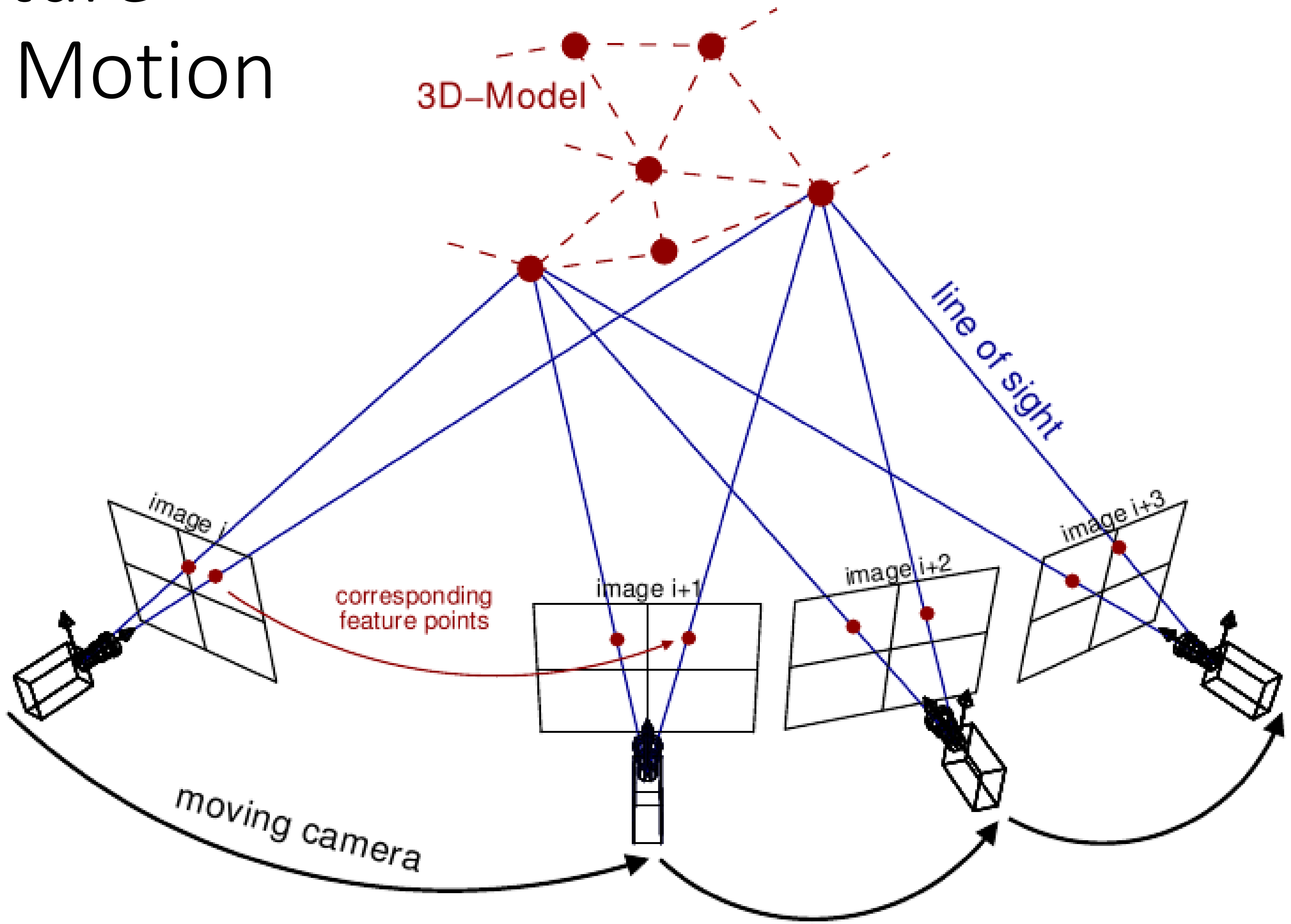
Lab 8: Visual Odometry of AR Drone using Structure from Motion

EE565: Mobile Robotics

Structure from Motion



SfM Algorithm
Use case:

- A camera is taking images of a static object from different views
- Given 2D Image points correspondence reconstruct the 3D Point Cloud
- As the object rotates so a different part of it is seen to the Camera, and thus, different points & correspondences are detected.

SfM Algorithm

Implementation:

Start with 2 images of the same scene

- Compute Camera relative Pose
  - Take the two images and calculate LK optical flow on feature points
  - Run 8-point algorithm / RANSAC to find Fundamental Matrix and consequently find Essential Matrix
  - Find R and T matrices by decomposing Essential Matrix (SVD)
- Triangulate to result the 3D Points (store 3D points in a PointCloud)
- Use the 3D points and corresponding 2D points in both images in the solvePnP function to estimate camera motion odometry.
- Repeat above with 3 images to get better scene reconstruction

Sample Code

```
**
 * Structure from motion from 2 cameras, using farneback optical flow as the 'features'
 * No, this doesn't work on more than 2 cams, because that requires bundle adjustment, which
 * I'm still searching if there's an OpenCV implementation of it
 */
Mat sfm( Mat& img1, Mat& img2, Mat& cam_matrix, Mat& dist_coeff ) {
    Mat gray1, gray2;
    cvtColor( img1, gray1, CV_BGR2GRAY );
    cvtColor( img2, gray2, CV_BGR2GRAY );

    /*  Find the optical flow using farneback dense algorithm
        Note that you might need to tune the parameters, especially window size.
        Smaller window size param, means more ambiguity when calculating the flow.
     */
    Mat flow_mat;
    calcOpticalFlowFarneback( gray1, gray2, flow_mat, 0.5, 3, 12, 3, 5, 1.2, 0 );

    vector left_points, right_points;
    for ( int y = 0; y < img1.rows; y+=6 ) {
        for ( int x = 0; x < img1.cols; x+=6 ) {
            /* Flow is basically the delta between left and right points */
            Point2f flow = flow_mat.at(y, x);

            /*  There's no need to calculate for every single point,
                if there's not much change, just ignore it
             */
            if( fabs(flow.x) < 0.1 && fabs(flow.y) < 0.1 )
                continue;

            left_points.push_back(  Point2f( x, y ) );
            right_points.push_back( Point2f( x + flow.x, y + flow.y ) );
        }
    }

    /* Undistort the points based on intrinsic params and dist coeff */
    undistortPoints( left_points, left_points, cam_matrix, dist_coeff );
    undistortPoints( right_points, right_points, cam_matrix, dist_coeff );
```

Take two images

Optical Flow and
2D point correspondences

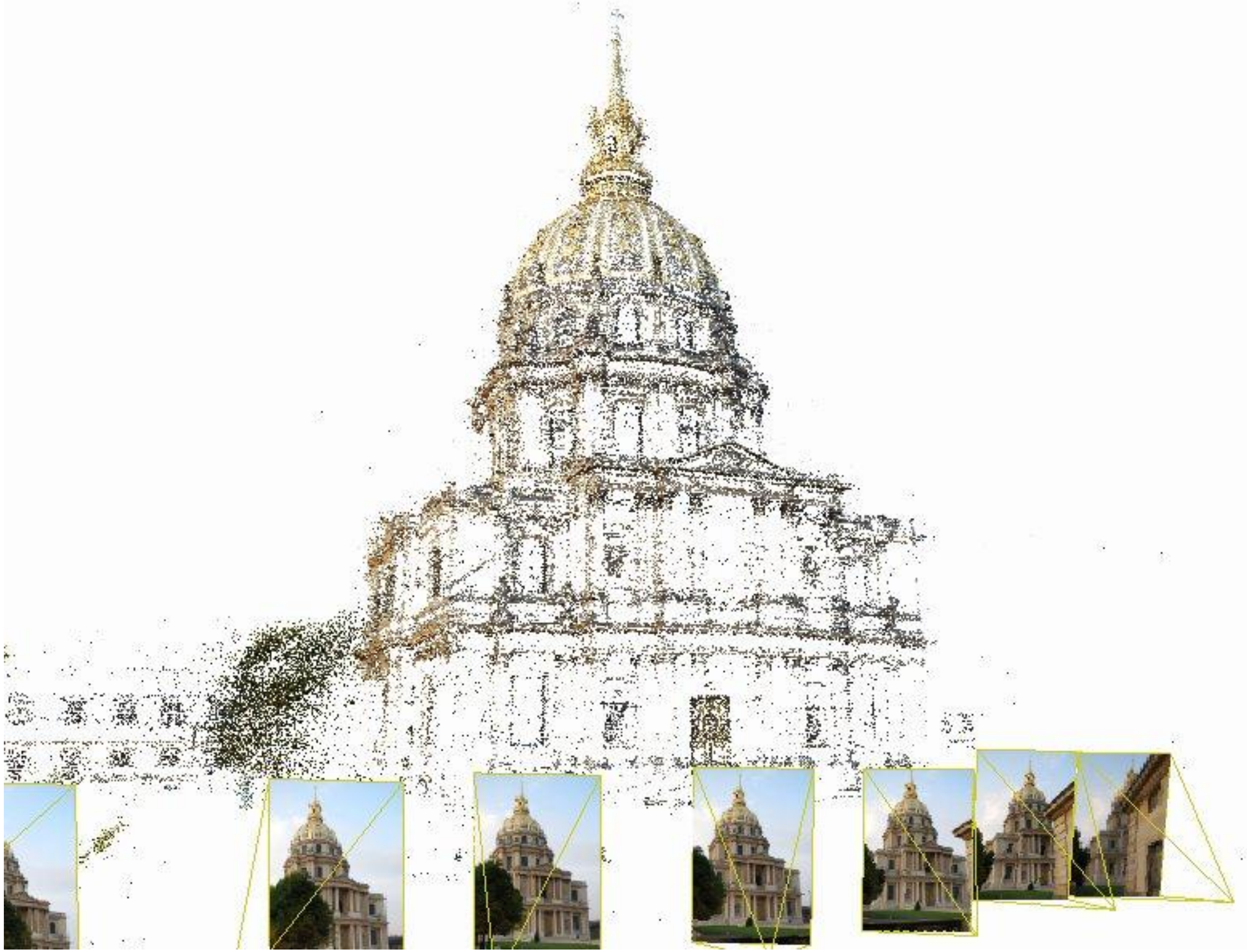
```
    /* Try to find essential matrix from the points */
    Mat fundamental = findFundamentalMat( left_points, right_points, FM_RANSAC, 3.0, 0.99 );
    Mat essential   = cam_matrix.t() * fundamental * cam_matrix;

    /* Find the projection matrix between those two images */
    SVD svd( essential );
    static const Mat W = (Mat_(3, 3) <<
                     0, -1, 0,
                     1, 0, 0,
                     0, 0, 1);

    static const Mat W_inv = W.inv();

    Mat_ R1 = svd.u * W * svd.vt;
    Mat_ T1 = svd.u.col( 2 );

    Mat_ R2 = svd.u * W_inv * svd.vt;
    Mat_ T2 = -svd.u.col( 2 );

    static const Mat P1 = Mat::eye(3, 4, CV_64FC1 );
    Mat P2 =( Mat_(3, 4) <<
             R1(0, 0), R1(0, 1), R1(0, 2), T1(0),
             R1(1, 0), R1(1, 1), R1(1, 2), T1(1),
             R1(2, 0), R1(2, 1), R1(2, 2), T1(2));

    /*  Triangulate the points to find the 3D homogenous points in the world space
        Note that each column of the 'out' matrix corresponds to the 3d homogenous point
     */
    Mat out;
    triangulatePoints( P1, P2, left_points, right_points, out );

    /* Since it's homogenous (x, y, z, w) coord, divide by w to get (x, y, z, 1) */
    vector splitted = {
        out.row(0) / out.row(3),
        out.row(1) / out.row(3),
        out.row(2) / out.row(3)
    };

    merge( splitted, out );

    return out;
}
```

Find Essential Matrix

Camera relative Pose

3D points

After triangulation, use 3D-2D point correspondences to find camera motion



References

- Jebara, Tony, Ali Azarbayejani, and Alex Pentland. "3D structure from 2D motion." *Signal Processing Magazine, IEEE* 16.3 (1999): 66-84.