Accomplishments:

Both:
1. Meeting on 4/9
  a. Refined function headers for the Inverse Perspective Mapping setup portion of the processing steps.
  b. Implemented the `pixelsToWorld` function using pieces of the `invPerspTest_Our2.m` test script.
  c. Formulated an interpolation method to work with the uneven spacing in the grid of image points mapped to world coordinates. (See scanned pages at the end of this document for the development.)
  d. Discussed alternatives for the efficient implementation of this interpolation method.

Eric:
1. Finished documentation portion of the `pixelsToWorld` header.
2. Implemented the interpolation method developed during our meeting, creating the `getInterpMap` function and its documentation.
3. Implemented and documented the `getWorldImage` function for the image preprocessing stage to allow the `getInterpMap` function output to be tested.
4. Created a test script – `test_invPerspMapSetup.m` – to verify the results and test the computation time of these three functions. The results are shown below. Note that the first two steps only need to be done once for initialization. After that, only the third has run to create the inverse-perspective-mapped image for each incoming camera image.

   Time required to set up the pixel to world map:
   0.052 seconds
   Time required to set up the interpolation map:
   17.828 seconds
   Time required to apply the interpolation map for an RGB image:
   0.169 seconds

   The two figures generated by the test script are shown on the following page.
5. Typed up weekly log report.

Randy:
1. Began working on the generation of ground truth data for several test images for the purposes of algorithm verification and the creation of quantitative performance measures.
Next Steps:
1. Finish creating the ground truth answers for the selected test images.
2. Start putting together our building blocks into a working lane detection algorithm.
   a. Basic Feature Extraction functions
   b. Post-Processing functions which combine the raw feature outputs and narrow in on lane marker pixels.
   c. Lane marker isolation methods to allow polynomials to be fit to the pixels belonging to a single line.
3. Create a utility to display the results for convenient visual feedback as we develop the algorithm.
4. Complete the organization of our website
5. (Eric) Final report.
Interpolating in the X, Y matrices

\[ V_i = \text{value of function(image intensity)} \text{ at each point} \]
\[ G = \text{grid point where we want an interpolated value} \]
\[ \Phi = \text{points in the X, Y matrices telling us where} \]
\[ \text{image pixels mapped in the world frame.} \]

- Find row in X which gives \[ \Phi_{3x}, \Phi_{4x} \geq G_x : r_{34} \]
- Add 1 and you have the row for \( \Phi_1 \) and \( \Phi_2 \) where \[ \Phi_{1x}, \Phi_{2x} < G_x \text{ is guaranteed} : r_{12} \]

- Search in \( r_{12} \) for the greatest column for which \[ \Phi_{1y} \geq G_y : c_1 \]
  add 1 and we have \[ \Phi_{2y} < G_y \]
- Search in \( r_{34} \) for the greatest column for which \[ \Phi_{3y} \geq G_y : c_3 \]
  add 1 and we have \[ \Phi_{4y} < G_y \]

- Get the X and Y values from:
  \[ \Phi_{1x} = X(r_{12}, c_1) \]
  \[ \Phi_{1y} = Y(r_{12}, c_1) \]
  \[ c_1 < c_i \]

Interpolate:
- First go along \( y \) between \( \Phi_1 \rightarrow \Phi_2 \) and \( \Phi_3 \rightarrow \Phi_4 \):
  \[ d_{1y} = \Phi_{1y} - \Phi_{2y} \]
  \[ d_{2y} = \Phi_{2y} - \Phi_{3y} \]
  \[ d_{3y} = \Phi_{3y} - \Phi_{4y} \]
  \[ d_{4y} = \Phi_{4y} - \Phi_{5y} \]

  \[ \Rightarrow \]
  \[ V_{1y} = \frac{d_{1y}}{d_{1y}} \Phi_{1y} + \frac{d_{2y}}{d_{1y}} \Phi_{2y} \]

  \[ V_{3y} = \frac{d_{3y}}{d_{3y}} \Phi_{3y} + \frac{d_{4y}}{d_{3y}} \Phi_{4y} \]

  \[ \Rightarrow \]
  \[ V_{1y} = \frac{d_{1y}}{d_{1y}} \Phi_{1y} + \frac{d_{2y}}{d_{1y}} \Phi_{2y} \]

  \[ V_{3y} = \frac{d_{3y}}{d_{3y}} \Phi_{3y} + \frac{d_{4y}}{d_{3y}} \Phi_{4y} \]
Next go across from \( v_{12} \) to \( v_{3y} \) in the \( x \) direction to get \( V_3 \):

\[
\begin{aligned}
\frac{d13x}{dl} &= \rho_{3x} - \rho_{1x} \\
\frac{d3x}{dl} &= \rho_{3x} - g_x \\
\frac{dl}{dx} &= g_x - \rho_{lx}
\end{aligned}
\]

\[
\Rightarrow V_3 = \frac{d3x}{dl} V_{12} + \frac{dl}{dx} V_{3y}
\]

Now expand:

\[
V_3 = \frac{d3x}{dl} \left[ \frac{dy}{dx} V_1 + \frac{dy}{dx} V_2 \right] + \frac{dl}{dx} \left[ \frac{dy}{dx} V_3 + \frac{dy}{dx} V_4 \right]
\]

\[
V_3 = \frac{d3x}{dl} \frac{dy}{dx} V_1 + \frac{d3x}{dl} \frac{dy}{dx} V_2 + \frac{dl}{dx} \frac{dy}{dx} V_3 + \frac{dl}{dx} \frac{dy}{dx} V_4
\]

Now we have the weights for each pixel.

To get the row column in the original image, we use the original image number of rows:

\[
\begin{aligned}
\Gamma_1 &= m - p + \Gamma_{12} \\
\Gamma_2 &= m - p + \Gamma_{12} \\
\Gamma_3 &= m - p + \Gamma_{34} \\
\Gamma_4 &= m - p + \Gamma_{34}
\end{aligned}
\]

Columns remain the same.