AA274 (Winter 2017-18): Problem Set 2

Anqi Fu

February 9, 2018

1 Camera Calibration

(i) See submitted code. We are given a 7×9 chessboard grid with each square's side length of $d_{square} = 20.5$ mm. The axes are arranged in traditional fashion with the origin at the bottom left. To generate the world coordinates, we stack the rows of

$$
G_X = \begin{pmatrix} 0 & d_{square} & \cdots & 8d_{square} \\ 0 & d_{square} & \cdots & 8d_{square} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & d_{square} & \cdots & 8d_{square} \end{pmatrix} \quad \text{and} \quad G_Y = \begin{pmatrix} 6d_{square} & 6d_{square} & \cdots & 6d_{square} \\ 5d_{square} & 5d_{square} & \cdots & 5d_{square} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix},
$$

producing $(X_k, Y_k) \in \mathbb{Z}_+^{2 \times 63}$ for each of the $k = 1, \ldots, 23$ chessboards.

(ii) See submitted code. For a given chessboard i, form the row vector $\tilde{M} = (X_k, Y_k, \mathbf{1})^T \in \mathbb{Z}_+^{189}$ by stacking the world coordinates and a 63-length vector of ones. Define

$$
L = \begin{pmatrix} \tilde{M}^T & \mathbf{0}^T & -u_{meas}\tilde{M}^T \\ \mathbf{0}^T & \tilde{M}^T & -v_{meas}\tilde{M}^T \end{pmatrix} \in \mathbf{R}^{378 \times 9}
$$

and solve $Lx = 0$ by taking the SVD of L and setting x^* equal to the right singular vector associated with the smallest singular value. Then, $x^* = (h_1, h_2, h_3) \in \mathbf{R}^9$ is the stacked columns of our homography matrix H_k for chessboard k.

(iii) See submitted code. Let the i-th column vector of H_k be $h_i = (h_{i1}, h_{i2}, h_{i3})$, and define

$$
v_{ij} = \begin{pmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{pmatrix} \quad \text{and} \quad V_k = \begin{pmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{pmatrix}
$$

for a particular chessboard k. Form $V \in \mathbb{R}^{378 \times 6}$ by stacking the V_k matrices. We can solve $Vb = 0$ by applying the SVD as in the previous part. Given the solution $b^* = (B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33})$, we can compute

$$
\begin{aligned}
v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\
\lambda &= B_{33} - (B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23}))/B_{11} \\
\alpha &= \sqrt{\lambda/B_{11}} \\
\beta &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\
\gamma &= -B_{12}\alpha^2\beta/\lambda \\
u_0 &= \gamma v_0/\beta - B_{13}\alpha^2/\lambda
\end{aligned}
$$

and form the camera intrinsic matrix

$$
A = \begin{pmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix}
$$

(iv) See submitted code. Given A, we compute for each chessboard

$$
q_1 = \lambda A^{-1}h_1, \quad q_2 = \lambda A^{-1}h_2 \quad q_3 = q_1 \times q_2
$$

where h_j is column j of H_k, and our normalizing factor is $\lambda = 1/\|A^{-1}h_1\|$. Then, we form the matrix $Q = (q_1, q_2, q_3)$ by concatenating the column vectors and take its SVD to get $Q = USV^T$. The desired rotation matrix is $R = UV^T$ with translation $t = \lambda A^{-1}h_3$.

(v) See submitted code. Assuming $f = 1$, the desired transformation comes from the lecture notes:

$$
\begin{pmatrix} X_C \\ Y_C \\ Z_C \end{pmatrix} = \begin{pmatrix} R & t \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} X_C/Z_C \\ Y_C/Z_C \end{pmatrix}
$$

$$
\begin{pmatrix} x_h \\ y_h \\ z_h \end{pmatrix} = A \begin{pmatrix} R & t \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x_h/z_h \\ y_h/z_h \end{pmatrix}
$$

(vi) See submitted code. Given $k = (0.15, 0.01)$, we can solve for

$$
\begin{aligned}
\breve{x} &= x + x(k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \\
\breve{y} &= y + y(k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \\
\breve{u} &= u_0 + \alpha\breve{x} + \gamma\breve{y} \\
\breve{v} &= v_0 + \beta\breve{y}
\end{aligned}
$$

2   Line Fitting

(i) See submitted code.

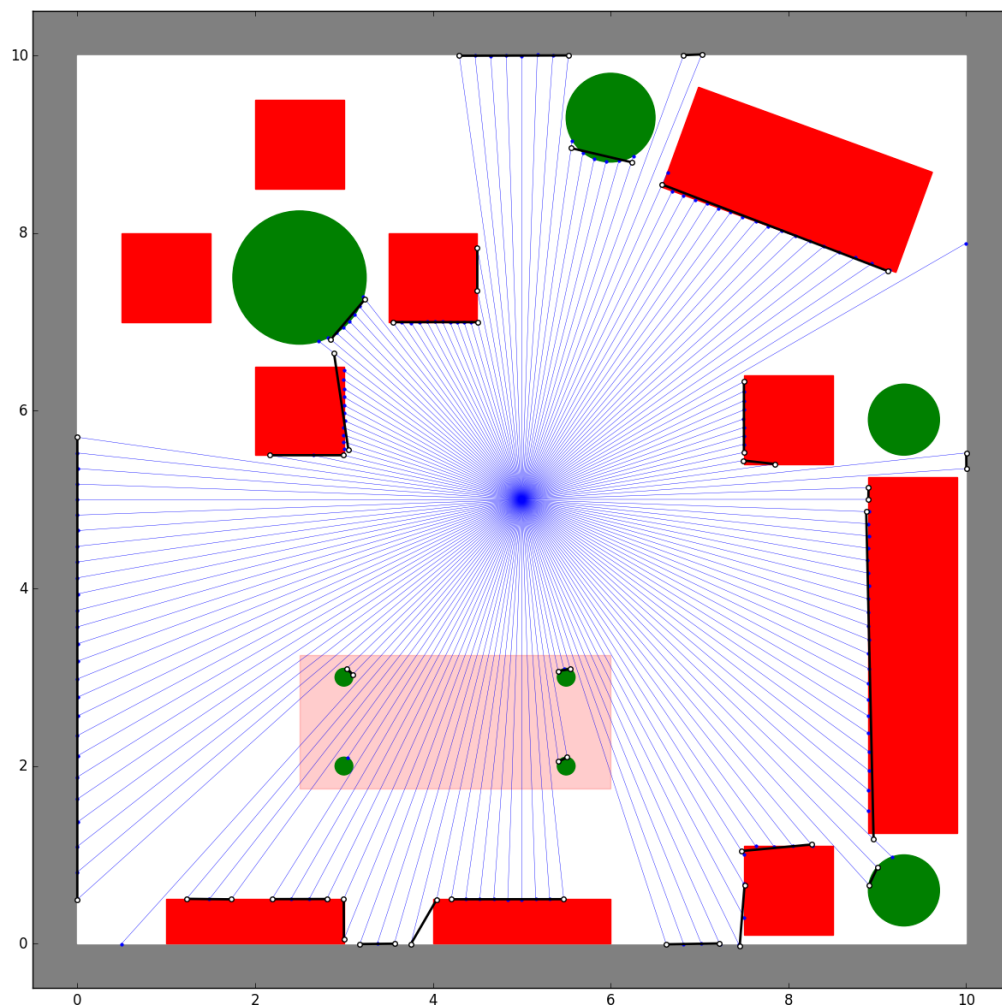(ii) Using the suggested algorithm, we extracted the following lines for each data set.

Range Data $(x_r, y_r, n_{pts}) = (5, 5, 180)$
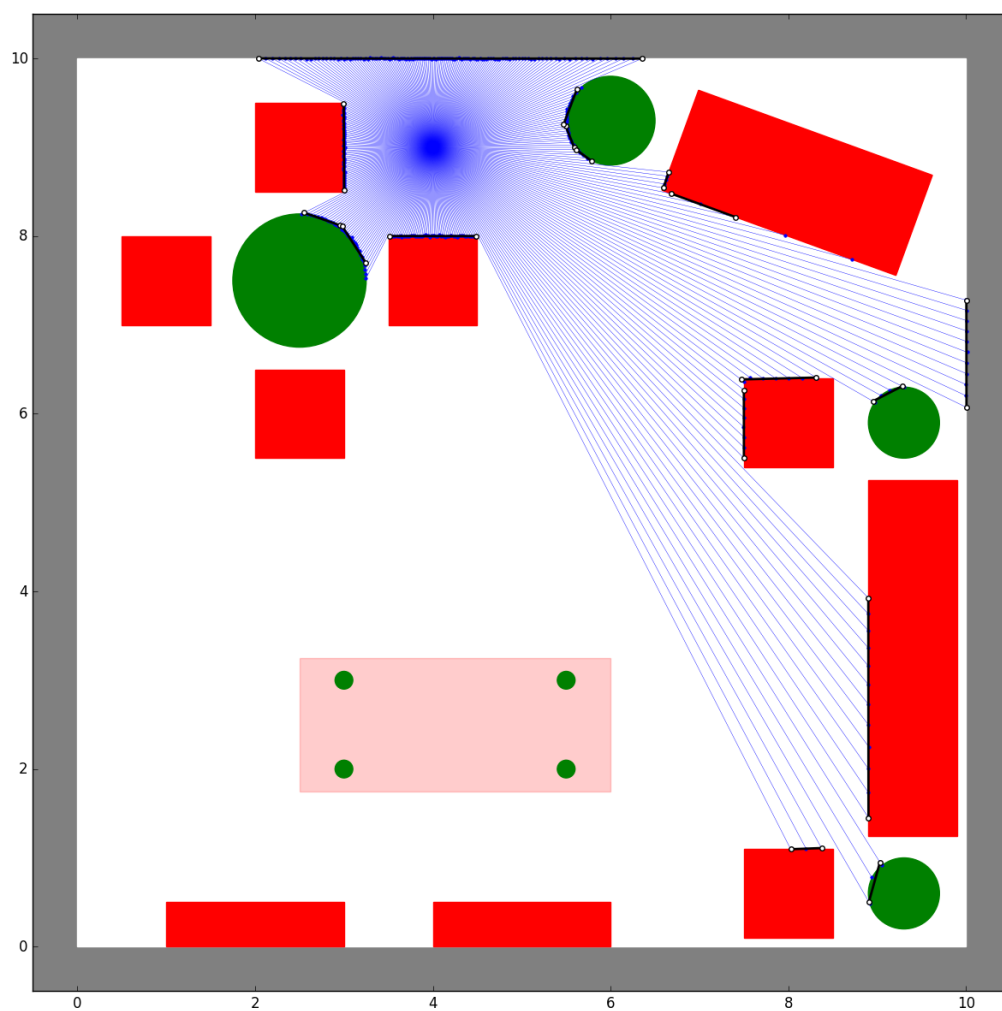
```
LINE_POINT_DIST_THRESHOLD = 0.05
 MIN_POINTS_PER_SEGMENT = 0.065
         MIN_SEG_LENGTH = 2
         MAX_P2P_DIST = 0.8
```

Range Data $(x_r, y_r, n_{pts}) = (4, 9, 360)$

LINE_POINT_DIST_THRESHOLD = 0.05
MIN_POINTS_PER_SEGMENT = 0.025
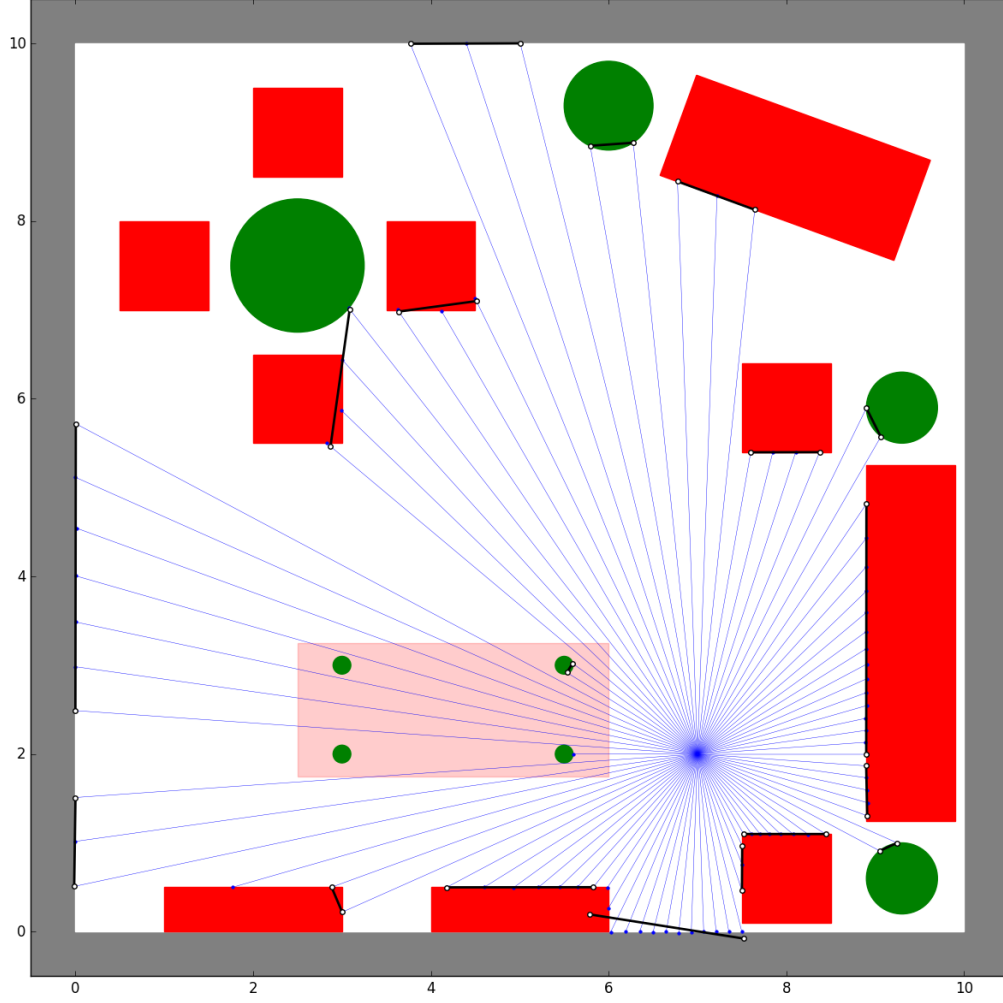MIN_SEG_LENGTH = 3
MAX_P2P_DIST = 0.54

Range Data $(x_r, y_r, n_{pts}) = (7, 2, 90)$

```
LINE_POINT_DIST_THRESHOLD = 0.05
MIN_POINTS_PER_SEGMENT = 0.165
MIN_SEG_LENGTH = 2
MAX_P2P_DIST = 0.42
```

3   Tensorflow and HOG + SVM Pedestrian Detection

(i) Let $p, q \in \mathbb{R}^2$ be two points on the parallel hyperplanes, i.e. $w \cdot p - b = 1$ and $w \cdot q - b = -1$. The perpendicular distance between the planes is the projection of the vector $(p - q)$ onto the unit normal to the plane, $\frac{w}{\|w\|}$, which is simply

$$(p - q) \cdot \frac{w}{\|w\|} = \frac{w \cdot p - w \cdot q}{\|w\|} = \frac{(b + 1) - (b - 1)}{\|w\|} = \frac{2}{\|w\|}$$

(ii) TensorFlow's workflow was created for modeling and executing distributed computati-

ons. The computation graph allows the user to see which mathematical operations will occur in parallel, making it easier to design and debug algorithms. Moreover, the actual data analysis is separated into another step to facilitate execution across multiple, distributed machines. Problems with the algorithm can thus be isolated from problems with the devices that carry out the computation, such as a breakdown in communication. This workflow differs from numpy, which immediately executes operations in the order they are entered. TensorFlow's paradigm makes sense for machine learning, because in this space, algorithms are complex and datasets are generally very large, requiring significant runtime on multiple machines. To avoid wasting resources, users will first test their computation graph locally before starting a cluster and feeding it the input data.

(iii) See submitted code.

(iv) With a learning rate of $\gamma = 0.1$ and $\lambda = 0.4$, I achieved a misclassification rate of 0.214 on the test dataset. This is rather high because the classes cannot be separated by a single hyperplane. The red class has curved boundaries on both top and bottom with the blue class. Thus, no matter where we draw the hyperplane, a large proportion of the classes will be misclassified. This model has placed it along the top boundary, so that all the blue points below it are labeled as red.

Misclassified Data with Original Features
$$(\gamma = 0.1, \ \lambda = 0.4)$$
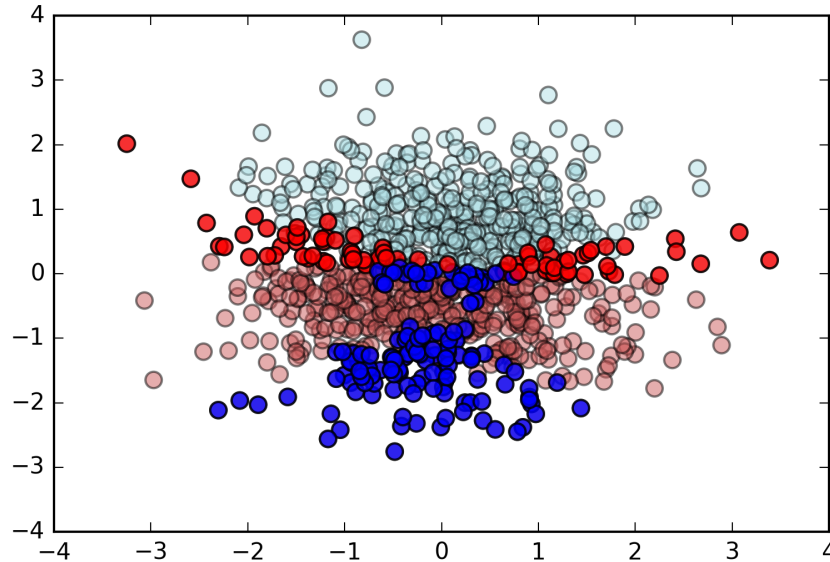
(v) See submitted code.

(vi) I selected the features $(x_1, x_2, x_1^2, x_2^2, x_1 x_2)$ because the boundaries between the two classes look like parabolas, so I expected there to be a quadratic dependency in the data. With a learning rate of $\gamma = 0.1$ and $\lambda = 1.1$, I achieved a misclassification rate of 0.120 on the test dataset.

Misclassified Data with Custom Features
$$(\gamma = 0.1, \ \lambda = 1.1)$$
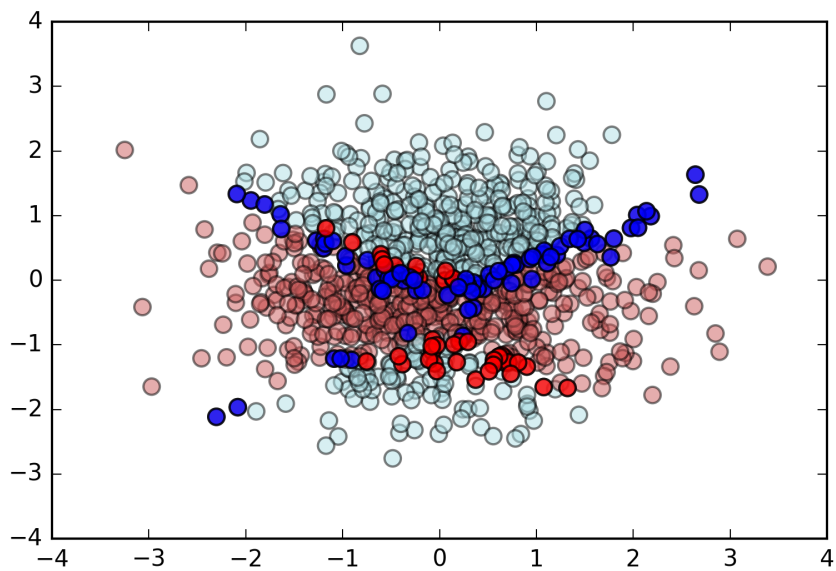
(vii) See submitted code.

(viii) With a learning rate of $\gamma = 0.075$ and $\lambda = 0.825$, I achieved a misclassification rate of 0.130 on the test dataset.

4 Classification and Sliding Window Detection

(i) See submitted code.

(ii) See submitted code.

(iii)

(iv)

(v)

(vi) See submitted code.

(vii)