Izračun povprečnih vrednosti meritev vetra in vetrovnih sunkov na oceanografski boji Vida

4. junij 2010

Merjene količine (vzorčenje 10Hz):

U_i	polurni ali petnajstminutni set meritev vzhodne komponente vetra
V_i	polurni ali petnajstminutni set meritev severne komponente vetra
W_i	polurni ali petnajstminutni set meritev navpične komponente vetra
N	število meritev v izbranem časovnem intervalu

Funkcija, ki izračunava statistične vrednosti iz surovih meritev se imenuje *MeanWind* in se nahaja v datoteki _wind.m. Količine, ki so v nadaljevanju označene z odebeljenimi črkami, se zapišejo v istoimenske stolpce v tabeli *buoy_means*. Funkcija je napisana v Octavu in se izvaja vsakih petnajst minut. Ob polni uri in pol polni uri izračunava tudi polurne vrednosti, sicer pa le petnajstminutne.

1 Vektorsko povprečje

$$\bar{U} = \frac{\sum U_i}{N}$$

$$\bar{V} = \frac{\sum V_i}{N}$$

$$\mathbf{vmspd} = \sqrt{\bar{U}^2 + \bar{V}^2} \tag{1}$$

$$\mathbf{vmdir} = \tan^{-1}\left(\frac{\bar{U}}{\bar{V}}\right) \cdot \frac{180}{\pi} + 90 \tag{2}$$

Navpične komponente ne upoštevamo. Pred vpisom v bazo se dobljene vrednosti korigira s statično korekcijo (glej 6).

2 Skalarno povprečje

$$v_i = \sqrt{U_i^2 + V_i^2}$$

$$\mathbf{smspd} = \frac{\sum v_i}{N} \tag{3}$$

$$\alpha_i = \tan^{-1}\left(\frac{U_i}{V_i}\right) \cdot 180\pi \tag{4}$$

$\alpha =$ najpogosteje zastopani 6° interval $(0 : 6 : 360)$

$$\mathbf{smsdir} = \alpha + 90 \tag{5}$$

Navpične komponente ne upoštevamo. Pred vpisom v bazo se dobljene vrednosti korigira s statično korekcijo (glej 6).

3 Sunki

Za izračun jakosti vetrovnega sunka vzamemo največjo izmed povprečnih vrednosti znotraj 3 s intervalov, ki si sledijo v 0.5 s razmakih. Povprečja so računana skalarno:

$$v_k = \sqrt{U_k + V_k}$$

$$v'_j = \frac{\sum_{k=1}^{K} v_k}{K}$$

$$\mathbf{maxspd} = \max[v'_{1,..,M}] \tag{6}$$

Kjer je K število meritev znotraj izbranega 3 s intervala, M pa število 3 s intervalov znotraj celotnega časovnega obdobja (ker se ti premikajo v korakih po 0.5 s, velja za polurni set: $M \approx 30 \cdot 60/0.5 - 6$).

$$\mathbf{minspd} = \min[v'_{1,..,M}] \tag{7}$$

Smer sunka, je vrednost (4) na sredini 3 s intervala, ki je bil izbran po zgoraj omenjenem postopku.

$$\mathbf{maxdir} = \alpha_{\max+90} \tag{8}$$

$$\mathbf{mindir} = \alpha_{\min+90} \tag{9}$$

Tudi vrednost sunkov, se pred vpisom korigira s statično korekcijo (glej 6).

4 Jakost vetra po Beaufortovi lestvici

Preračunavanje v bf se izvaja na podlagi vektorskega povprečja **vmspd** (1), pretvorjenega iz m/s v km/h in zaokrožnega na celo število. To se v bf pretvori v skladu s priloženo Tabelo 1. Tabela se sicer nahaja v datoteki _beaufort.dat.

bf	$\mathbf{vmspd}_{min}$ [km/h]	$\mathbf{vmspd}_{max}$ [km/h]
0	0	1
1	2	5
2	6	11
3	12	19
4	20	29
5	30	39
6	40	50
7	51	61
8	62	74
9	75	87
10	88	102
11	103	118
12	119	∞

Tabela 1: Beaufortova lestvica

5 Statistični podatki

Za nadaljno statistično obdelavo se uporabi hitrost vetra, ki je za vsako meritev posebej izračunana po naslednji enačbi:

$$v_i'' = \sqrt{U_i^2 + V_i^2 + W_i^2} \tag{10}$$

Torej se tu upošteva tudi navpična komponenta hitrosti.

5.1 meanspd

Za izračun se uporabi funkcija *mean(x)*.

$$\mathbf{meanspd} = \frac{\sum v_i''}{N} \tag{11}$$

meanspd je torej enaka skalarni povprečni hitrosti **smspd** (3), le da upošteva tudi navpično komponento. Poleg tega pri **meanspd** ne izvajamo statične korekcije.

5.2 stdspd

Za izračun standardne deviacije se uporabi funkcija *std(x)*.

$$\mathbf{stdspd} = \sqrt{\frac{\sum (v_i'' - \mathbf{meanspd})^2}{N-1}} \tag{12}$$

5.3 kurtspd

Za izračun se uporabi funkcija *kurtosis(x)*.

$$\mathbf{stdspd} = \frac{\sum(v_i'' - \mathbf{meanspd})^4}{N \cdot \mathbf{stdspd}^4} - 3 \tag{13}$$

5.4 skewspd

Za izračun se uporabi funkcija *skewness(x)*.

$$\mathbf{skewspd} = \frac{\sum(v_i'' - \mathbf{meanspd})^3}{N \cdot \mathbf{stdspd}^3} \tag{14}$$

5.5 stdx, stdy

Izračun standardne deviacije meritev v x (vzhod) in y (sever) smeri. Zopet je uporabljena funkcija *std(x)*.

$$\mathbf{stdx} = \mathrm{std}(U_i) = \sqrt{\frac{\sum(U_i - \bar{U})^2}{N-1}} \tag{15}$$

$$\mathbf{stdy} = \mathrm{std}(V_i) = \sqrt{\frac{\sum(V_i - \bar{V})^2}{N-1}} \tag{16}$$

6 Statična korekcija

Statična korekcija se izvaja le, kadar je v izbranem polurnem obdobju na voljo vsaj 900 kompas meritev in je standardna deviacija meritev vseh treh kotov manjša od 20. Oznake kotov:

$$\Omega = \text{heading};\ \Phi = \text{roll};\ \Theta = \text{pitch};$$

Pri korekciji vetrovnih maksimumov **maxspd** (6) in minimumov **minspd** (7) se uporabi povprečna vrednost kompas meritev v izbranem trisekundnem intervalu:

$$\Omega = \frac{\sum_{k=1}^{K} \Omega_k}{K} \tag{17}$$

$$\Phi = \frac{\sum_{k=1}^{K} \Phi_k}{K} \tag{18}$$

$$\Theta = \frac{\sum_{k=1}^{K} \Theta_k}{K} \tag{19}$$

Kjer je K število meritev znotraj 3 s intervala. Za korekcijo vektorskega **vmspd** (1) in skalarnega **smspd** (3) povprečja, se uporabi povprečna vrednost zgoraj navedenih kotov

čez celoten meritveni interval (30 min oz. 15 min). Torej enako, kot v (17), (18) in (19), le da je $K = N$. Iz meritev smeri in hitrosti vetra, sestavimo vektorje:

$$\vec{v} = \begin{pmatrix} \text{Spd} \cdot \cos(270 - \text{Dir}) \\ \text{Spd} \cdot \sin(270 - \text{Dir}) \\ 0 \end{pmatrix}$$

Rotacijska matrika:

$$\mathbf{A} = \begin{pmatrix} \cos(\Omega)\cos(\Phi) & \sin(\Omega)\cos(\Theta) - \cos(\Omega)\sin(\Theta)\sin(\Phi) & \sin(\Omega)\sin(\Theta) + \cos(\Omega)\cos(\Theta)\sin(\Phi) \\ -\sin(\Omega)\cos(\Phi) & \cos(\Omega)\cos(\Theta) + \sin(\Omega)\sin(\Theta)\sin(\Phi) & \cos(\Omega)\sin(\Theta) - \sin(\Omega)\cos(\Theta)\sin(\Phi) \\ -\sin(\Phi) & -\sin(\Theta)\cos(\Phi) & \cos(\Theta)\cos(\Phi) \end{pmatrix}$$

Tako so korigirane vrednosti $\vec{v}'$:

$$\vec{v}' = \mathbf{A}\vec{v} \tag{20}$$

Velikost in smer novo dobljenega vektorja $\vec{v}'$ se zapišeta v bazo namesto nekorigirane vrednosti (torej **maxspd**, **minspd**, **vmspd** ali **vmspd**).

7 Izvorna koda

```
1 %
  % -------------------------------------------------------------------
  %
3 % Copyright (c) 2008 DEZO Damir Dezeljin. All rights reserved.
  % Copyright (c) 2009 DEZO Damir Dezeljin. All rights reserved.
5 %
  % Posession of this software does not grant any rights to use,
    reproduce,
7 % modify or distribute it or to use any concept it may contain.
  %
9 % Licensed under DEZO license ("the License"); you may not use this
    software
  % unless in compliance with the License. Any use of the software
    without such
11 % license is a violation of copyright laws and may be subject to legal
    actions
  % (remedies and/or criminal prosecution).
13 %
  % NOTE:
15 % If you receive this content in error, please let us know by
    contacting
  % Damir Dezeljin (info@dezo.si) and destroy any copy you may have.
17 %
  %
  % -------------------------------------------------------------------
19
  %
  % -------------------------------------------------------------------
21 %
  % The below named procedures used for computing the wind speed statical
23 % correction were developed by NIB, Slovenina between 2002 and 2008.
  %
25 % Procedures for computing wind speed statical corrections:
  % - MbpVstat()
27 % - MbpArp()
  %
29 %
  % -------------------------------------------------------------------

  1;
31
  ## -*- texinfo -*-
33 ## @deftypefn {Function Function} {@var{rv} = } MeanWind(@var{pid}, @var{
    startTime}, @var{endTime})
  ## @end deftypefn
35 function funcRv = MeanWind(a_pid, a_startTime, a_endTime)
    global g_dbConn;
```

```
37  ClearError();
    funcRv = 1;
39
    termInterval       = 10*60;                    % Required by ARSO -
        terminating values time interval [seconds]
41  dataTimeInterval   = a_endTime - a_startTime;  % Current computing
        profile interval [seconds]
    timeMinMaxDelta    = 3*1000;                   % Interval of min / max
        computation bin intervals [milliseconds]
43  timeMinMaxDeltaInc= 500;                       % Scalling of the min /
        max computation bin intervals [milliseconds]
                                                   %  see also:
                                                      timeMinMaxDelta
45                                                 %  e.g. using 3s interval
                                                      we get bin centers at:
                                                      1.5, 4.5, 7.5, etc.
                                                   %       setting this value
                                                       to 0.5 we would lake
                                                      to get bin centers
47                                                 %       at 1.5, 2.0, 2.5,
                                                      etc., using the 3s
                                                      interval

49  % -- Get the year --
    query = sprintf("SELECT YEAR(FROM_UNIXTIME(%u))%%2000", a_startTime);
51  rv = mysql_query(g_dbConn, query);
    if (rv != 0)
53    SetError("Failed determining the data year!");
      return;
55  endif
    qRes = mysql_store_result(g_dbConn);
57  row = mysql_fetch_row(qRes);
    year = str2num(row(0));
59  mysql_free_result(qRes);

61  % -- Get RAW data --
    msecClmn  = 0;
63  uClmn     = 1;
    vClmn     = 2;
65  wClmn     = 3;
    query = sprintf(
67      "SELECT (((UNIX_TIMESTAMP(time)-%u)*1000)+msec) AS mmsec, U, V, W
           FROM 'buoy%02d'.'wind' WHERE time>=FROM_UNIXTIME(%u) and time<
           FROM_UNIXTIME(%u) AND error_code=0 AND device_error_code=0 ORDER
           BY mmsec",
        a_startTime,
69      year, a_startTime, a_endTime);
    rv = mysql_query(g_dbConn, query);
71  if (rv != 0)
      strErr = sprintf("Failed retrieving the RAW data (err: '%s')!",
          mysql_error(g_dbConn));
73    SetError(strErr);
```

```
    return;
75  endif
    qRes = mysql_store_result(g_dbConn);
77
    noRows = int32(mysql_num_rows(qRes));
79  msec    = zeros(noRows, 1);
    wnd_U   = zeros(noRows, 1);
81  wnd_V   = zeros(noRows, 1);
    wnd_W   = zeros(noRows, 1);
83  for i=1:noRows,
      row       = mysql_fetch_row(qRes);
85    msec(i)   = str2num(row(msecClmn));
      wnd_U(i)  = str2num(row(uClmn));
87    wnd_V(i)  = str2num(row(vClmn));
      wnd_W(i)  = str2num(row(wClmn));
89  endfor
    mysql_free_result(qRes);
91  if (noRows <= 0)
      funcRv = 1;
93    strErr = sprintf("number of rows = %d", noRows);
      SetError(strErr);
95    return;
    endif
97
    % -- Compute values --
99  % The UVW coordinates into asimut conversion is done at the end
    %
```

```
101 % Compute mean U,V and W wind speed vectors
    vect_U_mean = mean(wnd_U);
103 vect_V_mean = mean(wnd_V);
    % Compute mean vector speed
105 vmSpd = sqrt(vect_U_mean^2 + vect_V_mean^2);
    % Determinate mean vector direction
107 vmDir = atan2(vect_U_mean, vect_V_mean);
    % Convert to degrees
109 vmDir = vmDir * 180 / pi;
    % Convert from UVW angle to asimut
111 vmDir = vmDir + 90;
    if (vmDir > 360)
113   vmDir -= 360;
    elseif (vmDir < 0)
115   vmDir += 360;
    endif
117
    % Scalar method for wind speed computing (all vector variables have a
        prefix scal_)
119 % The UVW coordinates into asimut conversion is done at the end
    %
```

```
121   scal_wnd = sqrt(wnd_U.^2 + wnd_V.^2);
      scal_dir = atan2(wnd_U, wnd_V) * 180 / pi;
123   % Compute mean scalar speed
      scalmSpd = mean(scal_wnd);
125   % Prepare slots for scalar direction computing with hist() function (6
        degrees each slot)
      X_Bins = 3:6:357;
127   % Compute scalar mean direction
      scal_dir_hist = hist(scal_dir, X_Bins);
129   [hist_max, hist_max_location] = max(scal_dir_hist);
      scalmDir = X_Bins(hist_max_location);
131   % Convert from UVW angle to asimut
      % scalmDir = scalmDir + 60;
133   scalmDir = scalmDir + 90;
      if (scalmDir > 360)
135     scalmDir -= 360;
      elseif (scalmDir < 0)
137     scalmDir += 360;
      endif
139

141   % ------------------------------------------------
      % --- MIN / MAX wind speed and its location ---
143   % NOTE: values are computed within bins of predefined time lengths (
        default: 3 s)
      maxSpd = realmin;  maxSpdLoc = 1;      % stores MAX spd value and its
        location within msec vector
145   minSpd = realmax;  minSpdLoc = 1;      % stores MIN spd value and its
        location within msec vector
      for shiftNo = 0:(floor(timeMinMaxDelta/timeMinMaxDeltaInc)-1)
147     timeBins = ...
            ((timeMinMaxDelta/2) + (shiftNo*timeMinMaxDeltaInc)): ...      %
              shiftNo is used to shift the start interval
149         timeMinMaxDelta: ...                                           %
              the end interval is automatically adopted to
            ((dataTimeInterval*1000) - (timeMinMaxDelta/2));               %
              to the new value
151     % Find the first and last elements of the msec vector used for
          histogram computation
        %  e.g.1: for timeMinMaxDelta = 3s, shtimeMinMaxDeltaInc = 0.5s,
          shiftNo = 0 -> first el.idx of 0s offset
153     %  e.g.2: for timeMinMaxDelta = 3s, shtimeMinMaxDeltaInc = 0.5s,
          shiftNo = 1 -> first el.idx of >=0.5s offset
        %  It works similarly for the last one
155     firstIdx  = 0;
        lastIdx   = 0;
157     for i = 1:length(msec)
          limit = timeBins(1) - (timeMinMaxDelta/2);
159       if (msec(i) >= limit)
            firstIdx = i;
161         break;
          endif
```

```
163     endfor
        for i = length(msec):-1:1
165       limit = timeBins(length(timeBins)) + (timeMinMaxDelta/2);
          if (msec(i) <= limit)
167         lastIdx = i;
            break;
169       endif
        endfor
171     if (firstIdx == lastIdx)
          strErr = sprintf("Computing min / max bins time limits (min.idx: %d
              / max.idx: %d)", firstIdx, lastIdx);
173       SetError(strErr);
          return;
175     endif

177     % Compute the number of RAW data (msec-s) in each time bin
        binSizes = hist(msec(firstIdx:lastIdx), timeBins);
179
        % Compute the actual min / max values for each bin and 'remember' the
            correct ones
181     firstEl = firstIdx;
        for i = 1:length(binSizes)
183       if (binSizes(i) == 0)
            continue;
185       endif
          binSize = binSizes(i);
187       lastEl = firstEl + binSize - 1;
          val = mean(scal_wnd(firstEl:lastEl));
189       if (val > maxSpd)
            maxSpd = val;
191         maxSpdLoc = firstEl + floor(binSize/2);
          endif
193       if (val < minSpd)
            minSpd = val;
195         minSpdLoc = firstEl + floor(binSize/2);
          endif
197
          firstEl += binSize - 1;
199     endfor % for i = 1:length(binSizes)
      endfor % for shiftNo = 0:(floor(timeMinMaxDelta/timeMinMaxDeltaInc)-1)
201
      % 'normalize' the directions
203   % MAX
      maxDir = scal_dir(maxSpdLoc) + 90;
205   if (maxDir > 360)
        maxDir -= 360;
207   elseif (maxDir < 0)
        maxDir += 360;
209   endif
      % MIN
211   minDir = scal_dir(minSpdLoc) + 90;
      if (minDir > 360)
```

```
213     minDir -= 360;
      elseif (minDir < 0)
215     minDir += 360;
      endif
217   % MAX and MIN msec (time of occurence)
      maxms = msec(maxSpdLoc);
219   minms = msec(minSpdLoc);

221   % --- Beaufort computation ---
      % NOTE: the beaufort scale is in [km/h] - the convertion of vector [m/s]
          speed into [km/h] is needed
223   spd_kmph = round(vmSpd * 3.6);
      beaufort = load("_beaufort.dat");
225   beauf_spd = 0;
      for i=1:length(beaufort)
227     if ((spd_kmph>=beaufort(i,2)) & (spd_kmph<=beaufort(i,3)))
          beauf_spd=beaufort(i,1);
229     endif
      endfor
231
      % --- Statistical data ---
233   spd = sqrt(wnd_U.^2 + wnd_V.^2 + wnd_W.^2);
      mean_spd = mean(spd);
235   std_spd  = std(spd);
      kurt_spd = kurtosis(spd);
237   skew_spd = skewness(spd);

239   % SLO: terminska vrednost - 10 min
      termStartLoc = min(find(msec >= ((dataTimeInterval - termInterval) *
          1000)));
241   if (length(termStartLoc) != 1)
        SetError("Can't find 'term' start location!");
243     return;
      endif
245   % No data in 'termInterval' means there was an error
      if (termStartLoc == noRows)
247     SetError("No data in the term interval!");
        return;
249   endif
      wnd_term_U = mean(wnd_U(termStartLoc:noRows));
251   wnd_term_V = mean(wnd_V(termStartLoc:noRows));
      wnd_term_spd = sqrt(wnd_term_U^2 + wnd_term_V^2);
253
      % Determinate last 10 minutes vector direction
255   wnd_term_dir = atan2(wnd_term_U, wnd_term_V);
      % Convert to degrees
257   wnd_term_dir = wnd_term_dir * 180 / pi;
      % Convert from UVW angle to asimut
259   wnd_term_dir = wnd_term_dir + 90;
      if (wnd_term_dir > 360)
261     wnd_term_dir -= 360;
      elseif (wnd_term_dir < 0)
```

```
263       wnd_term_dir += 360;
      endif
265
      % Compute the standard deviations of wnd_U and wnd_V
267   stdx = std(wnd_U);
      stdy = std(wnd_V);
269
      % ------------------------------
271   % -- Static correction --
      % ------------------------------
273   %
      % -- Gather compass data --
275   % The compass mean value for the whole period should be already computed
        !
      query = sprintf("SELECT dirmean, rollmean, pitchmean, dirstd, rollstd,
          pitchstd, no_of_data FROM compass WHERE pid=%u", a_pid);
277   rv = mysql_query(g_dbConn, query);
      if (rv != 0)
279     strErr = sprintf("Failed retrieving the MEAN data (err: '%s')!",
            mysql_error(g_dbConn));
        SetError(strErr);
281     return;
      endif
283   % Set invalid STD compass values so the static corection computation is
      % skipped if compass data isn't available!
285   stdChead = 99.0;  stdCroll = 99.0;  stdCpitch = 99.0;
      % Set inlid number of source data for computing the mean compass values
287   noOfSrcCData = 0;
      qRes = mysql_store_result(g_dbConn);
289   if (int32(mysql_num_rows(qRes)) == 0)
        fprintf(2, "    INFO: No compass data available - skipping static
            correction!\n");
291   elseif (int32(mysql_num_rows(qRes)) != 1)
        SetError("Query returned an invalid number of rows!");
293     mysql_free_result(qRes);
        return;
295   else
        row = mysql_fetch_row(qRes);
297     meanChead   = str2num(row(0));
        meanCroll   = str2num(row(1));
299     meanCpitch  = str2num(row(2));
        stdChead    = str2num(row(3));
301     stdCroll    = str2num(row(4));
        stdCpitch   = str2num(row(5));
303     noOfSrcCData= str2num(row(6));
        mysql_free_result(qRes);
305   endif

307   % -- Static correction: BEGIN --
      vmSpdValid    = 0;
309   vmDirValid    = 0;
      scalmSpdValid = 0;
```

```
311   scalmDirValid = 0;
      maxSpdValid   = 0;
313   maxDirValid   = 0;
      minSpdValid   = 0;
315   minDirValid   = 0;
      wasScUsed     = 0;
317   % Static correction will be used if all of the following is met:
      % - STDEV() of any of heading, roll and pitch < 20
319   % - there were 900 source compass data from which the compass mean
        values
      %   were computed - this should corespond to minimum of 3 minutes of
        data
321   %   considering the compass sampling ration is 1 sample per 0.2 sec.
      if (stdChead < 20.0) && (stdCroll < 20.0) && (stdCpitch < 20.0) && (
        noOfSrcCData > 900);
323     % --- Static correction: use it!!! ---

325     % compass values for MAX wind speed: not computed - will do below
        val         = maxms;
327     minTime     = (a_startTime * 1000) + val - floor(timeMinMaxDelta/2);
        maxTime     = minTime + timeMinMaxDelta;
329     minTimeSec  = floor(minTime/1000);
        minTimeMsec = minTimeSec*1000 + mod(minTime, 1000);
331     maxTimeSec  = floor(maxTime/1000);
        maxTimeMsec = maxTimeSec*1000 + mod(maxTime, 1000);
333     query = sprintf(
            "SELECT ((UNIX_TIMESTAMP(time)*1000)+msec) AS mmsec, heading, roll
                , pitch FROM `buoy%02d`.`compass` WHERE time BETWEEN
                FROM_UNIXTIME(%u) AND FROM_UNIXTIME(%u)",
335         year,
            (minTimeSec - 1),
337         (maxTimeSec + 1));  % Extends the time limits as a subquery will
                be used to retrieve the actual data
        query = sprintf(
339         "SELECT avg(heading) AS meanheading, avg(roll) AS meanroll, avg(
                pitch) AS meanpitch FROM (%s) t WHERE mmsec>%lu AND mmsec<=%lu"
                ,
            query,
341         minTimeMsec,
            maxTimeMsec);
343     rv = mysql_query(g_dbConn, query);
        if (rv != 0)
345       strErr = sprintf("Failed retrieving the MEAN data (err: '%s')!",
              mysql_error(g_dbConn));
          SetError(strErr);
347       return;
        endif
349     qRes = mysql_store_result(g_dbConn);
        if (int32(mysql_num_rows(qRes)) != 1)
351       SetError("Query returned an invalid number of rows!");
          mysql_free_result(qRes);
353       return;
```

```
    endif
355 row = mysql_fetch_row(qRes);
    meanMaxChead  = str2num(row(0));
357 meanMaxCroll  = str2num(row(1));
    meanMaxCpitch = str2num(row(2));
359 mysql_free_result(qRes);

361 % compass values for MIN wind speed: not computed - will do below
    val         = minms;
363 minTime     = (a_startTime * 1000) + val - floor(timeMinMaxDelta/2);
    maxTime     = minTime + timeMinMaxDelta;
365 minTimeSec  = floor(minTime/1000);
    minTimeMsec = minTimeSec*1000 + mod(minTime, 1000);
367 maxTimeSec  = floor(maxTime/1000);
    maxTimeMsec = maxTimeSec*1000 + mod(maxTime, 1000);
369 query = sprintf(
        "SELECT ((UNIX_TIMESTAMP(time)*1000)+msec) AS mmsec, heading, roll
            , pitch FROM 'buoy%02d'.'compass' WHERE time BETWEEN
           FROM_UNIXTIME(%u) AND FROM_UNIXTIME(%u)",
371     year,
        (minTimeSec - 1),
373     (maxTimeSec + 1)); % Extends the time limits as a subquery will
           be used to retrieve the actual data
    query = sprintf(
375     "SELECT avg(heading) AS meanheading, avg(roll) AS meanroll, avg(
           pitch) AS meanpitch FROM (%s) t WHERE mmsec>%lu AND mmsec<=%lu"
           ,
        query,
377     minTimeMsec,
        maxTimeMsec);
379 rv = mysql_query(g_dbConn, query);
    if (rv != 0)
381   strErr = sprintf("Failed retrieving the MEAN data (err: '%s')!",
         mysql_error(g_dbConn));
      SetError(strErr);
383   return;
    endif
385 qRes = mysql_store_result(g_dbConn);
    if (int32(mysql_num_rows(qRes)) != 1)
387   SetError("Query returned an invalid number of rows!");
      mysql_free_result(qRes);
389   return;
    endif
391 row = mysql_fetch_row(qRes);
    meanMinChead  = str2num(row(0));
393 meanMinCroll  = str2num(row(1));
    meanMinCpitch = str2num(row(2));
395 mysql_free_result(qRes);

397
    % ------------------------------------------------
399 % --- Static correction computation ---
```

```
% vector speed and direction
v(1)     = vmSpd * cos((270 - vmDir) * 3.14159 / 180);
v(2)     = vmSpd * sin((270 - vmDir) * 3.14159 / 180);
v(3)     = 0;
vs(:)    = (MbpVstat(v(:), meanChead, meanCroll, meanCpitch));
scVmSpd  = sqrt(vs(1)*vs(1) + vs(2)*vs(2));
scVmDir  = 270. - atan2(vs(2),vs(1)) / 3.14159 * 180;
if (scVmDir < 0)
  scVmDir += 360;
endif
if (scVmDir > 360)
  scVmDir -= 360;
endif

% scalar speed and direction
v(1)     = scalmSpd * cos((270 - scalmDir) * 3.14159 / 180);
v(2)     = scalmSpd * sin((270 - scalmDir) * 3.14159 / 180);
v(3)     = 0;
vs(:)    = (MbpVstat(v(:), meanChead, meanCroll, meanCpitch));
scScalmSpd  = sqrt(vs(1)*vs(1) + vs(2)*vs(2));
scScalmDir  = round(270. - atan2(vs(2),vs(1)) / 3.14159 * 180);
if (scScalmDir < 0)
  scScalmDir += 360;
endif
if (scScalmDir > 360)
  scScalmDir -= 360;
endif

% MAX mean speed and direction
v(1)     = maxSpd * cos((270 - maxDir) * 3.14159 / 180);
v(2)     = maxSpd * sin((270 - maxDir) * 3.14159 / 180);
v(3)     = 0;
vs(:)    = (MbpVstat(v(:), meanMaxChead, meanMaxCroll, meanMaxCpitch));
scMaxSpd  = sqrt(vs(1)*vs(1) + vs(2)*vs(2));
scMaxDir  = 270. - atan2(vs(2),vs(1)) / 3.14159 * 180;
if (scMaxDir < 0)
  scMaxDir += 360;
endif
if (scMaxDir > 360)
  scMaxDir -= 360;
endif

% MIN mean speed and direction
v(1)     = minSpd * cos((270 - minDir) * 3.14159 / 180);
v(2)     = minSpd * sin((270 - minDir) * 3.14159 / 180);
v(3)     = 0;
vs(:)    = (MbpVstat(v(:), meanMinChead, meanMinCroll, meanMinCpitch));
scMinSpd  = sqrt(vs(1)*vs(1) + vs(2)*vs(2));
scMinDir  = 270. - atan2(vs(2),vs(1)) / 3.14159 * 180;
if (scMinDir < 0)
  scMinDir += 360;
endif
```

```
    if (scMinDir > 360)
453   scMinDir -= 360;
    endif
455
    % -- Copy values to variables that will be written to the DB --
457 vmSpdValid    = scVmSpd;
    vmDirValid    = scVmDir;
459 scalmSpdValid = scScalmSpd;
    scalmDirValid = scScalmDir;
461 maxSpdValid   = scMaxSpd;
    maxDirValid   = scMaxDir;
463 minSpdValid   = scMinSpd;
    minDirValid   = scMinDir;
465 wasScUsed     = 1;

467 else % if (stdChead > 20.0) || (stdCroll > 20.0) || (stdCpitch > 20.0)
    % -- Static correction: DO NOT use it !!! --
469 % -- Copy values to variables that will be written to the DB --
    vmSpdValid    = vmSpd;
471 vmDirValid    = vmDir;
    scalmSpdValid = scalmSpd;
473 scalmDirValid = scalmDir;
    maxSpdValid   = maxSpd;
475 maxDirValid   = maxDir;
    minSpdValid   = minSpd;
477 minDirValid   = minDir;
    wasScUsed     = 0;
479 end

481 % -- Static correction: END --

483 % -- Store computed values --
    query = sprintf(
485     "%s %s VALUES (%u, %u, %u, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %
          f, %f, %d, %d, %d, %f, %f, %f, %f)",
        "INSERT INTO wind",
487     "(pid, no_of_data, sc_used, vmspd, vmdir, smspd, smdir, maxspd,
          maxdir, minspd, mindir, meanspd, stdspd, kurtspd, skewspd, beauf,
           maxspd_ms, minspd_ms, termspd, termdir, stdx, stdy)",
        a_pid, noRows, wasScUsed,
489     vmSpdValid, vmDirValid,
        scalmSpdValid, scalmDirValid,
491     maxSpdValid, maxDirValid, minSpdValid, minDirValid,
        mean_spd, std_spd, kurt_spd, skew_spd,
493     beauf_spd,
        maxms, minms, wnd_term_spd, wnd_term_dir, stdx, stdy);
495 rv = mysql_query(g_dbConn, query);
    if (rv != 0)
497   strErr = sprintf("Storing computed values (err: '%s')!", mysql_error(
        g_dbConn));
      SetError(strErr);
499   return;
```

```
    endif
501
    funcRv = 0;
503 endfunction

505
  %
    ------------------------------------------------------------------------

507 % Computes the static corrected wind speed from u,v,w,heading,pitch,roll
  % (u,v and w in m/s, other arguments in degrees)
509 %
    ------------------------------------------------------------------------

  function vstat = MbpVstat(v, heading, roll, pitch)
511   arp = MbpArp(heading, roll, pitch);
    vstat = arp * v;
513 endfunction

515
  %
    ------------------------------------------------------------------------

517 % Computes the transformation matrix ARP needed for the wind static
      correction
  % Usage: arp(heading,roll,pitch)
519 %        (arguments in degrees)
  %
    ------------------------------------------------------------------------

521 function arp = MbpArp(heading, roll, pitch)
    conrad = pi/180.0;
523
    sin_om = sin(heading*conrad);
525   cos_om = cos(heading*conrad);
    sin_fi = sin(roll*conrad);
527   cos_fi = cos(roll*conrad);
    sin_th = sin(pitch*conrad);
529   cos_th = cos(pitch*conrad);

531   arp = zeros(3,3);

533   arp(1,1) = cos_om * cos_fi;
    arp(1,2) = sin_om * cos_th - cos_om * sin_th * sin_fi;
535   arp(1,3) = sin_om * sin_th + cos_om * cos_th * sin_fi;

537   arp(2,1) = (-1.0) * sin_om * cos_fi;
    arp(2,2) = cos_om * cos_th + sin_om * sin_th * sin_fi;
539   arp(2,3) = cos_om * sin_th - sin_om * cos_th * sin_fi;

541   arp(3,1) = (-1.0) * sin_fi;
    arp(3,2) = (-1.0) * sin_th * cos_fi;
```

```
543     arp(3,3) = cos_th * cos_fi;
    endfunction
```