Homework 1

Perception for Autonomous Robots

February 7, 2022

Instructors:
S. Charifa

Student:
Jerry Pittman, Jr. (117707120)

Semester:
Spring 2022

Course code:
ENPM673

Contents

List of Figures

1 Problem 1

1.1 Part 1: Compute FOV

I solved for the half angle in each direction then doubled that result. I used the atan2 function to ensure positive quadrant although a basic arctangent function would suffice for this particular case. It is a square camera but I solved each direction independently to have base code for a rectangular camera sensor.

$$FOV_{Horizonatal} = 2 * \text{atan2}(camera_{width}, 2f) = 0.5460$$

1.2 Part 2: Compute Minimum number of Pixels of Object

I first scaled the actual object's height to it's projected size using the focal length, distance from the camera, and aperture. I orignally solved for purely the projection location then after reviewing the problem when writing this report thought it odd to not account for actual size of object and discovered I didn't scale. In order to determine the $pixels_{camera}$ in each direction, I took the $\sqrt{5MP}$, since the camera is square. I rounded down even though the decimal was .9 since pixels need to be rounded down vice up for whole pixels (i.e. no partial pixels)

$$ObjectPixels_{horizontal_{min}} = \frac{f * object_{width} * pixels_{camera_{horizontal}}}{z * camera_{width}} = 99.82$$

$$ObjectPixels_{total_{min}} = ObjectPixels_{horizontal_{min}} * ObjectPixels_{width_{min}} = 9964.9 \approx 9964$$

2 Problem 2

2.1 Part 1: Standard Least Squares to fit curves

The hardest part I had with this problem was determining the coordinates of the the ball for creating (x,y) coordinates based on topmost and bottommost of red pixel. I determined that I needed to convert RGB to HSV then use OpenCV's built in features to determine the pixel locations. I made one function to look for image center to make the code iterative and to cut down on code. Download and run the code "jpittma1_hw1_P2.py" within same folder as the ball videos.

For calculating the Standard (or Orthongonal) Least Squares, I used the quadratic equation $y = ax^2 + bx + c$ and created a new array A with x^2, x, and the rest zeros. From here I solved for Least Squares values:

$$A \cdot ((A^T \cdot A)^{-1} \cdot (A^T \cdot Y))$$

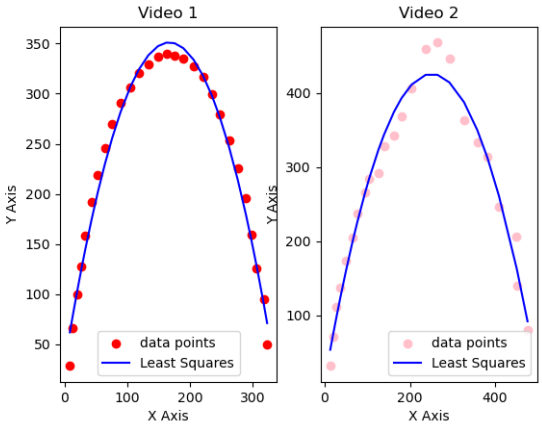

Figure 1: Standard Least Squares Line fit for each video's ball.

3 Problem 3

3.1 Part 1: Compute Covariance Matrix manually

Download and run the code "jpittma1_hw1_P3.py" within same folder as the "ENPM673_hw1_linear_regression_dataset.cs file. To solve for the Covariance Matrix (S), I solved for the mean of the x_axis values (ages), took the difference of the mean from the x_axis value (age), and then created a new matrix based on those values:

$$S = \sum_{i=1}^{n} \frac{(X_i - \overline{x})(Y_i - \overline{y})}{n-1}$$

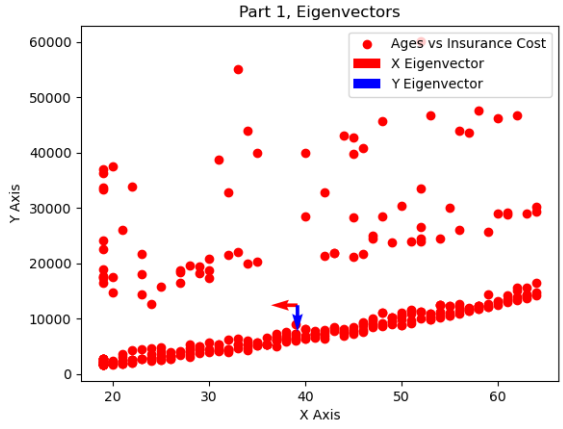

Figure 2: Eigenvectors from Covariance Matrix

3.2 Part 2: Fit line using Least Squares, Total Least Squares, and RANSAC

- I had several difficulties with this part. First of all, I was trying to make parabolic curves (based on process I used in Problem 2). I also was overwriting my global x_axis variables ("age") in my covariance function. I solved this issue by creating a new array to save the values for the

difference of the x_axis and mean of the x_axis vice using $-$ $=$ on the local x_axis variables. I also had difficulty doing linear algebra and linear regression in python and so used [4] and [2] in order to try and figure out how to operationally apply equations and concepts from the lecture [3] and textbook to 2D arrays in python understanding that they had different tasks and they used parabolas.

- For calculating **Standard Least Squares**, I used the same methodology as Problem 2 except as a line equation vice a parabola.

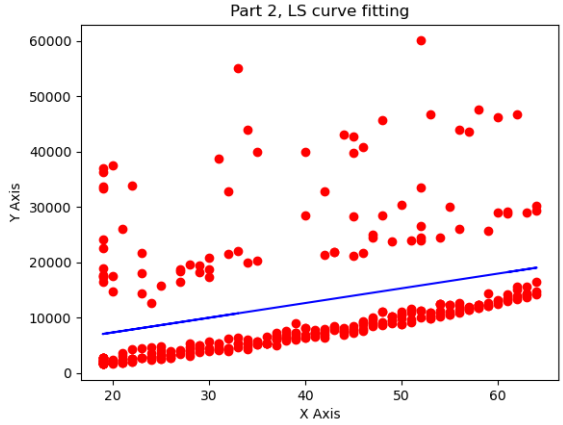

Figure 3: Standard Least Squares Line fit for Age vs Insurance Cost

- For calculating **Total Least Squares**, I solved for $U^T \cdot U$ after making the U matrix:

$$U = \begin{bmatrix} (X_i - \overline{x}) & (Y_i - \overline{y}) \end{bmatrix}$$

I then determined the smallest eigenvalue and its associated eigenvector in order to solve for the line equation:

$$y_{tls} = \frac{(A\overline{x} + B) - (Ax_{axis})}{B}$$

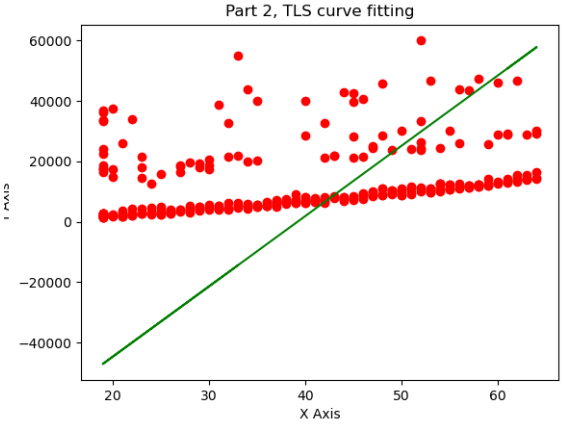

Figure 4: Total Least Squares Line fit for Age vs Insurance Cost

- For calculating the **RANSAC** line, I started with max iterations of infinity but had it recalculated each iteration based on the equation from week two's slides (i.e. using probability of outliers and desired probability). When solving for the RANSAC curve, I realized that I was modifying the global variables based on using same variable names locally. The TA assisted me and discovering why my curves were being displayed far below the main body of data because of this poor variable use and modifying the same variable to be be mean difference vice creating new a variable to store the differences in. I decided to use 95% desired probability, two samples per iteration, and threshold of:

$$threshold = \frac{\sigma_{y_{charges}}}{3}$$

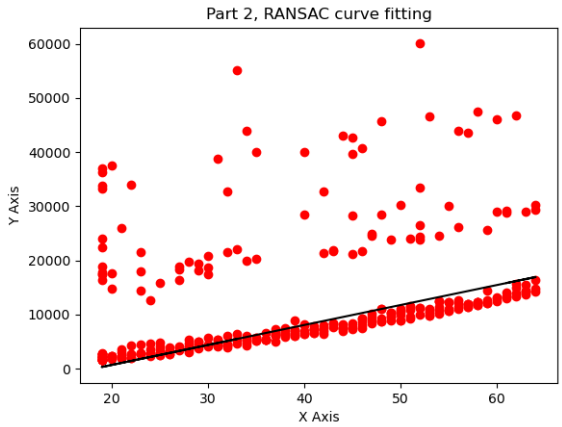

Figure 5: RANSAC Line fit for Age vs Insurance Cost

- Comparison: OLS performed better than TLS for me but RANSAC performed the best. If there was minimal error or the error (ouliers) were contained to near the main grouping of data then TLS would be the best. Based on the large variations and normalized distances of the outliers, RANSAC is the best line fit and this exercised proved it. The downside to RANSAC is the computing power required based on large number of iterations. It would take further more complicated calculations if this was a 3-Dimensional array/problem.

4 Problem 4

4.1 Part 1: Show how to compute the SVD for matrix "A"

- Singular Value Decomposition (SVD) [6] involves solving for **U** (the eigenvectors of AA^T), Σ (diagonal identity matrix consisting of $\sqrt{\lambda}$ (with λ representing eigenvalues), and **V** (eigenvectors of A^TA). I started by plugging in given x variables into the A matrix provided.
- **Solve for U** by solving for $A \cdot A^T$ then solve for λ using the equation

$$(A \cdot A^T - \lambda * I) = 0$$

  then obtain eigenvectors. Normalize this vector and you have the U matrix.
- **Solve for V** by solving $A^T \cdot A$ then solve for λ using the equation

$$(A^T \cdot A - \lambda * I) = 0$$

  then obtain eigenvectors, similar to what was performed for the U matrix. Normalize this vector and you have the V matrix. I then transposed the V matrix to get V^T.
- **Solve for** Σ by making a diagonal matrix consisting of $\sqrt{\lambda}$. I encountered several issues with doing this in python due to the matrix not being square. I used a for loop in order to iterate through the diagonal eigenvalue matrix to put into a matrix of same size as the **A** matrix (8x9). This fixed my errors.

```
diag=np.diag((np.sqrt(sorted_val_U)))

sigma=np.zeros_like(A).astype(np.float64) #8x9

for i in range(len(sigma)):
    sigma[i][i]=diag[i][i]
```

- **Compute SVD** by multiplying the three values just solved for:

$$SVD = U \cdot \Sigma \cdot V^T$$

- The Homography Matrix is as follows:

$$H = \begin{bmatrix} 0.0531 & -0.00492 & 0.615 \\ 0.0177 & -0.00393 & 0.787 \\ 0.000236 & -0.0000492 & 0.00762 \end{bmatrix}$$

5 Resources

5.1 My Github repository

https://github.com/jpittma1/ENPM673.git

5.2 References

Whole bibliography

[1] R. Bannai. "Compare tls to ols." (), [Online]. Available: https://towardsdatascience.com/total-least-squares-in-comparison-with-ols-and-odr-f050ffc1a86a. (accessed: 02.06.2022).

[2] ——, "Total least squares." (), [Online]. Available: https://github.com/RyotaBannai/total-least-squares.git. (accessed: 02.06.2022).

[3] S. Charifa, "Enpm673 spring 2022 notes," University of Maryland - College Park, MAGE, College Park, MD, Tech. Rep. Week 2 Lecture, Jan. 2022.

[4] G. A. Kumar. "Enpm673 2020." (), [Online]. Available: https://github.com/govindak-umd/ENPM673.git. (accessed: 02.06.2022).

[5] T. T. Mark Peterson. "How to calculate image resolution." (), [Online]. Available: https://www.yumpu.com/en/document/read/28938952/how-to-calculate-image-resolution-theia-technologies. (accessed: 02.06.2022).

[6] MIT. "Singular value decomposition (svd) tutorial." (), [Online]. Available: https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm. (accessed: 02.07.2022).

[7] Nist.gov. "Total least squares." (), [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section5/pmc541.htm. (accessed: 02.06.2022).

[8] C. Zaiontz. "Total least squares." (), [Online]. Available: https://www.real-statistics.com/regression/total-least-squares/. (accessed: 02.06.2022).