PLMP - Point-Line Minimal Problems in Complete Multi-View Visibility Suplementary Material

We present additional details about our notation, explain basic concepts from algebraic geometry used, and implementation detail including commented implementations of typical examples of dominance checking and symbolic as well as numeric degree computation.

1. Notation and Concepts

We use nomenclature from [**?**] for basic concepts in geometry of computer vision. See [**?**] for the fundamentals of algebraic geometry, including Gröbner bases.

$\mathrm{SO}(3)$ stands for the special orthogonal group, *i.e.* rotations, defined algebraically as 3×3 matrices R such that $RR^\intercal = I$ and $\det(R) = 1$. We note that the dimension of $\mathrm{SO}(3)$ is three.

For u in $\mathbb{R}^3$, the skew-symmetric matrix $[u]_\times$ in $\mathbb{R}^{3\times3}$ represents the cross product with u in $\mathbb{R}^3$, *i.e.* $[u]_\times\, v = u \times v$ for all v in $\mathbb{R}^3$.

Points in space are in the projective space $\mathbb{P}^3$, whereas image points are in $\mathbb{P}^2$. So points are represented by homogeneous coordinates.

We sometimes refer to the concept of algebraic varieties. A *projective variety* is the common zero set of a system of homogeneous polynomial equations. If the polynomials are defined in $N+1$ homogeneous variables, the projective variety lives in $\mathbb{P}^N$. Similarly, an *affine variety* is the common zero set of a system of polynomial equations which are not necessarily homogeneous. If the polynomials are defined in N variables over the ground field $\mathbb{F}$, the affine variety lives in $\mathbb{F}^N$. For any subset S of either $\mathbb{P}^N$ or $\mathbb{F}^N$, the *Zariski closure* $\overline{S}$ of S is the smallest projective resp. affine variety containing S.

The Grassmannians $\mathbb{G}_{1,3}$ and $\mathbb{G}_{1,2}$, which are the sets of all lines in $\mathbb{P}^3$ and $\mathbb{P}^2$, respectively, are examples of smooth projective varieties. The Grassmannian $\mathbb{G}_{1,2}$ of lines in $\mathbb{P}^2$ is isomorphic to $\mathbb{P}^2$ as every line in the projective plane is uniquely defined by a single linear equation with three homogeneous coordinates as coefficients. The Grassmannian $\mathbb{G}_{1,3}$ of lines in $\mathbb{P}^3$ can be seen as a projective variety via its embedding into $\mathbb{P}^5$ defined by its *Plücker coordinates*:

A line in $\mathbb{P}^3$ is defined by two linear equations. We write the homogeneous coefficients of these two equations as the two rows of a 2×4 matrix $\left[\begin{smallmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \end{smallmatrix}\right]$. The six maximal minors of this matrix form the Plücker coordinates of the line in $\mathbb{P}^3$: $p_{ij} = a_i b_j - a_j b_i$. Every line in $\mathbb{P}^3$ is uniquely defined by its six Plücker coordinates. Moreover, the Plücker coordinates of a line in $\mathbb{P}^3$ satisfy the equation $p_{01}p_{23} - p_{02}p_{13} + p_{03}p_{12} = 0$. In addition, every projective tuple $(p_{01} : p_{02} : p_{03} : p_{12} : p_{13} : p_{23}) \in \mathbb{P}^5$ satisfying the equation $p_{01}p_{23} - p_{02}p_{13} + p_{03}p_{12} = 0$ is the Plücker coordinates of a line in $\mathbb{P}^3$.

This shows that the Grassmannian $\mathbb{G}_{1,3}$ is the zero set of the polynomial equation $p_{01}p_{23} - p_{02}p_{13} + p_{03}p_{12} = 0$ in $\mathbb{P}^5$. Furthermore, the Plücker coordinates of a line in $\mathbb{P}^3$ serve as homogeneous coordinates defining the line. So we can represent both points and lines in $\mathbb{P}^2$ as well as $\mathbb{P}^3$ by homogeneous coordinates.

A variety X is said to be *irreducible* if it is not the union of two proper subvarieties, *i.e.* if it cannot be written as $X = X_1 \cup X_2$ where X_1 and X_2 are non-empty subvarieties of X which are not equal to X. For instance, Grassmannians are irreducible.

Throughout our article we consider an arbitrary ground field $\mathbb{F}$ unless explicitly specified. For different purposes we use different fields. For instance, the minimal problems we study are clearly defined over the field $\mathbb{R}$ of real numbers. When setting up systems of polynomial equations, coefficients originate from the field $\mathbb{Q}$ of rational numbers. Solutions of the equations are in the field $\mathbb{C}$ of complex numbers. We carry out symbolic computations in a finite field $\mathbb{Z}_p$ for a prime p for the sake of exactness and computational efficiency. Numerical algorithms use floating point to approximate complex numbers.

2. Implementation Details

In this section, we give describe the Macaulay2 [**?**] implementation of our computations through a series of guided examples. Each subsection provides commentary for a dedicated Macaulay2 file:

2.1 `"example-2111_1-unrolled.m2"`

2.2 `"example-2111_1.m2"`

2.3 `"example-2111_1-jacobian.m2"`

2.4 `"example-2111_1-numerical.m2"`

Besides these examples and setup code providing the core functionality, there are several scripts which automate the computations reported in the main text.

2.1. Computing the degree symbolically "from scratch"

We begin by setting a ground field `FF` to be the finite field $\mathbb{Z}_{10007}$:

```
FF = ZZ/10007
```

We define point data in $\mathbb{P}^2$ in three views over $\mathbb{Z}_{10007}$ consistent with the problem "2111_1"). This is represented as a list of three matrices in Macaulay2, which can be viewed in an output buffer using the `netList` command:

```
i2 : P = {matrix{{-2639,-4936,1789},
                 {2653,-591,-643},
                 {1,1,1}},
          matrix{{-3868,-1776,3174},
                 {3669,-4143,-1982},
                 {1,1,1}},
          matrix{{-1889,-1604,4629},
                  {-473,-4513,-4210},
                  {1,1,1}}};
i3 : netList P

     +---------------------+
o3 = || -2639 -4936 1789 | |
     || 2653  -591  -643 | |
     || 1     1     1    | |
     +---------------------+
     || -3868 -1776 3174  ||
     || 3669  -4143 -1982 ||
     || 1     1     1     ||
     +---------------------+
     || -1889 -1604 4629  ||
     || -473  -4513 -4210 ||
     || 1     1     1     ||
     +---------------------+
```

The first two columns of each matrix in `P` were randomly generated, while the last column is (projectively) a random $\mathbb{Z}_{10007}$-linear combination of the first two. This reflects the dependence of the third point on the first two in each view.

In general, we encode point-line problems by specifying the number of visible lines, the number of ghost lines, and, for each point, a list of lines passing through it. For problem "2111_1" this is accomplished as follows:

```
D = (3,2,{{1,2},{1,3},{1,4}})
nLines = D#0 + D#1
lineIncidences = D#2
```

The list `L` contains random line matrices (homogeneous coordinates of lines as in rows) consistent with `P` and the encoding `D`:

```
i8 : netList L

     +---------------------+
o8 = || 107   4376  3187  ||
     || 3897  -4638 2998  ||
     || -1009 2785  -4328 ||
     || 2883  -1540 1031  ||
     || -1517 -713  3879  ||
     +---------------------+
     || 3783  -1437 275   ||
     || 2926  -687  -1310 ||
     || -2582 -2466 1236  ||
     || -2721 -1085 -1127 ||
     || 3072  4259  1727  ||
     +---------------------+
     || -1563 -4868 -1852 ||
     || -968  -960  -1036 ||
     || -3676 382   1454  ||
     || 1795  2177  -4909 ||
     || 2696  -1409 1206  ||
     +---------------------+
```

We may verify that the first row of each matrix in `L` represents a free line, that the second row gives a line through all points, the third rows gives a "pinned line" through the first point, and the fourth and fifth rows give ghost lines through the second and third points, respectively.

```
i9 : netList apply(L,P,(l,c)-> l*c)

     +---------------------+
o9 = || 2418  996   2676 | |
     || 0     0     0    | |
     || 0     -2170 2336 | |
     || -4650 0     4640 | |
     || 4176  -2363 0    | |
     +---------------------+
     || -799  -4310 -4744 ||
     || 0     0     0     ||
     || 0     3153  -4110 ||
     || -1742 0     -2575 ||
     || 2884  -3026 0     ||
     +---------------------+
     || -456  -2638 -1974 ||
     || 0     0     0     ||
     || 0     873   57    ||
     || -2291 0     -422  ||
     || -1987 4218  0     ||
     +---------------------+
```

Now, we define a polynomial ring over FF with indeterminates for the 11 unknown camera parameters:

```
R = FF[r_(1,1), r_(1,2), r_(1,3),
  r_(2,1), r_(2,2), r_(2,3),
  t_(1,1), t_(2,1),t_(1,2), t_(2,2), t_(2,3)]
```

The next line defines a list `Rs` to be a list representing each of the camera matrices' rotations.

```
Rs = {matrix{{1,0,0},{0,1,0},{0,0,1}},
matrix{{ -r_(1,1)^2-r_(1,2)^2+r_(1,3)^2+1,
          -2*r_(1,2)*r_(1,3)+2*r_(1,1),
          2*r_(1,1)*r_(1,3)+2*r_(1,2) },
        { -2*r_(1,2)*r_(1,3)-2*r_(1,1),
          -r_(1,1)^2+r_(1,2)^2-r_(1,3)^2+1,
          -2*r_(1,1)*r_(1,2)+2*r_(1,3) },
        { 2*r_(1,1)*r_(1,3)-2*r_(1,2),
          -2*r_(1,1)*r_(1,2)-2*r_(1,3),
          r_(1,1)^2-r_(1,2)^2-r_(1,3)^2+1 }},
matrix{{ -r_(2,1)^2-r_(2,2)^2+r_(2,3)^2+1,
          -2*r_(2,2)*r_(2,3)+2*r_(2,1),
          2*r_(2,1)*r_(2,3)+2*r_(2,2)},
        { -2*r_(2,2)*r_(2,3)-2*r_(2,1),
          -r_(2,1)^2+r_(2,2)^2-r_(2,3)^2+1,
          -2*r_(2,1)*r_(2,2)+2*r_(2,3) },
        { 2*r_(2,1)*r_(2,3)-2*r_(2,2),
```

```
          -2*r_(2,1)*r_(2,2)-2*r_(2,3),
          r_(2,1)^2-r_(2,2)^2-r_(2,3)^2+1
}}};
```

Similarly, we may define a list of camera translations,

```
ts={matrix{{0},{0},{0}},
     matrix{{t_(1,1)},{t_(1,2)},{1}},
     matrix{{t_(2,1)},{t_(2,2)},{t_(2,3)}}};
```

and a list of camera matrices

```
C = apply(Rs,ts,(R,t)->R|t);
```

The polynomials defining each line correspondence (LC) and common point constraint (CP) will depend on `C` and certain submatrices from `L`. The following functions return ideals generated by these polynomials:

```
cl = (C,L') -> minors(3,
        matrix apply(C,L',(c,l)->{l*c}),
        Strategy=>Cofactor);
cp = (C,L') -> minors(4,
        matrix apply(C,L',(c,l)->{l*c}),
        Strategy=>Cofactor);
```

We let `I` be the ideal of camera poses consistent with `L`, which may be constructed as follows:

```
Icl = sum(nVisibleLines,
       i->cl(C,L/(l->l^{i})));
Icp = sum(lineIncidences,
       LINES->cp(C,L/(l->l^LINES)));
I = Icl + Icp;
```

At this point we must note that the matrices defined in `Rs` are "scaled" so that the denominators appearing in the usual Cayley parameterization of $\mathrm{SO}(3)$—namely, $d_1 = r_{2,1}^2 + r_{2,2}^2 + r_{2,3}^2 + 1$ and $d_2 = r_{3,1}^2 + r_{3,2}^2 + r_{3,3}^2 + 1$—do not appear. This scaling has the effect of introducing spurious solutions to the determinantal equations where these denominators may be zero. A standard technique for eradicating these spurious solutions is via introducing an auxiliary variable z and equation $z\,(d_1\,d_2) + 1$ enforcing $d_1 \neq 0$ and $d_2 \neq 0$:

```
Rz = FF[z, gens R,
        MonomialOrder=>{Eliminate 1}]
Iz = sub(I,Rz) + ideal(1+z*
        (r_(2,1)^2+r_(2,2)^2+r_(2,3)^2+1)*
        (r_(1,1)^2+r_(1,2)^2+r_(1,3)^2+1));
```

We may compute a Gröbner basis for the ideal `Iz` using Maculay2's implementation of the *F4 algorithm.*

```
gbIz = groebnerBasis(Iz, Strategy => "F4");
```

The monomial order for the ring `Rz` is a block-wise graded reverse lexicographical ordering that eliminates the auxiliary variable `z`. It follows [**?**] that the original ideal `I` is generated by the Gröbner basis elements which do not contain this variable.

```
gbI = selectInSubring(1,gbIz);
```

We get the dimension and degree of `I` by passing to the initial ideal.

```
i23 : inI = ideal sub(leadTerm gbI,R);

o23 : Ideal of R

i24 : dim inI

o24 = 0

i25 : degree inI

o25 = 40
```

2.2. Computing the degree symbolically using setup code

The following Macaulay2 code replicates the entire Gröbner basis computation outlined in the previous section. Note that a seed for the random number generator has been set to ensure a reproducible result.

```
setRandomSeed 0;
m = 3;
FF = ZZ/nextPrime 10000;
isParametric = false;
D = (3,2,{{1,2},{1,3},{1,4}});
needs "problem-builder.m2";
gbIz = groebnerBasis(Iz, Strategy => "F4");
```

The script "`problem-builder.m2`" and its dependencies automate tasks such as building equations and generating point-line data. It assumes several global variables have been defined:

- `m` — the number of cameras
- `FF` — the ground field
- `isParametric` — a Boolean determining how point-line data are represented. A value of `false` indicates that the data are set up randomly over the ground field. A value of `true` means that point-line data are defined in terms of parametric indeterminates, to be specialized later at some fabricated point as in the minimality check described in the next section.
- `D` — an encoding of the problem as described in the previous section.

2.3. Minimality check via Jacobian

We now show how to check minimality without computing degrees for "2111_1."

```
setRandomSeed 0;
m = 3;
FF = ZZ/nextPrime 10000
isParametric = true;
D = (3,2,{{1,2},{1,3},{1,4}});
needs "problem-builder-matrices.m2"
matrices = pointMatrices | lineMatrices;
```

As before, the CP and LC `matrices` are defined over a polynomial ring in 11 indeterminates. However, the coefficient ring is itself a polynomial ring in 27 indetermines, representing the data defining an instance of the problem.

```
i8 : (numgens R, numgens coefficientRing R)

o8 = (11, 27)
```

As in the pseudocode for the minimality check, we fabricate a parameter point satisfying the LC and CP equations:

```
i9 : xy = fabricatedVectorFLPQ D;

              1        38
o9 : Matrix FF  <--- FF

i10 : netList(lineMatrices/(
              m->ker transpose sub(m,xy)))

      +---------------+
o10 = |image | -2428 ||
      |      | -1877 ||
      |      | 1     ||
      +---------------+
      |image | -1975 ||
      |      | 4507  ||
      |      | 1     ||
      +---------------+
      |image | 2657  ||
      |      | -3077 ||
      |      | 1     ||
      +---------------+

i11 : netList(pointMatrices/(
              m->ker sub(m,xy)))

      +---------------+
o11 = |image | -2511 ||
      |      | 417   ||
      |      | -4137 ||
      |      | 1     ||
      +---------------+
      |image | -3306 ||
      |      | 2306  ||
      |      | 1800  ||
      |      | 1     ||
      +---------------+
      |image | 3529  ||
      |      | 2429  ||
      |      | -1474 ||
      |      | 1     ||
      +---------------+
```

The function `goodMinors` then checks minimality by computing a subset of the equations whose Jacobian at `xy` with respective to the 11 camera parameters has maximal rank.

```
i12 : gm = goodMinors(pointMatrices,
                      lineMatrices,xy);

i13 : #gm

o13 : 11
```

2.4. Numerical computations of degree

As an additional check, the degrees of minimal problems with 2 and 3 cameras were re-computed using *monodromy*. [**?**] This is a general, randomized technique for solving parametric polynomial systems $F(c, y) = 0$ for generic parameter values y. We also used monodromy to compute the degrees of problems with 4 and 5 cameras. For the minimal problem with 6 cameras, monodromy did not terminate after several days, but computed more than $450,000$ solutions. Rather than provide detailed pseudocode, we outline the salient features of this approach:

- Starting from a generic point $(c_0, y_0) \in \text{Inc}'$ (cf. main text) with $F(c_0, y_0) = 0$, known solutions are numerically continued along a fixed set of paths $h_0, \dots h_j$, where each $h_i : [0, 1] \to \mathcal{Y}_{p,l,\mathcal{I},m}$ is a random path connecting the solutions of $F(\cdot, y_0) = 0$ with those of some other fixed instance $F(\cdot, y_1) = 0$
  - Here, "random path" is meant in the sense that each $h_i(0)$ (resp. $h_i(1)$) is randomly generated point defining a problem instance with the same solutions as y_0 (resp. y_1.) For example, we might take $h_i(0) = \gamma\, y_0$ for some random constant $\gamma \in \mathbb{C}$, yielding the well-known γ-trick of [**?**].
  - Continuation along some path from $t = 0$ to 1 followed by continuation from 1 to 0 induces a permutation of the solutions of $F(\cdot, y_0) = 0$. The group of all such permutations acts transitively by the irreducibility of Inc′—thus all solutions may be discovered starting from one.
- The algorithm maintains a set of correspondences between solutions along each path h_i, and attempts the above continuation step until no other correspondences may be established.
  - This stabilization-based stopping criterion comes with no theoretical guarantee of correctness. Heuristic arguments suggest that all solutions are found with very high probability, even for small j and problems of moderate-to-large degree. [**?**] [**?**] Multiple correct runs for the problems in 2 and 3 views support this thesis, increasing our confidence in the reported results. Nevertheless, we explicitly mark in our table of results the degrees which were only computed numerically with a "*", a practice originating from [**?**].

Computations were performed using the Macaulay2 package `MonodromySolver`. At the time of writing, this package is available in the current release of Macaulay2 (version 1.13.) However, the computations made use of features not available in the currently released package:

- To guard against the possibility of *path-jumping* (which could result in extraneous solutions), the ranks of all CP and CL matrices were computed numerically for each newly discovered solution by thresholding singular values less than 10^{-4}. If any CP or CL matrix did not yield the expected ranks of 3 or 2, the solution was thrown out and not used to generate further correspondences.

- To speed up time-intensive polynomial evaluation, `MonodromySolver` was linked to the package `SLPexpressions`, which allows for custom straight-line program representations of polynomial systems.

These enhancements will be incorporated into a future version of the package.

The output of the main function `monodromySolve` includes solution data for a chosen parameter point y_i, which may be written to file. This solution data may be used as a *start system* for solving an instance of the same problem via homotopy continuation. We illustrate this for the minimal problem "2111_1."

We begin by loading the setup script `"numerical-problem-builder.m2"`.

```
setRandomSeed 0
m = 3
D = (3,2,{{1,2},{1,3},{1,4}})
Jpivots = {0, 1, 4, 5, 8, 9, 14, 29,
          33, 44, 48, 57, 58, 59}
RERUNMONODROMY = false;
needs "numerical-problem-builder.m2"
```

The global variable `Jpivots` is a list indexing a square subsystem of the LC and CP equations. These equations are the same except that rotations and translations are represented in homogeneous coordinates (*i.e.* quaternion parameterization of $\mathrm{SO}(3)$)—yielding an extra three variables and equations. The square subsystem, depending on parametric indeterminates $y \in \mathbb{C}^{62}$ as well as $c \in \mathbb{C}^{14}$, is an object of type `GateMatrix` which the setup script assigns to the variable `F`. We may load the results of a previous monodromy run for this problem and verify that the starting solutions are valid:

```
i7 : (yStart, cStarts) =
        readStartSys "2111_1-start";

i8 : max(cStarts/(c->
        norm evaluate(F,c||yStart)))

o8 = 2.49953543586694e-12
o8 : RR (of precision 53)
```

We now consider the problem of reconstructing relative camera poses

```
i13 : netList rotations23

      +-------------------------------+
o13 = || .100226   -.858789 .502431  ||
      || .991267   .129682  .0239204 ||
      || -.0856987 .495646  .864287  ||
      +-------------------------------+
      || -.246694 .614271  .749542  | |
      || .828175  -.268025 .492228  | |
      || .503258  .742182  -.442603 | |
      +-------------------------------+

i14 : netList translations23

      +-------------+
o14 = || .455949   ||
      || .00471457 ||
      || .771746   ||
      +-------------+
      || .74002  |  |
      || .20217  |  |
      || .774737 |  |
      +-------------+
```

from line data in each view

```
i15 : netList L

      +-------------------------------+
o15 = || -.729481 -.373996 .5727   | |
      || -.617925 -.259526 .742169 | |
      || -.33284  -.499815 .799626 | |
      || -.441933 -.298528 .845917 | |
      || -.886353 -.145719 .439481 | |
      +-------------------------------+
      || -.568543 .804717  -.170851 ||
      || -.685669 .639348  -.347982 ||
      || .881415  -.405026 .243023  ||
      || .490952  -.819545 .295486  ||
      || .158324  -.969189 .188695  ||
      +-------------------------------+
      || -.518733 .16617   .838632 | |
      || -.61258  .0842219 .785909 | |
      || -.468687 -.259022 .844535 | |
      || -.595602 -.232359 .768939 | |
      || -.589568 .295501  .751724 | |
      +-------------------------------+.
```

The derived (parameter, solution) pair is defined by respective variables `yTarget`, `cTarget`. The starting solutions are numerically continued to solutions of `F` specialized at `yTarget` using core functions for `NumericalAlgebraicGeometry` in Macaulay2.

```
PH' = specialize(PH,yStart||yTarget);
cTargets = trackHomotopy(PH', cStarts);
```

The following code filters the target solutions `cTargets` for the ground-truth solution `y`.

```
cTargets2 = cTargets/(c-> (
    R2params = take(c.Coordinates,{0,3});
    Q2R(R2params,Normalized=>true)));
i=minPosition(
    cTargets2/(R2->
    norm(R2-rotations23#0)));
y=(cTargets#i).Coordinates;
```

We recover the original camera poses:

```
i23 : Q2R(take(y,{0,3}),Normalized=>true)

o23 = | .100226   -.858789 .502431  |
      | .991267   .129682  .0239204 |
      | -.0856987 .495646  .864287  |

                3          3
o23 : Matrix CC    <--- CC
             53         53

i24 : Q2R(take(y,{4,7}),Normalized=>true)

o24 = | -.246694 .614271  .749542  |
      | .828175  -.268025 .492228  |
      | .503258  .742182  -.442603 |

                3          3
o24 : Matrix CC    <--- CC
             53         53

i25 : transpose matrix take(y,{8,13})

o25 = | .455949   |
      | .00471457 |
      | .771746   |
      | .74002    |
      | .20217    |
      | .774737   |
```