```
def compute_innovations(self, z_raw, Q_raw):
    """
    Given lines extracted from the scanner data, tries to associate each one
    to the closest map entry measured by Mahalanobis distance.
    Inputs:
        z_raw: np.array[2,I]   - I lines extracted from scanner data in
                                    columns representing (alpha, r) in the scanner frame.
        Q_raw: np.array[I,2,2] - I covariance matrices corresponding
                                    to each (alpha, r) column of z_raw.
    Outputs:
        vs: np.array[M,2I] - M innovation vectors of size 2I
                                (predicted map measurement - scanner measurement).
    """
    ########## Code starts here ##########
    # TODO: Compute vs (with shape [M x I x 2]).

    n     = self.M                  # Num of particles. M.
    n_lin = self.map_lines.shape[1] # Num of known lines on map. J.
    n_mea = z_raw.shape[1]          # Num of scanned lines. I.

    z_raw = z_raw.T                     # shape(n_mea, 2)
    # Q_raw                             # shape(n_mea, 2, 2)
    hs = self.compute_predicted_measurements().transpose(0, 2, 1) # shape(n, n_lin, 2)

    z_mat = z_raw[None, None, :, :]     # shape(1, 1,     n_mea, 2)
    h_mat = hs[:, :, None, :]           # shape(n, n_lin, 1,     2)

    # Vectorized angle_diff()
    z_alp, h_alp = z_mat[..., 0], h_mat[..., 0]     # shape(n, n_lin, n_mea)
    z_alp, h_alp = z_alp % (2.*np.pi), h_alp % (2.*np.pi)
    diff = z_alp - h_alp
    idx = np.abs(diff) > np.pi
    sign = 2. * (diff[idx] < 0.) - 1.
    diff[idx] += sign * 2. * np.pi
    v_alp = diff
    # Reconstruct v
    v_r = z_mat[..., 1] - h_mat[..., 1]
    v_mat = np.stack((v_alp, v_r), axis=3)

    v_fat = v_mat[..., None]                # shape(n, n_lin, n_mea, 2, 1)
    Q_inv = np.linalg.inv(Q_raw)            # shape(      n_mea, 2, 2)
    Q_inv = Q_inv[None, None, :, :, :]      # shape(1, 1, n_mea, 2, 2) # PEP20

    d_mat = np.matmul(v_fat.transpose(0, 1, 2, 4, 3), Q_inv)
    d_mat = np.matmul(d_mat, v_fat)         # shape(n, n_lin, n_mea, 1, 1)
    d_mat = d_mat.reshape((n,n_lin,n_mea))  # shape(n, n_lin, n_mea)

    # For each particle, for each scanned line, this returns the index
    # of the best known line.
    d_argmin = np.argmin(d_mat, axis=1)                 # shape(n, n_mea)
    d_argmin = d_argmin[:, None, :, None]               # shape(n, 1, n_mea, 1)
    vs = np.take_along_axis(v_mat, d_argmin, axis=1)    # shape(n, 1, n_mea, 2)
    vs = vs.reshape((n, n_mea, 2))                      # shape(n, n_mea, 2)

    ########## Code ends here ##########

    # Reshape [M x I x 2] array to [M x 2I]
    return vs.reshape((self.M,-1))  # [M x 2I]
```