# プログラムの概要

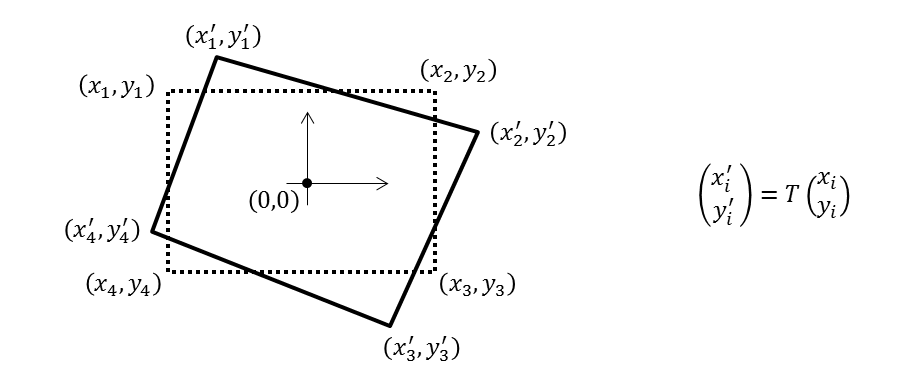
TILT.mでの流れを以下で説明する。

1. Branch-and-boundにより、大まかに変換行列を推定する。  
   論文のSection 3.2の中の**”Branch-and-Bound Scheme”**のparagraphの中で、branch and boundについて説明している。  
   まず、回転行列（[-30°,+30°]の範囲で11サンプル）を適用し、f値が最小となる回転角度を探す。この時の変換行列を***initial\_tfm\_matrix***に代入する。  
   次に、上で得た変換行列をベースとして、x軸方向のskew行列（[-1, 1]の範囲で11サンプル）を適用し、f値が最小となるskewを探す。この時の変換行列を***initial\_tfm\_matrix***に代入する。  
   最後に、上で得た変換行列をベースとして、y軸方向のskew行列（[-1, 1]の範囲で11サンプル）を適用し、f値が最小となるskewを探す。この時の変換行列を***initial\_tfm\_matrix***に代入する。
2. 低い解像度から2倍、4倍と解像度を上げながら、***initial\_tfm\_matrix***からスタートしてf値を最小とするように変換行列を更新していく。  
   詳細は、tilt\_kernel.m。

# Tilt\_kernel.m

# Tfm2para.m

下図のように、画像の四隅の座標の変換後の座標を8x1のパラメータ値として使用する。パラメータ値の最適化では、直接行列を扱えないので、こうやってベクトルに変換するわけだ。



# Para2tfm.m

Tfm2param.mとは逆に、変換後の四隅の座標を、変換行列に変換する。もとの長方形の四隅の座標は分かっているし、変換後の四隅の座標も分かっているので、以下の式が成り立つ。

つまり、以下の8個の式が成り立つわけだ。

で整理すると、以下のように行列の式で表せる。よって、左辺の8x8行列を右辺にもっていけば、を計算できるので、変換行列を得ることができる。

# Jacobi.m

論文によると、変換パラメータに対する画像データのヤコビ行列を計算する。まず、ヤコビ行列とは、一般に、

この問題では、は、画像の各ピクセルの輝度値、は、変換パラメータに相当する。つまり、

しかし、画像を変換パラメータで微分するというのは難しいので、以下のように、X、Y方向の微分を使って求める。

ここで、とは、**Sobelフィルタ**によって簡単に計算できるので、とだけ計算すれば良い。

さて、変換行列は、

なので、

よって、

後は、これらを使って、, を解いていく。

まず、は、変換後の座標が、変換パラメータによってどのように変化するか、ということだ。つまり、

同様に、

よって、

これは、Matlabコードの、以下の部分（52行目）に該当する。

dIdH(:, :, 1)=du.\*X0./N;

ここで、duは、Nはに相当する。

次に、は、変換後の座標が、変換パラメータによってどのように変化するか、ということだ。つまり、

同様に、

よって、

これは、Matlabコードの、以下の部分（53行目）に該当する。

dIdH(:, :, 2)=du.\*Y0./N;

ここで、duは、Nはに相当する。

さらに、は、変換後の座標が、変換パラメータによってどのように変化するか、ということだ。つまり、

同様に、

よって、

これは、Matlabコードの、以下の部分（54行目）に該当する。

dIdH(:, :, 3)=du./N;

ここで、duは、Nはに相当する。

による微分も同様に計算できる。については、

同様に、

よって、

これは、Matlabコードの、以下の部分（58行目）に該当する。

dIdH(:, :, 7)=du.\*(-N1./N.^2.\*X0)+dv.\*(-N2./N.^2.\*X0);

ここで、duは、dvは、N1は、N2は、Nはに相当する。

についても同様に、

また、

よって、

これは、Matlabコードの、以下の部分（59行目）に該当する。

dIdH(:, :, 8)=du.\*(-N1./N.^2.\*Y0)+dv.\*(-N2./N.^2.\*Y0);

ここで、duは、dvは、N1は、N2は、Nはに相当する。

以上で、が求まった。Matlabコードでは、dIdHに相当する。本来ならこれで終わりなのだが、このMatlabコードでは、変換パラメータとして、変換行列の要素ではなく、画像の四隅の変換後の座標を使っているため、その値で微分する必要がある。つまり、

ここで、は、返還後の四隅の座標だ。残念ながら、は計算できないので、を計算して、その逆行列を使用する。つまり、

ちなみに、もヤコビ行列だ。なぜなら、もも多次元ベクトルだから。つまり

では、各要素について、計算していこう。まず、もとの四隅の座標をとすると、返還後の座標は、

よって、

従って、

についても同様に計算できる。このあたりは、Matlabコードの、以下の部分（68～77行目）に該当する。

dPdH(2\*i-1, 1)=X(i)/N(i);

dPdH(2\*i-1, 2)=Y(i)/N(i);

dPdH(2\*i-1, 3)=1/N(i);

dPdH(2\*i-1, 7)=-N1(i)/N(i)^2\*X(i);

dPdH(2\*i-2, 8)=-N1(i)/N(i)^2\*Y(i);

dPdH(2\*i, 4)=X(i)/N(i);

dPdH(2\*i, 5)=Y(i)/N(i);

dPdH(2\*i, 6)=1/N(i);

dPdH(2\*i, 7)=-N2(i)/N(i)^2\*X(i);

dPdH(2\*i, 8)=-N2(i)/N(i)^2\*Y(i);

ここで、N1は、N2は、Nはに相当する。

結局、以下のようにが計算できる。

後は、先に述べた通り、以下のようにしてヤコビ行列を計算するだけだ。

# Inner\_IALM\_constraints.m

論文のAlgorithm 2を見ると、

ここで、は、行列の各要素について、値がより大きい要素のみを残して、を引く。論文の式(12)で定義されている。

**※厳密には、この式はおかしい。これでは、の場合も値は0にならないよね。**

以下に、簡単な例を示す。論文中では、これを***soft-thresholding***あるいは***shrinkage operator***と呼んでいる。

これは、Matlabコードの101行目に相当する。

A=U\*((S>1/mu).\*(S-1/mu))\*V’;

ただし、Sは、Aはを表す。ちなみに、このコードではが負の値の場合を考慮していないが、SVD分解したら固有値は必ず正なので問題ない。

また、Algorithm 2の次のステップでは、

ここで、は、ラグランジュ乗数行列。

これは、Matlabコードの104行目に相当する。

E=(temp\_2>lambda/mu).\*(temp\_2-lambda/mu)+(temp\_2<-lambda/mu).\*(temp\_2+lambda/mu);

ただし、temp\_2はを表す。こちらは、値が負の場合も考慮していることが分かる。

Algorithm 2の次のステップは、

これは、Matlabコードの109行目に相当する

delta\_tau=pinv\_J\_vec\*temp\_3;

ただし、temp\_3はを表す。また、temp\_3は、Matlabコードの106行目から108行目で計算している。

temp\_3=A+E-Dotau-Y\_1/mu;

temp\_3=reshape(temp\_3, m\*n, 1);

temp\_3=[temp\_3; -Y\_2/mu];

ただし、Aは、Dotauはを表す。

また、Y\_2については、論文のSection 3.2の**Constraints on the Transformations**のparagraphに説明されている。まず、2つの制約について説明している。

* *Constraints on Translation*  
  画像の中心座標は、変換後も変わらない。
* *Constraints on Scale*  
  画像の面積、および、縦横比は、変換後も変わらない。

なぜ、との積の形で制約を表す必要があるか？それは、ヤコビ行列などと合わせてラグランジュの未定乗数法で解くため、同じ形式にする必要があるから。

さらに、詳細は**Appendix A**にて説明されている。まず1つ目の制約について、中心座標をとすると、

よって、

さて、

とすると、

となり、制約を式で表せる。

**※このあたり、理解が少し怪しい？**

2つ目の制約については、実際には縦横比ではなく、対角線の比を使っている。つまり、2つの対角線ベクトルをとすると、その面積は、

で計算できる。実際には、以下のようにもっと簡単に計算できる。