，一串地拉過來。

注意的是，迴環幀與對應幀的基於圖像算出的相對4自由度姿態的權重是5，為了平衡每幀圖像與它前5幀的相對4自由度姿態。對應幀的前後對它的拉力要相同。

假設是第0幀和第m幀迴環閉環了。

$$
residual = \begin{bmatrix}
q(\psi_0)^{-1}\left(p_5^w - p_0^w\right) - \hat{p}_{0,5}^0 \\
\psi_5 - \psi_0 - \hat{\psi}_{0,5} \\
q(\psi_1)^{-1}\left(p_5^w - p_1^w\right) - \hat{p}_{1,5}^1 \\
\psi_5 - \psi_1 - \hat{\psi}_{1,5} \\
q(\psi_2)^{-1}\left(p_5^w - p_2^w\right) - \hat{p}_{2,5}^2 \\
\psi_5 - \psi_2 - \hat{\psi}_{2,5} \\
q(\psi_3)^{-1}\left(p_5^w - p_3^w\right) - \hat{p}_{3,5}^3 \\
\psi_5 - \psi_3 - \hat{\psi}_{3,5} \\
q(\psi_4)^{-1}\left(p_5^w - p_4^w\right) - \hat{p}_{4,5}^4 \\
\psi_5 - \psi_4 - \hat{\psi}_{4,5} \\
\text{:(cnblogs.com/ilekoaiq)} \\
5 * \left(q(\psi_0)^{-1}\left(p_m^w - p_0^w\right) - \hat{p}_{L,0,m}^0\right) \\
5 * \left(\psi_m - \psi_0 - \hat{\psi}_{L,0,m}\right)
\end{bmatrix}
$$

在這裏，究竟是使用

$$q(\psi_1)^{-1}$$

還是

$$q_1^0(\psi_1)^{-1}$$

，我認為是一樣的。前者是把下一幀相機的原點轉換到上一幀相機的yaw角的水平座標系下，以此為測量值，預測值與測量值的殘差，關於yaw角求導。後者是把把下一幀的相機的原點轉換到上一幀的座標系下，預測值與測量值的殘差，關於yaw角求導。但這裏的yaw角都是指在水平世界座標系裏面的yaw角。最終優化的結果應該是一樣的。像這

種，只是把殘差從一個直角座標系轉換到另外一個直角座標系，結果應該是一樣的。但是，如果是另外一些轉換，那結果就可能不一樣了，比如非直角座標系，各維度的有不同的縮放尺度，或者維度變換，那樣子的話，優化結果就會不一樣。

這裏使用進程裏面的表示，

$$q_1^0(\psi_1)^{-1}$$

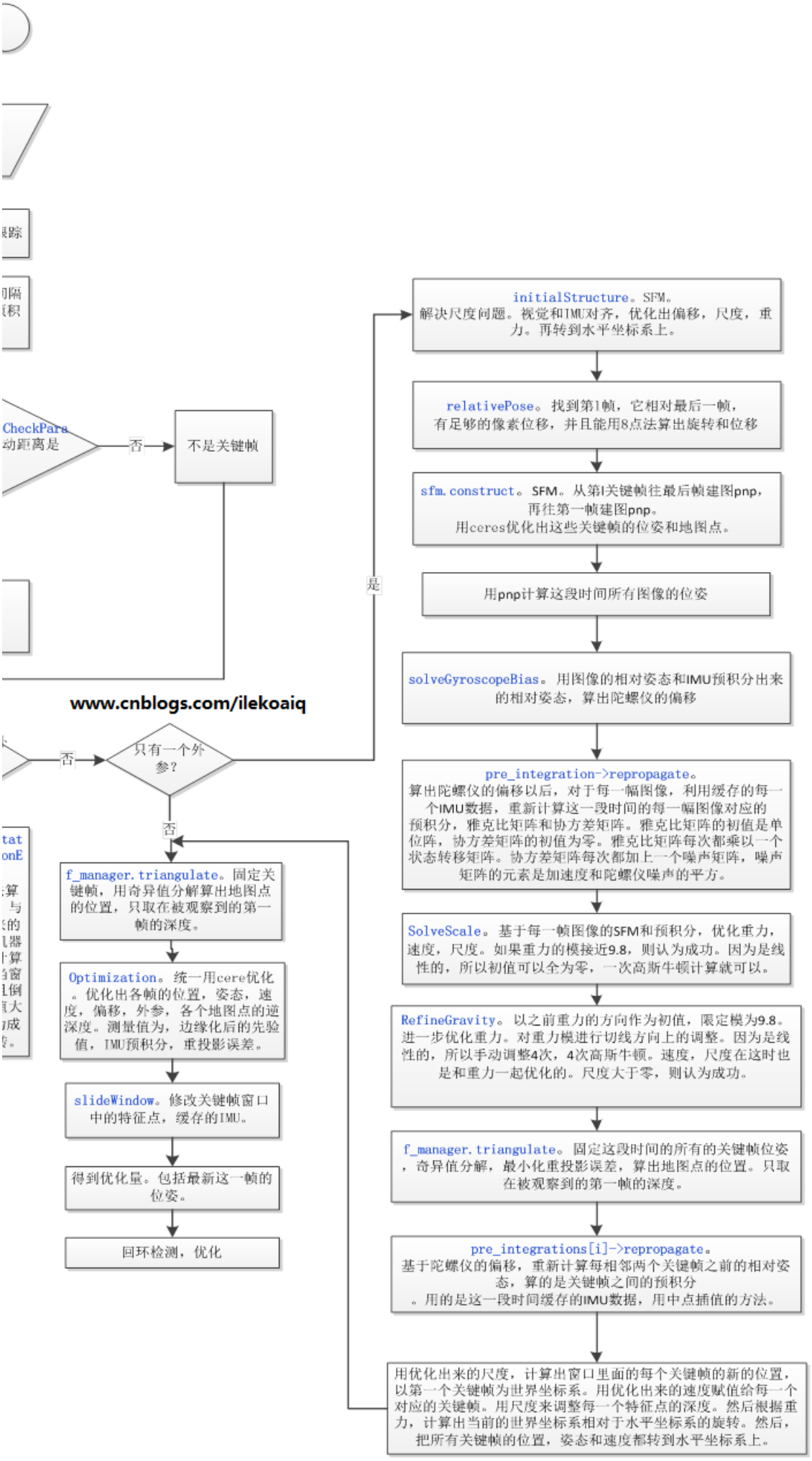
。進程裏面沒有求雅克比，而是用自動求導的方法。

$$J\begin{bmatrix} p_0^w \\ \psi_0 \\ p_1^w \\ \psi_1 \\ p_2^w \\ \psi_2 \\ p_3^w \\ \psi_3 \\ p_4^w \\ \psi_4 \\ p_5^w \\ \psi_5 \\ \vdots \\ p_m^w \\ \psi_m \end{bmatrix} = \begin{bmatrix} \left(q_0^0(\psi_0)\right)^{-1}\left(p_5^w - p_0^w\right) - \hat{p}_{0,5}^0 \\ \psi_5 - \psi_0 - \hat{\psi}_{0,5} \\ \left(q_1^0(\psi_1)\right)^{-1}\left(p_5^w - p_1^w\right) - \hat{p}_{1,5}^1 \\ \psi_5 - \psi_1 - \hat{\psi}_{1,5} \\ \left(q_2^0(\psi_2)\right)^{-1}\left(p_5^w - p_2^w\right) - \hat{p}_{2,5}^2 \\ \psi_5 - \psi_2 - \hat{\psi}_{2,5} \\ \left(q_3^0(\psi_3)\right)^{-1}\left(p_5^w - p_3^w\right) - \hat{p}_{3,5}^3 \\ \psi_5 - \psi_3 - \hat{\psi}_{3,5} \\ \left(q_4^0(\psi_4)\right)^{-1}\left(p_5^w - p_4^w\right) - \hat{p}_{4,5}^4 \\ \psi_5 - \psi_4 - \hat{\psi}_{4,5} \\ \vdots\text{(cnblogs.com/ilekoaiq)} \\ 5*\left(\left(q_{L,m}^0(\psi_0)\right)^{-1}\left(p_m^w - p_0^w\right) - \hat{p}_{L,0,m}^0\right) \\ 5*\left(\psi_m - \psi_0 - \hat{\psi}_{L,0,m}\right) \end{bmatrix}$$

迴環檢測之後的處理，非常有創新點。一般的，迴環檢測之後，是要根據迴環檢測結果，把窗口關鍵幀的位姿都轉換掉。但實際上，是可以不管窗口裏面的關鍵幀的，可以認為窗口裏面的關鍵幀的世界座標系相對真實的世界座標系發生了偏移和轉換。只需要在輸出位姿的時候，乘以這個轉換就可以了，窗口裏面的位姿還是原來的位姿。這樣子，就可以避免對窗口的位姿進行大幅度調整，也不需要重新計算雅克比，協方差矩陣。輸出的結果是應用所需要的，修正過的結果。

為了輸出最新的位姿，需要把新緩存進來的IMU數據也都積分起來。更新實時的最新的位姿。然後，每新進來一個IMU數據，就在之前的基礎上預測一下。因為IMU數據都是在載體座標系的，所以，可以先在窗口的座標系裏積分，然後最後再轉換到真實的世界座標系，也可以直接把座標轉換到真實的世界座標系，然後再積分。最後都是得到在真實的世界座標系裏面的位姿，然後每新進來一個IMU數據，就在之前的基礎上預測一下。這兩種方法是一樣的。

5.流程圖


CheckPara
动距离是
否
不是关键帧
www.cnblogs.com/ilekoaiq
否
只有一个外参?
是
否
f_manager.triangulate。固定关键帧，用奇异值分解算出地图点的位置，只取在被观察到的第一帧的深度。
Optimization。统一用cere优化。优化出各帧的位置，姿态，速度，偏移，外参，各个地图点的逆深度。测量值为，边缘化后的先验值，IMU预积分，重投影误差。
slideWindow。修改关键帧窗口中的特征点，缓存的IMU。
得到优化量。包括最新这一帧的位姿。
回环检测，优化
initialStructure。SFM。
解决尺度问题。视觉和IMU对齐，优化出偏移，尺度，重力。再转到水平坐标系上。
relativePose。找到第l帧，它相对最后一帧，有足够的像素位移，并且能用8点法算出旋转和位移
sfm.construct。SFM。从第l关键帧往最后帧建图pnp，再往第一帧建图pnp。
用ceres优化出这些关键帧的位姿和地图点。
用pnp计算这段时间所有图像的位姿
solveGyroscopeBias。用图像的相对姿态和IMU预积分出来的相对姿态，算出陀螺仪的偏移
pre_integration->repropagate。
算出陀螺仪的偏移以后，对于每一幅图像，利用缓存的每一个IMU数据，重新计算这一段时间的每一幅图像对应的预积分，雅克比矩阵和协方差矩阵。雅克比矩阵的初值是单位阵，协方差矩阵的初值为零。雅克比矩阵每次都乘以一个状态转移矩阵。协方差矩阵每次都加上一个噪声矩阵，噪声矩阵的元素是加速度和陀螺仪噪声的平方。
SolveScale。基于每一帧图像的SFM和预积分，优化重力，速度，尺度。如果重力的模接近9.8，则认为成功。因为是线性的，所以初值可以全为零，一次高斯牛顿计算就可以。
RefineGravity。以之前重力的方向作为初值，限定模为9.8。进一步优化重力。对重力模进行切线方向上的调整。因为是线性的，所以手动调整4次，4次高斯牛顿。速度，尺度在这时也是和重力一起优化的。尺度大于零，则认为成功。
f_manager.triangulate。固定这段时间的所有的关键帧位姿，奇异值分解，最小化重投影误差，算出地图点的位置。只取在被观察到的第一帧的深度。
pre_integrations[i]->repropagate。
基于陀螺仪的偏移，重新计算每相邻两个关键帧之前的相对姿态，算的是关键帧之间的预积分
。用的是这一段时间缓存的IMU数据，用中点插值的方法。
用优化出来的尺度，计算出窗口里面的每个关键帧的新的位置，以第一个关键帧为世界坐标系。用优化出来的速度赋值给每一个对应的关键帧。用尺度来调整每一个特征点的深度。然后根据重力，计算出当前的世界坐标系相对于水平坐标系的旋转。然后，把所有关键帧的位置，姿态和速度都转到水平坐标系上。