Appendix A: RDDF Analysis Application

```
<!--Calculate the geodesic distance (in meters) between adjacent
waypoints in the 2004 and 2004 QID and RACE Route Data Definition File
(RDDF).-->
<?php
//PHP include statements and the author's Google Maps™ key were deleted
// from the source code prior to inclusion in this technical report.
// This comment does not appear in the original source code.

  $map = $_POST['map'];
  if ($map == "Yes")
    $debug = TRUE;
  else
    $debug = FALSE;

  $temp = $_FILES['file']['tmp_name']; //temporary file name
  if (is_uploaded_file($temp))
  {
//Create primary input table (RDDF), if it does not exist:
    $connectionID = get_mysql_connectionID("thesis", "f");
    $sql = "CREATE TABLE IF NOT EXISTS RDDF (
    Waypoint INT NOT NULL,
    Latitude DOUBLE(9,7) NOT NULL,
    Longitude DOUBLE(10,7) NOT NULL,
    LBO INT NOT NULL,
    Speed INT NOT NULL,
    PRIMARY KEY(Waypoint))";
    mysql_query($sql);
//Create primary output table (RDDF_OUT), if it does not exist:
    $sql = "CREATE TABLE IF NOT EXISTS RDDF_OUT (
    waypoint1 INT NOT NULL,
    latitude1 DOUBLE(9,7) NOT NULL,
    longitude1 DOUBLE(10,7) NOT NULL,
    lbo1 INT NOT NULL,
    speed1 INT NOT NULL,
    waypoint2 INT NOT NULL,
    latitude2 DOUBLE(9,7) NOT NULL,
    longitude2 DOUBLE(10,7) NOT NULL,
    lbo2 INT NOT NULL,
    speed2 INT NOT NULL,
    s DOUBLE (7,3) NOT NULL,
    alpha1 DOUBLE (6,3) NOT NULL,
    alpha2 DOUBLE (6,3) NOT NULL,
    beta DOUBLE (6,3) NOT NULL,
    speedm DOUBLE (6, 3) NOT NULL,
    radiusm DOUBLE (10, 3) NOT NULL,
    lbom DOUBLE (10, 3) NOT NULL,
    radius DOUBLE (10, 3) NOT NULL,
    rollover TEXT NOT NULL,
    PRIMARY KEY(waypoint1))";
    mysql_query($sql);
//Delete all data in the primary input and output tables, if they
exist:
    $sql = "TRUNCATE TABLE RDDF";
    mysql_query($sql);
    $sql = "TRUNCATE TABLE RDDF_OUT";
    mysql_query($sql);
//Load primary data table (RDDF):
    $sql = "LOAD DATA LOCAL INFILE '$temp'
    REPLACE INTO TABLE RDDF
    FIELDS TERMINATED BY ','
    ENCLOSED BY ''
    ESCAPED BY '\\\\'
    LINES TERMINATED BY '\\n'";
    mysql_query($sql);
//Map header.
    echo
"<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0
Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">
<html xmlns=\"http://www.w3.org/1999/xhtml\">
  <head>
    <meta http-equiv=\"content-type\" content=\"text/html; charset=utf-
8\" />
    <link rel=\"stylesheet\" href=\"../css/style.css\" type=\"text/css;
charset=utf-8\" />
    <title>
      Route Data Definition File (RDDF) Analysis
    </title>";

    if ($debug)
    {
      echo
    "
    <script src=\"http://maps.google.com/maps?
file=api&amp;v=2&amp;key=\"
      type=\"text/javascript\"></script>
    <script type=\"text/javascript\">

    function initialize() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById(\"content-map\"));
        map.addControl(new GLargeMapControl());
        map.addControl(new GMapTypeControl());
        map.addControl(new GScaleControl());
        var polyLine = new GPolyline ([";
    }
//End map header.
//Read primary data table (RDDF):
    $sql = "SELECT Waypoint, Latitude, Longitude, LBO, Speed
    FROM RDDF";
    $result = mysql_query($sql);
    $first = TRUE;
    while ($row = mysql_fetch_array($result))
    {
      if ($first)
      {
        $waypoint2 = $row['Waypoint'];
        $latitude2 = $row['Latitude'];
        $longitude2 = $row['Longitude'];
        $lbo2 = $row['LBO'];
        $speed2 = $row['Speed'];

  //////Static Stabilty Factor (SSF).
  ////// Minimum possible value is 0.95 (1992-2000 Mitsubishi Montero).
  ////// Maximum possible value is 1.30 (2003 Honda Pilot 4-DR 4x4,
2003 Nissan Murano 4-DR 4x2/4x4, and 1980 Plymouth Arrow).
  ////// "Trends in the Static Stability Factor of Passenger Cars,
Light Trucks, and Vans"
        $ssf = $_POST['ssf'];
        if (is_numeric($ssf))
        {
          if ($ssf < 0.95)
            $ssf = 0.95;
          if ($ssf > 1.30)
            $ssf = 1.30;
        }
        else
        {
          $ssf = 1.30;
        }
  //////Reportable speed, in miles per hour (mph).
  ////// Minimum possible value is 5 mph.
  ////// Maximum possible value is 50 mph.
  ////// "DARPA Grand Challenge 2005 Route Data Definition File"
        $mph = $_POST['mph'];
        if (is_numeric($mph))
        {
          if ($mph < 5)
            $mph = 5;
          if ($mph > 50)
            $mph = 50;
        }
        else
        {
          $mph = 20;
        }
  //////Reportable change in bearing.
  ////// Minimum possible value is 5 degrees.
  ////// Maximum possible value is 90 degrees.
        $angle = $_POST['angle'];
        if (is_numeric($angle))
        {
          if ($angle < 5)
            $angle = 5;
          if ($angle > 90)
            $angle = 90;
        }
        else
        {
          $angle = 10;
        }
        $first = FALSE;
  //////Set initial course boundaries:
        $latitude_min = $latitude2;
        $latitude_max = $latitude2;
        $longitude_min = $longitude2;
        $longitude_max = $longitude2;
  //////Set initial aggregate values:
        $number = 1; //first waypoint: number of waypoints = 1.
        $length = 0; //first waypoint: total course length = 0.
        $agg_mph = 0; //first waypoint: number of waypoints at which
the speed is reportable = 0.
        $agg_angle = 0; //first waypoint: number of waypoints at which
the angle is reportable = 0.
        $agg_risk = 0; //first waypoint: number of waypoints at which
there is a rollover risk = 0.
  //////First waypoint.
        if ($debug)
        {
          echo
                                                  "
          new GLatLng($latitude2, $longitude2)";
        }
  //////End first waypoint.
      }
      else
      {
        $waypoint1 = $waypoint2;
        $latitude1 = $latitude2;
        $longitude1 = $longitude2;
        $lbo1 = $lbo2;
        $speed1 = $speed2;
        $waypoint2 = $row['Waypoint'];
        $latitude2 = $row['Latitude'];
        $longitude2 = $row['Longitude'];
        $lbo2 = $row['LBO'];
        $speed2 = $row['Speed'];
  //////Adjust course boundaries, if necessary:
        if ($latitude2 > $latitude_max)
        {
          $latitude_max = $latitude2;
        }
        if ($latitude2 < $latitude_min)
        {
          $latitude_min = $latitude2;
        }
        if ($longitude2 > $longitude_max)
        {
          $longitude_max = $longitude2;
        }
        if ($longitude2 < $longitude_min)
        {
          $longitude_min = $longitude2;
        }
  //////Waypoint.
        if ($debug)
        {
          echo
                                                "，
          new GLatLng($latitude2, $longitude2)";
        }
  //////End waypoint.
        $return = inverse($waypoint1, $latitude1, $longitude1,
$waypoint2, $latitude2, $longitude2);
  //////Increment number of waypoints by one.
        $number++;
//   //////Calculate reduced latitudes.
//         $a = 6378137.0;
//         $b = 6356752.3142;
//         $f = 1/298.257223563;
//         $Usub1 = rad2deg(atan((1 - $f) * tan(deg2rad($latitude1))));
//         $Usub2 = rad2deg(atan((1 - $f) * tan(deg2rad($latitude2))));

        $s = $return[0];
  //////Total course length += s.
        $length += $s;

        if ($return[1] < 0)
        {
          $alpha1 = rad2deg($return[1] + 2*pi());
        }
        else
        {
          $alpha1 = rad2deg($return[1]);
        }

        if ($return[2] < 0)
        {
          $alpha2 = rad2deg($return[2] + 2*pi());
        }
        else
        {
          $alpha2 = rad2deg($return[2]);
        }

      if ($number < 3)
      { //Three waypoints are required to determine the angle at which
route segments intersect.
        $alpha2p = $alpha2;
        $beta = 0;
      }
      else
      {
        if (abs($alpha2p - $alpha1) > 180)
        {
          if ($alpha2p > $alpha1)
            $beta = ($alpha2p - 360) - $alpha1;
          else
            $beta = $alpha2p - ($alpha1 - 360);
        }
        else
```

```
         {
           $beta = $alpha2p - $alpha1;
         }
         $alpha2p = $alpha2;
       }

       $return = rollover($latitude1, $lbo1, $speed1, $beta, $ssf,
$angle);

       $speedm = $return[0];
       $radiusm = $return[1];
       $lbom = $return[2];
       $radius = $return[3];
       $rollover = $return[4];

       if ($speed1 > $mph)
       {
         $agg_mph++;
       }

       if (abs($beta) > $angle)
       {
         $agg_angle++;
       }

       if ($rollover == "Y")
       {
         $agg_risk++;
       }

       $sql = "INSERT INTO RDDF_OUT (waypoint1, latitude1, longitude1,
lbo1, speed1,
                                    waypoint2, latitude2, longitude2,
lbo2, speed2,
                                    s, alpha1, alpha2, beta,
                                    speedm, radiusm, lbom, radius,
rollover)
                            VALUES ($waypoint1, $latitude1,
$longitude1, $lbo1, $speed1,
                                    $waypoint2, $latitude2,
$longitude2, $lbo2, $speed2,
                                    $s, $alpha1, $alpha2, $beta,
                                    $speedm, $radiusm, $lbom,
$radius, \"$rollover\")";
       mysql_query($sql);
       }
     }
  $latitude_center = ($latitude_max + $latitude_min) / 2;
  $latitude_bounds = $latitude_max - $latitude_min;
  $longitude_center = ($longitude_max + $longitude_min) /2;
  $longitude_bounds = $longitude_max - $longitude_min;
//Map footer.
  if ($debug)
  {
    echo
        "
          ], \"#ff0000\", 2);
          map.setCenter(new GLatLng($latitude_center,
$longitude_center), 8);
          map.addOverlay(polyLine);
        }
      }
    </script>
  </head>
  <body onload=\"initialize()\" onunload=\"GUnload()\">
    <div id=\"content-top\"></div>
    <div id=\"content-map\"></div>
    <div id=\"content-middle\">";
  }
  else
  {
    echo
 "
  </head>
  <body>
    <div id=\"content-top\"></div>
    <div id=\"content-middle\">";
  }
  echo
    "
      <h1>
        Form values:
      </h1>
      <p>
        Static Stability Factor (SSF) [default = 1.02]: $ssf
      </p>
      <p>
        Reportable speed, in miles per hour (mph) [default = 20]: $mph
      </p>
      <p>
        Reportable change in bearing [default = 10]: $angle
      </p>
      <h1>
        Map boundaries:
      </h1>
      <p>
        Latitude (maximum, minimum, map center, bounds): $latitude_max,
$latitude_min, $latitude_center, $latitude_bounds
      </p>
      <p>
        Longitude (maximum, minimum, map center, bounds):
$longitude_max, $longitude_min, $longitude_center, $longitude_bounds
      </p>
      <h1>
        Aggregate data:
      </h1>
      <p>
        Number of waypoints: $number.
      </p>
      <p>
        Overall course length: $length (meters).
      </p>
      <p>
        Total number of course segments for which the speed exceeds the
reportable speed: $agg_mph.
      </p>
      <p>
        Total number of intersections at which the change in bearing
exceeds the reportable change in bearing: $agg_angle.
      </p>
      <p>
        Total number of waypoints at which there is a rollover risk:
$agg_risk.
      </p>
    </div>
    <div id=\"content-bottom\"></div>
  </body>
</html>\n";
//End map footer.
  }
  else
  {
    echo "Error: " . $_FILES['nameFile']['error'];
    die("<p>The file upload was unsuccessful</p>");
  }
  exit();

        function inverse($waypoint1, $latitude1, $longitude1,
$waypoint2, $latitude2, $longitude2) {
//////////Apply the Vincenty formula to determine the distance between
waypoint1 and waypoint2, above.
//////////Constants for WGS84 ellipsoid:
          $a = 6378137.0;
          $b = 6356752.3142;
          $f = 1/298.257223563; //this is a deviation from the Vincenty
formula, for which: f, flattening = (a - b) / a = 1/298.257222933.
//////////Calculate derived variables from given variables:
          $L = deg2rad(($longitude2 - $longitude1));
          $Usub1 = atan((1 - $f) * tan(deg2rad($latitude1)));
          $Usub2 = atan((1 - $f) * tan(deg2rad($latitude2)));
          $sinUsub1 = sin($Usub1);
          $cosUsub1 = cos($Usub1);
          $sinUsub2 = sin($Usub2);
          $cosUsub2 = cos($Usub2);
//////////Step 13:
          $lambda = $L; //first approximation
          $lambdaprime = 2 * pi();
          $iterations = 0;

//          echo print_constants($waypoint1, $waypoint2, $a, $b, $f,
$L, $latitude1, $latitude2, $Usub1, $Usub2, $sinUsub1, $cosUsub1,
$sinUsub2, $cosUsub2, $lambda);
```

```
          while (abs($lambda - $lambdaprime) > 1.0e-12) {
//////////Step 14:
            $sin2sigma = ($cosUsub2 * sin($lambda)) * ($cosUsub2 *
sin($lambda)) + ($cosUsub1 * $sinUsub2 - $sinUsub1 * $cosUsub2 *
cos($lambda)) * ($cosUsub1 * $sinUsub2 - $sinUsub1 * $cosUsub2 *
cos($lambda));
            $sinsigma = sqrt($sin2sigma);
            if ($sinsigma == 0) {
              echo _p("Warning: co-incident points (sin &#x03C3; =
$sinsigma).");//Warn on co-incident points.
              break;
            }
//////////Step 15:
            $cossigma = $sinUsub1 * $sinUsub2 + $cosUsub1 * $cosUsub2 *
cos($lambda);
//////////Step 16:
            $tansigma = $sinsigma / $cossigma;
            $sigma = atan2($sinsigma, $cossigma);
//////////Step 17:
            $sinalpha = $cosUsub1 * $cosUsub2 * sin($lambda) /
$sinsigma;
            $sin2alpha = $sinalpha * $sinalpha;
//////////Step 18:
            $cos2alpha = 1 - $sin2alpha;
            if (is_infinite($cossigma - 2 * $sinUsub1 * $sinUsub2 /
$cos2alpha)) // equatorial line: $cos2alpha = 0.
              $costwosigmasubm = 0;
            else
              $costwosigmasubm = $cossigma - 2 * $sinUsub1 *
$sinUsub2 / $cos2alpha;
//////////Step 10:
            $C = ($f / 16) * $cos2alpha * (4 + $f * (4 - 3 *
$cos2alpha));
//////////Step 11:
            $lambdaprime = $lambda;
            $lambda = $L + (1 - $C) * $f * $sinalpha * ($sigma + $C *
$sinsigma * ($costwosigmasubm + $C * $cossigma * (-1 + 2 *
$costwosigmasubm * $costwosigmasubm)));
            $iterations++;

//            echo print_intermediates($iterations, $sin2sigma,
$sinsigma, $cossigma, $tansigma, $sigma, $sin2alpha, $sinalpha,
$cos2alpha, $costwosigmasubm, $C, $lambdaprime, $lambda);
          }
//////////Step 03:
          $u2 = $cos2alpha * ($a * $a - $b * $b) / ($b * $b);
          $A = 1 + $u2 / 16384 * (4096 + $u2 * (-768 + $u2 * (320 - 175
* $u2)));
//////////Step 04:
          $B = $u2 / 1024 * (256 + $u2 * (-128 + $u2 * (74 - 47 *
$u2 )));
//////////Step 06:
```

```
          $deltasigma = $B * $sinsigma * ($costwosigmasubm + 1 / 4 * $B
* ($cossigma * (-1 + 2 * $costwosigmasubm * $costwosigmasubm) - 1 / 6 *
$B * $costwosigmasubm * (-3 + 4 * $sin2sigma) * (-3 + 4 *
$costwosigmasubm * $costwosigmasubm)));
//////////Step 19:
          $s = $b * $A * ($sigma - $deltasigma);
//////////Step 20:
          $alpha1 = atan2(($cosUsub2 * sin($lambda)), ($cosUsub1 *
$sinUsub2 - $sinUsub1 * $cosUsub2 * cos($lambda)));
//////////Step 21:
          $alpha2 = atan2(($cosUsub1 * sin($lambda)), (-$sinUsub1 *
$cosUsub2 + $cosUsub1 * $sinUsub2 * cos($lambda)));
//////////Output the result:
//          echo print_intermediate_results($waypoint1, $waypoint2,
$u2, $A, $B, $deltasigma);
//          echo print_results($waypoint1, $waypoint2, $s, $alpha1,
$alpha2);
          return array($s, $alpha1, $alpha2);
        }

        function rollover ($latitude1, $lbo1, $speed1, $beta, $ssf,
$angle) {
          $a = 6378137.0;
          $b = 6356752.3142;
          $f = 1/298.257223563;

          $gamma = deg2rad(90 - abs($beta / 2)); //convert beta, in
degrees, to gamma (beta complementary angle) in radians
          $lbom = $lbo1 / 3.2808399; //convert lbo, in feet, to meters
          $speedm = ($speed1 * 5280) / (3600 * 3.2808399); //convert
speed, in miles per hour, to meters per second

//////////Calculate acceleration due to gravity.
          $phi = deg2rad($latitude1); //convert latitude, in degrees,
to radians
          $gammae = 9.7803253359; //theoretical (normal) gravity at the
equator (on the ellipsoid)
          $gammap = 9.8321849378; //theoretical (normal) gravity at the
pole (on the ellipsoid)
          $k = ($b * $gammap) / ($a * $gammae) - 1;
          $gravity = $gammae * (1 + $k * sin($phi) * sin($phi)) / sqrt
(1 - (1 - ($b * $b) / ($a * $a)) * sin($phi) * sin($phi));

          $radiusm = ($speedm * $speedm) / ($gravity * $ssf);
//required turn radius at speedm

          if (sin($gamma) == 1) {
            $radius = 0;
            $rollover = "N";
          } else {
            $radius = -$lbom / (sin($gamma) - 1); //maximum allowable
turn radius

            if ($radiusm > $radius)
```

```
              $rollover = "Y";
            else
              $rollover = "N";
          }
          return array($speedm, $radiusm, $lbom, $radius, $rollover);
        }
      ?>
    </div>
  </body>
</html>
```