LAMS Processing: Construct relative wind components from the LAMS measurements

This routine reads a netCDF file with the normal data, to which Scott Spuler has added three variables that contain the measurements of airspeed from the three beams of 3D-LAMS. It adds three new variables to that file representing the three relative-wind components as measured by LAMS but translated back to the IRS reference system. These relative-wind components are analogous to the same measurements made by the radome gust system and can be used as alternate measurements of relative wind when constructing the true wind.

The processing sequence is as follows:
1. The conventional measurements are processed by nimbus and output to a netCDF file. That file has the measurements needed to determine the components of the relative wind from the radome gust probe, TASX, the IRS attitude angles, and the CMIGITS attitude angles.
2. Scott uses TASX to determine folding and the histograms recorded by LAMS to determine three velocities, {VA, VB, VC}. which he adds to the netCDF file.
3. This routine reads the reulting netCDF file for the components listed in step 1 and also for A={VA, VB, VC}. It then performs the transformations discussed in this note: 'LAMSprocessing3Dwind.pdf' on the RAF science wiki: http://wiki.eol.ucar.edu/rafscience/LAMS%20Processing?action=AttachFile&do=view&target=LAMSprocessing3Dwind.pdf . Some sections of that memo are reproduced below to indicate how they relate to the segments of the code that follow.

In [214]:
```
# initialization:
import numpy as np    # array manipulations
import numpy.ma as ma # masked arrays -- masks used to track and propagate missin
from pylab import *   # included to be able to plot easily. Some plots are genera
from netCDF4 import Dataset  # used to read and write the netCDF file
import os             # for copying netCDF file before changing it
```

Constants that define the geometry:

From Scott Spuler's email of 19 Sept 2013:

"Here are the distances (in meters) from LAMS C-MIGITS to the three GV IMU locations.
Coordinates are z-longitudinal (fore is positive), x-horizontal (port is positive), y-vertical (up is positive)

```
    DZ    DX     DY
```

1 -10.305 6.319 -1.359
2 -10.394 7.452 -1.486
3 -10.470 6.319 -1.359

Also, from the laser tracker survey, the beams theta and phi angles are A: 34.95 deg, 0.08 deg; B: 34.96 deg, 119.93 deg; C: 35.01 deg, -120.08 deg "

In the 'LL' statement below, I have changed the signs of the distances. These are obviously distances from the IMUs to the CMIGITS.

In [215]:
```
# The polar and azimuthal angles of the three beams wrt the GV longitudinal axis
Theta = np.array ([34.95, 34.96, 35.01]) * np.pi / 180. # values in radians
Phi = np.array ([180.08, -60.07, 59.92]) * np.pi / 180. #  "  "
# also need the distances from the IRS to LAMS: (x,y,z)
LL = ma.array ([-10.305, -6.319, 1.359])                # see Scott's email, reco
# unit vectors along beams are then:
#   a[i] = [cos(Theta[i]), -sin(Theta[i])*sin(Phi[i]), sin(Theta[i])*cos(Phi[i])]
# and the dot products with the (i,j,k) unit vectors give the direction cosine ma
S = np.array ([[cos(Theta[0]), -sin(Theta[0])*sin(Phi[0]), sin(Theta[0])*cos(Phi[
               [cos(Theta[1]), -sin(Theta[1])*sin(Phi[1]), sin(Theta[1])*cos(Phi[
               [cos(Theta[2]), -sin(Theta[2])*sin(Phi[2]), sin(Theta[2])*cos(Phi[
Si = linalg.inv (S)  # calculate the inverse of S
```

Acquire the data from the conventional wind-sensing system

In [ ]:
```
# copy the netCDF file to another before opening, to be able to write to the new 
InputFile = 'IDEASGrf04'
DataDirectory = '/Data/LAMS/'
FullFileName = DataDirectory + InputFile + 'SMS.nc'
RevisedFileName = DataDirectory + InputFile + 'WAC.nc'
copyCommand = 'cp ' + FullFileName + ' ' + RevisedFileName
print FullFileName
print RevisedFileName
print copyCommand
os.system(copyCommand)
#!cp /Data/LAMS/IDEASGrf04SMS.nc /Data/LAMS/IDEASGrf04WAC.nc  # shell command
netCDFfile = Dataset (RevisedFileName, 'a')
DLEN = len (netCDFfile.dimensions['Time'])

# variables from netCDF file needed for processing:
varNames = ['TASX', 'THDG', 'PITCH', 'ROLL', 'AKRD', 'SSLIP', 'Time', \
            'CPITCH_LAMS', 'CROLL_LAMS', 'CTHDG_LAMS', 'VNSC', 'VEWC', 'VSPD', \
            'BEAM1_speed', 'BEAM2_speed', 'BEAM3_speed']
# if any are not found, routine will crash when they are used
Vars = []   # set up list used temporarily for transfer to masked arrays
for h in varNames:
    for v in netCDFfile.variables:
        if h == v:
            Vars.append (netCDFfile.variables[v][:])

# set up masked arrays with these variables:
TASX =        ma.array (Vars[0], fill_value=-32767.)

# the following segment is to handle high-rate data, which need to be flattened
Shape = netCDFfile.variables['TASX'].shape
DL = 1
SampleRate = 1
if len (Shape) > 1:
    SampleRate = Shape[1]
for i in range (0, len (Shape)):
    DL *= Shape[i]
TASX.shape = (DL)

THDG =        ma.array (Vars[1].reshape (DL), fill_value=-32767.)
PITCH =       ma.array (Vars[2].reshape (DL), fill_value=-32767.)
ROLL =        ma.array (Vars[3].reshape (DL), fill_value=-32767.)
AKRD =        ma.array (Vars[7].reshape (DL), fill_value=-32767.)
SSLIP =       ma.array (Vars[8].reshape (DL), fill_value=-32767.)
Time =        ma.array (Vars[9], fill_value=-32767.)  # Time is not high-rate
CPITCH =      ma.array (Vars[10].reshape (DL), fill_value=-32767.)
CROLL =       ma.array (Vars[11].reshape (DL), fill_value=-32767.)
CTHDG =       ma.array (Vars[12].reshape (DL), fill_value=-32767.)
VNSC =        ma.array (Vars[13].reshape (DL), fill_value=-32767.)
VEWC =        ma.array (Vars[14].reshape (DL), fill_value=-32767.)
VSPD =        ma.array (Vars[15].reshape (DL), fill_value=-32767.)
BEAM1speed = ma.array (Vars[16].reshape (DL), fill_value=-32767.)
BEAM2speed = ma.array (Vars[17].reshape (DL), fill_value=-32767.)
BEAM3speed = ma.array (Vars[18].reshape (DL), fill_value=-32767.)
BEAM1speed = ma.masked_where (BEAM1speed == -32767., BEAM1speed)
BEAM2speed = ma.masked_where (BEAM2speed == -32767., BEAM2speed)
BEAM3speed = ma.masked_where (BEAM3speed == -32767., BEAM3speed)


# temporary: try using CTHDG values advanced by 1 s:
```

Retrieve the conventional relative wind

In [203]:
```
# Construct the relative-wind array, a DL x 3 array with
# first dimension the time index and 2nd components u,v,w. This
# is the relative wind as measured by the radome and pitot-tube system.
# It is included here to be able to compare to the corresponding LAMS results
UR = TASX
VR = TASX * ma.tan (SSLIP * pi/180.)
WR = TASX * ma.tan (AKRD * pi/180.)

# relative-wind array: RWR as measured by radome gust system
RWR = transpose (ma.array([UR,VR,WR], fill_value=-32767.))
```

Get the rotation-rate array for use below

Bulletin 23 gives the expected addition to the relative wind $\vec{v}_a$ caused by rotation:

$$\vec{v}_a^{\,\prime} = \vec{v}_a + \vec{\Omega}_p \times \vec{R}$$

where $\vec{\Omega}_p$ is the angular rotation rate or angular velocity of the aircraft. Here $\vec{R}$ is the vector from the aircraft reference location at the IRS to the LAMS, represented by the location of the CMIGITS.

In [204]:
```
# get rotation-rate-vector array:
Omega = ma.array ([ROLL, PITCH, THDG], fill_value = -32767.) * pi / 180.
Omega = np.diff (Omega.T, axis=0) * SampleRate    # this differentiates step-wise
Omega[:,2][Omega[:,2] > pi] -= 2.*pi              # handle where THDG goes throug
Omega[:,2][Omega[:,2] < -pi] += 2.*pi
Omega = np.append (Omega, ma.zeros((1,3)), axis=0) # to match dimensions of other

# get false contribution to relative wind from rotation:
F = ma.array (np.cross (Omega, LL), fill_value=-32767.)
```

Transform relative wind to the Earth-reference system:

For a description of the procedure that follows, see 'LAMSprocessing3Dwind.pdf' on the RAF science wiki: http://wiki.eol.ucar.edu/rafscience/LAMS%20Processing?action=AttachFile&do=view&target=LAMSprocessing3Dwind.pdf
.

Get the three-beam measurements of airspeed:

In [205]:
```
A = transpose (ma.array ([BEAM1speed, BEAM2speed, BEAM3speed], fill_value=-32767.
```

Get the u,v,w components at LAMS and correct for rotation rate:

In [206]:
```
RW = transpose (ma.dot (Si, A.T)) - F  # F is the rotation-rate correction
```

Rotate to the Earth-reference system:

If $\{\phi, \theta, \psi\}$ are respectively roll, pitch, and heading, the needed transformation matrices are:

$$T_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$T_2 = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$T_3 = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In [207]:
```
#RWT = np.dot (T3, np.dot(T2, np.dot(T1, RW))) -- I wish! Try later to get an arr
RWT = ma.empty (shape (RW))
Cradeg = pi / 180.  # convert to radians from degrees
CR = CROLL * Cradeg
CP = CPITCH * Cradeg
CH = CTHDG * Cradeg
T1 = ma.array ([One,Z,Z, Z,cos(CR),-sin(CR), Z,sin(CR),cos(CR)])  # rotation abou
T2 = ma.array ([cos(CP),Z,sin(CP), Z,One,Z, -sin(CP),Z,cos(CP)])  # rotation abou
T3 = ma.array ([cos(CH),-sin(CH),Z, sin(CH),cos(CH),Z, Z,Z,One])  # rotation abou
T1.shape = (3,3,len(CR))
T2.shape = (3,3,len(CP))
T3.shape = (3,3,len(CH))
for i in range (0,DL):
    RWT[i,:] = ma.dot (T3[...,i], ma.dot (T2[...,i], ma.dot (T1[...,i], RW[i,:])))
```

```
('GW[IX] = ', masked_array(data = [-1.8553613738666428 -20.592056572322136 0.0384
             mask = [False False False],
       fill_value = 1e+20)
)
```

Find the wind in the Earth reference frame:

RWT is now the relative wind in the earth reference frame, but still with {x,y,z} {north, east, down}. These components can now be used to find the Earth-relative wind by subtracting from them the corresponding velocity components of the aircraft, {VNSC, VEWC, VSPD}. Subtraction is needed because wind is characterized by the direction **from which** the wind blows, while the aircraft motion is characterized by motion toward the corresponding direction. Subtracting the aircraft velocity then gives

```
        (-air motion wrt Earth) = (-air motion wrt aircraft) - (aircraft motion wrt Earth)

        (Earth-relative wind)   = (LAMS-
    measured wind)       - (aircraft motion wrt Earth)
```

For the vertical wind: positive is upward and LAMS measures positive upward as positive incoming along the z axis. Upward motion of the aircraft should be *added* to this, so the sign of VSPD must be reversed:

In [ ]:
```

RWG = RWT  # relative wind wrt ground; save for use later
GW = RWG - transpose (ma.array([VNSC, VEWC, -1.*VSPD], fill_value=-32767.))
```

Transform LAMS-relative wind to the aircraft reference frame:

For direct comparison to relative wind in the aircraft (Honeywell) reference frame, transform back to that frame. For this transformation, the signs of the attitude angles are reversed and the order in which the rotations is applied is also reversed.

This section is not needed to get Earth-relative wind, so later it can be omitted. This is useful in the development stage, though.

In [ ]:
```
CR = -1. * ROLL * Cradeg    # negative sign because now doing the transform in the
CP = -1. * PITCH * Cradeg
CH = -1. * THDG * Cradeg
T1 = ma.array ([One,Z,Z, Z,cos(CR),-sin(CR), Z,sin(CR),cos(CR)])  # rotation about
T2 = ma.array ([cos(CP),Z,sin(CP), Z,One,Z, -sin(CP),Z,cos(CP)])  # rotation about
T3 = ma.array ([cos(CH),-sin(CH),Z, sin(CH),cos(CH),Z, Z,Z,One])  # rotation about
T1.shape = (3,3,len(CR))
T2.shape = (3,3,len(CP))
T3.shape = (3,3,len(CH))
for i in range (0,DL):  # note reversal in order of transformations
    RWT[i,:] = ma.dot (T1[...,i], ma.dot (T2[...,i], ma.dot (T3[...,i], RWT[i,:]))
```

Create the netCDF variables and write the file

In [208]:
```
# create new netCDF variables for {u,v,w} and for {WD, WS, WI}:
if len (Shape) < 2:
    ULAMSCDF = netCDFfile.createVariable ('U_LAMS', 'f4', ('Time'), fill_value=-3
    VLAMSCDF = netCDFfile.createVariable ('V_LAMS', 'f4', ('Time'), fill_value=-3
    WLAMSCDF = netCDFfile.createVariable ('W_LAMS', 'f4', ('Time'), fill_value=-3
    WDLAMSCDF = netCDFfile.createVariable ('WD_LAMS', 'f4', ('Time'), fill_value=
    WSLAMSCDF = netCDFfile.createVariable ('WS_LAMS', 'f4', ('Time'), fill_value=
    WILAMSCDF = netCDFfile.createVariable ('WI_LAMS', 'f4', ('Time'), fill_value=
else:                   # later, generalize this to spsXX
    ULAMSCDF = netCDFfile.createVariable ('U_LAMS', 'f4', ('Time', 'sps50'), fill_
    VLAMSCDF = netCDFfile.createVariable ('V_LAMS', 'f4', ('Time', 'sps50'), fill_
    WLAMSCDF = netCDFfile.createVariable ('W_LAMS', 'f4', ('Time', 'sps50'), fill_
    WDLAMSCDF = netCDFfile.createVariable ('WD_LAMS', 'f4', ('Time', 'sps50'), fi
    WSLAMSCDF = netCDFfile.createVariable ('WS_LAMS', 'f4', ('Time', 'sps50'), fi
    WILAMSCDF = netCDFfile.createVariable ('WI_LAMS', 'f4', ('Time', 'sps50'), fi

# add attributes
ULAMSCDF.units = 'm/s'
WLAMSCDF.units = 'm/s'
WLAMSCDF.units = 'm/s'
WDLAMSCDF.units = 'degrees wrt true north'
WSLAMSCDF.units = 'm/s'
WILAMSCDF.units = 'm/s'
ULAMSCDF.long_name = 'LAMS-derived longitudinal component of the relative wind (f
VLAMSCDF.long_name = 'LAMS-derived lateral component of the relative wind (from s
WLAMSCDF.long_name = 'LAMS-derived vertical component of the relative wind (from
WDLAMSCDF.long_name = 'LAMS-derived wind direction from true north'
WSLAMSCDF.long_name = 'LAMS-derived wind speed)'
WILAMSCDF.long_name = 'LAMS-derived vertical wind'
ULAMSCDF.standard_name = 'u from LAMS'
VLAMSCDF.standard_name = 'v from LAMS'
WLAMSCDF.standard_name = 'w from LAMS'
WSLAMSCDF.standard_name = 'WD_LAMS'
WDLAMSCDF.standard_name = 'WS_LAMS'
WILAMSCDF.standard_name = 'WI_LAMS'
#print (RWT[0,np.unravel_index (RWT[0,:].argmax (), RWT[0,:].shape)])

# is actual_range attribute needed?-- not supplied here
ULAMSCDF.Category = 'Winds'
VLAMSCDF.Category = 'Winds'
WLAMSCDF.Category = 'Winds'
WDLAMSCDF.Category = 'Winds'
WSLAMSCDF.Category = 'Winds'
WILAMSCDF.Category = 'Winds'
ULAMSCDF.DataQuality = 'Preliminary'
VLAMSCDF.DataQuality = 'Preliminary'
WLAMSCDF.DataQuality = 'Preliminary'
WDLAMSCDF.DataQuality = 'Preliminary'
WSLAMSCDF.DataQuality = 'Preliminary'
WILAMSCDF.DataQuality = 'Preliminary'
ULAMSCDF.Dependencies = '9 BEAM1speed BEAM2speed BEAM3speed ROLL PITCH THDG CROLL
VLAMSCDF.Dependencies = '9 BEAM1speed BEAM2speed BEAM3speed ROLL PITCH THDG CROLL
WLAMSCDF.Dependencies = '9 BEAM1speed BEAM2speed BEAM3speed ROLL PITCH THDG CROLL
WDLAMSCDF.Dependencies = '11 BEAM1speed BEAM2speed BEAM3speed ROLL PITCH THDG CRO
WSLAMSCDF.Dependencies = '11 BEAM1speed BEAM2speed BEAM3speed ROLL PITCH THDG CRO
WILAMSCDF.Dependencies = '10 BEAM1speed BEAM2speed BEAM3speed ROLL PITCH THDG CRO
# put the unmasked data in the netCDF file, with masked values replaced by -32767
WDL = ma.arctan2 (GW[:,1], GW[:,0]) / Cradeg  # wind direction based on LAMS
WDL[WDL < 0.] += 360.
if len (Shape) < 2:
```

In [213]:
```
print ('Reached end of routine')
```

```
Reached end of routine
```

Find how the result differs from the starting values

In [209]:
```
# now get standard deviations of the differences between the radome-based and LAM
UDiff = RWT[:,0] - RWR[:,0]
VDiff = RWT[:,1] - RWR[:,1]
WDiff = RWT[:,2] - RWR[:,2]
# not sure if the following protection against NAN is still needed with masked ar
print (' u standard deviation is ', np.std(UDiff[np.isnan(UD) ==  False]))    # m
print (' v standard deviation is ', np.std(VDiff[np.isnan(VD) ==  False]))    # t
print (' w standard deviation is ', np.std(WDiff[np.isnan(WD) ==  False]))
#i = np.unravel_index (UD.argmax (), UD.shape) # find the maximum value (or repla
```

```
(' u standard deviation is ', 51.83340923146455)
(' v standard deviation is ', 60.140992453918919)
(' w standard deviation is ', 60.720430913659335)
```

The following cells are temporary work space and can be deleted or changed

In [210]:
```
# work space: can delete this and preceding cell
clf ()
plot (RWT[6428:6668,2], label = 'RWTw')
plot (RWR[6428:6668,2], label = 'RWRw')
#plot (100.*DP[6428:6668], label = 'DP*100')
DF = THDG[6428:6668]-CTHDG[6428:6668]
#plot (DF, label = 'diff')
ylim ([-5.,15.])
legend ()
#show ()
```

In [212]:
```
print GW[IX]
print (WDL[IX], WSL[IX])
plot (GW[6400:6800,2])
#show ()
```

```
[-1.8553613738666428 -20.592056572322136 0.038428955673914134]
(264.85150481965178, 20.675472417948559)
```