Fluent User Defined Function: WindCube_comp_sim

M T Stickland, S Fabre, T Scanlon

February 2011

University of Strathclyde NORSEWInD Report UoSNW013

Department of Mechanical Engineering
University of Strathclyde
75 Montrose St, Glasgow
G1 1XJ

Fax +44 141 552 5105 Email: matt.stickland@strath.ac.uk
Tel: +44 141 548 2842

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

ABSTRACT

This report describes the Fluent User Defined Function *WindCube_comp_sim* used to interrogate FLUENT a Fluent data set by simulating the operation of a LeoSphere Windcube LiDAR. The user defined function is contained in the program *lidar.c* (version 1.06) which has been written in the C programming language.

This report contains a listing of the user defined function, describes its method of operation and presents a validation of the analysis process.

The report also includes a description of the output data file formats.

1 TABLE OF CONTENTS

2 Introduction

Whilst the CFD simulation of the flows, over the many different platforms which are used as mounts for the various LiDAR and SoDARs employed by NORSEWInD, can give an estimation of the distortion of the flow field on point measurement quite readily. The effect of the flow distortion on the output of the LiDARs and SoDARs requires a more complex analysis. This analysis requires the CFD data to be interrogated as if a LiDAR or SoDAR was actually present in the simulation.

This report describes the Fluent User Defined Function (UDF) *Windcube_full_sim* which can be used to interrogate FLUENT data and simulate the operation of a Leosphere Windcube LiDAR.

To simulate the operation of a Windcube LiDAR mounted on a platform the results of a CFD simulation of the flow are interrogated by a User Defined Function*, Windcube_full_sim*. The UDF interrogates the fluent data set at the measurement points of the Windcube LiDAR, rotates the vector data from the CFD coordinate system into the LiDAR coordinate system and calculates the Velocity in the line of sight of the laser beam (VloS). From the Vlos data the UDF calculates the velocity vector data that would be returned by the LiDAR at the interrogation height. The UDF also returns the velocity vector data at the point directly above the LiDAR at the interrogation height.

To assess the effect of the flow distortion on the measurements made by the LiDAR and to allow correction of the data should it be required a range of correction factors are determined from the LiDAR and point measurements.

This report contains a listing of the UDF and describes its method of operation.

The report also includes a description of the correction factors and the output data file formats.

A verification of the procedure is presented based on the data from the simulation of the Horns Rev 2 platform.

3 Analysis Technique

3.1 Windcube method of operation

The Leosphere Windcube measures the component of the velocity vector in the direction of a laser beam at four points at a known height above the system, figure 1.

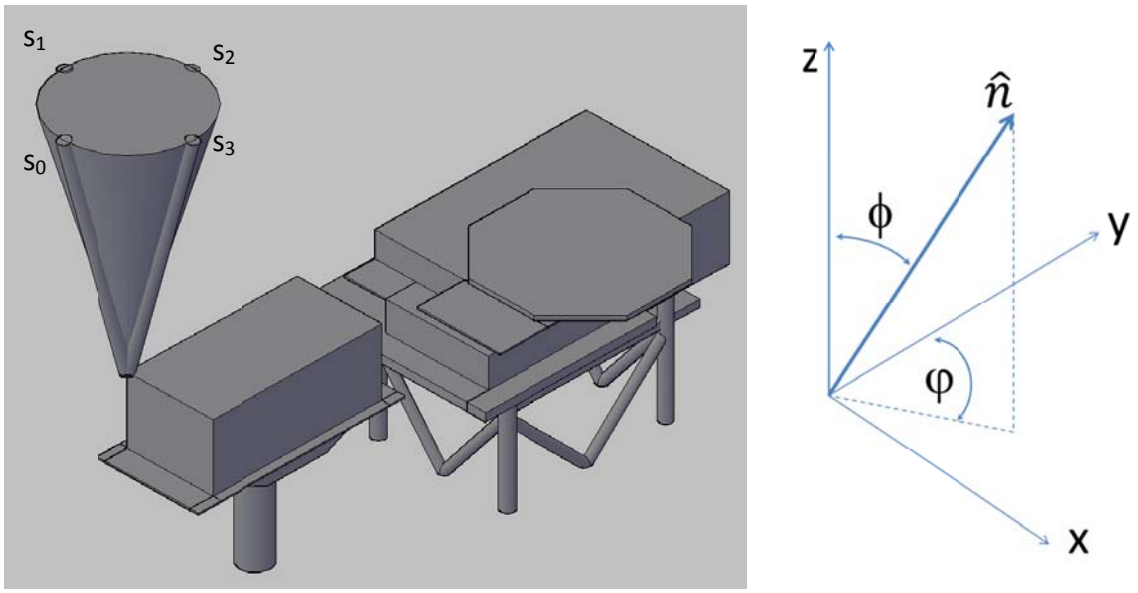


Figure 1: Measurement points above wind cube LiDAR and coordinate system

The unit vector in the direction of the laser beam is given by equation 1.

$$\vec{n}_{(x,y,z)} = (\sin\phi \sin\varphi,\ \sin\phi\cos\varphi,\ \cos\phi) \qquad \text{equation 1}$$

The projection of the wind vector on to the unit vector calculated by the dot product of the velocity vector and the unit vector gives the component of the wind velocity in the direction of the laser beam, V_{los}, equation 2.

$$V_{los} = u\sin\phi\sin\varphi + v\sin\phi\cos\varphi, + w\cos\varphi \qquad \text{equation 2}$$

Harris (2009) also gives V_{los} but in the Cartesian coordinate system, equation 3, where x, y and z are the Cartesian coordinates of the measurement point from the laser.

$$V_{los} = \frac{ux+vy+wz}{\sqrt{x^2+y^2+z^2}} \qquad \text{equation 3}$$

The wind cube LiDAR takes the four line of site velocities at the four ordinal points of a circular scan, $\varphi=0^{o}, 90^{o}, 180^{o}, 270^{o}$ where the angle, φ, is taken positive clockwise from the y

axis. ϕ is the cone angle of the Windcube, 30°, if the line of sight velocity at these four points are taken as s_0, s_1, s_2 and s_3 respectively then from equations 4 to 7

$$s_0 = v \sin \emptyset + w \cos \emptyset \quad \text{equation 4}$$

$$s_1 = u \sin \emptyset + w \cos \emptyset \quad \text{equation 5}$$

$$s_2 = -v \sin \emptyset + w \cos \emptyset \quad \text{equation 6}$$

$$s_3 = -u \sin \emptyset + w \cos \emptyset \quad \text{equation 7}$$

taking the V_{los} from diametrically opposite sides of the scan and summing

$$s_0 + s_2 = v \sin \emptyset + w \cos \emptyset - v \sin \emptyset + w \sin \emptyset \quad \text{equation 8}$$

$$s_0 + s_2 = 2w \cos \emptyset \quad \text{equation 9}$$

$$w = \frac{s_0 + s_2}{2 \cos \emptyset} \quad \text{equation 10}$$

using the same process the other components of the wind vector can be determined from the line of sight velocity.

$$v = \frac{s_0 - s_2}{2 \sin \emptyset} \text{e} \quad \text{equation 11}$$

$$u = \frac{s_1 + s_3}{2 \cos \emptyset} \quad \text{equation 12}$$

Utilising a CFD simulation of the flow over a platform and applying the technique discussed above it is possible to simulate the measurement that would be made by a Windcube LiDAR. If the coordinate systems of the LiDAR and the CFD simulation were exactly the same then the analysis would be straight forward. However, various reasons this is not always possible and therefore the measurement positions of the LiDAR need to be transformed into the coordinate system of the simulation and the wind vector data from the CFD simulation at these points need to be transformed into the coordinate system of the LiDAR.

3.2 Transformation Matrix

To simplify the process and to make the UDF that performs the analysis as flexible as possible a single transformation matrix (TM) was used. The TM performs a rotation about

the Z axis in a right handed coordinate system which creates an anticlockwise rotation for a positive angle. The TM is shown in equation 13

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \quad \text{equation 13}$$

3.3 Analysis process

The analysis process described here uses the Horns Rev 2 platform as an example but, providing that the location of a LiDAR on a platform is known and the angles β and γ are defined, then the UDF can be used on any installation of a Windcube.

Initially the location of the LiDAR in the CFD coordinate system and its orientation relative to the coordinate system γ are required as shown in figure 2. The LiDAR will usually be aligned with true north so this is the angle between the platform and true north.

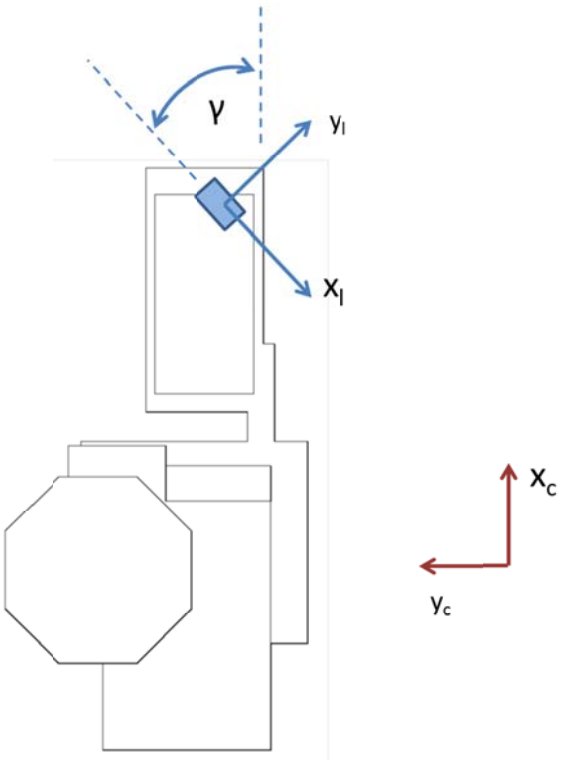


Figure 2: CFD and LiDAR coordinate systems

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

To simulate the wind approaching the platform at different angles several simulations are run with the platform rotated through an angle, β, relative to the CFD axis system. This angle is also required to be known, figure 3.

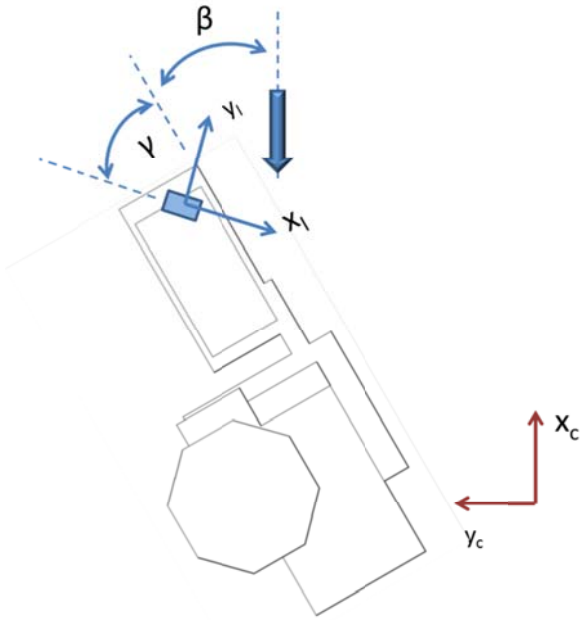


Figure 3: Rotation of platform relative to wind vector

For the given interrogation height the four measurement locations s_0 to s_3 are calculated relative to the CFD coordinate system origin, figure 4.

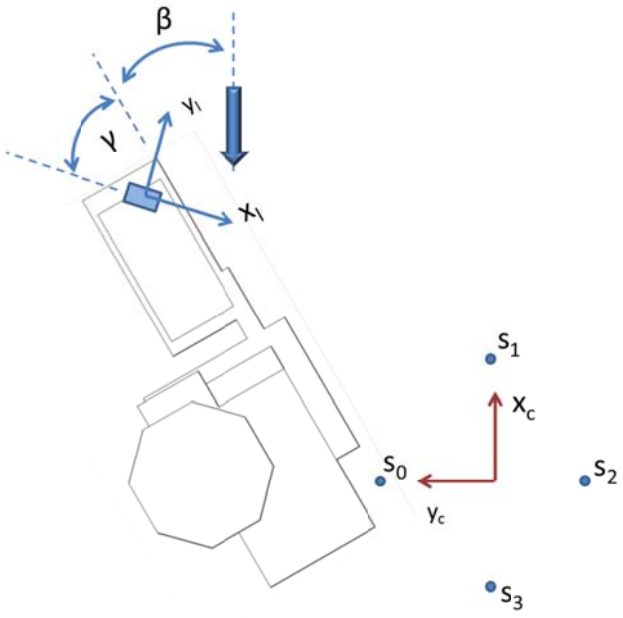


Figure 4: LiDAR measurement locations in CFD coordinate system

The measurement locations are rotated to align with the LiDAR coordinate system, figure 5

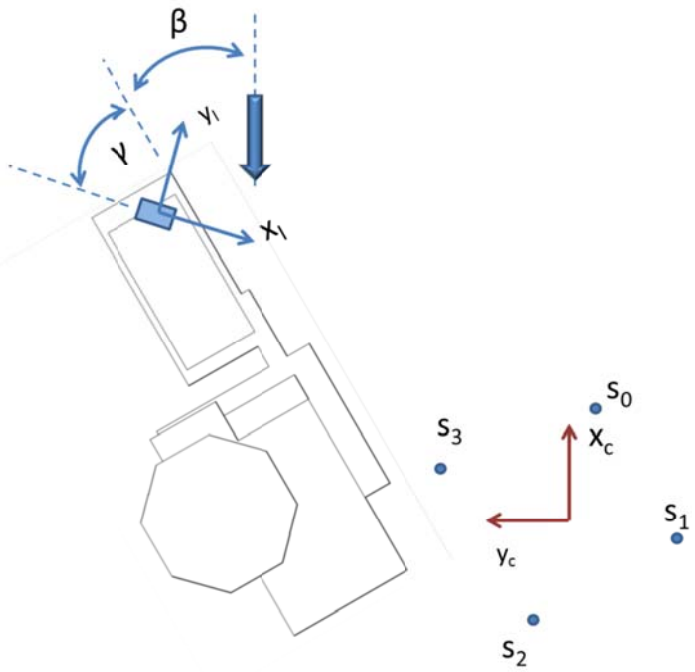


Figure 5: LiDAR measurement locations rotated into LiDAR coordinate system

The measurement locations are then translated from the CFD coordinate system into the LiDAR coordinate system, figure 6.

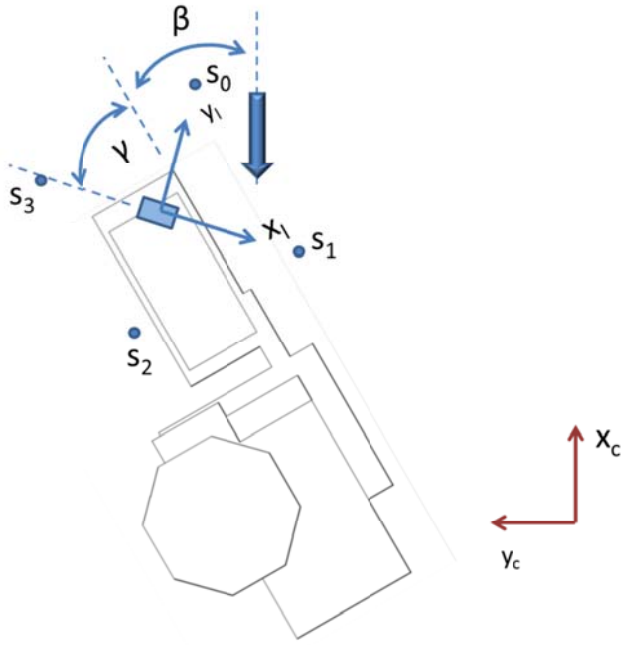


Figure 6; measurement points in LiDAR coordinate system

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

The CFD data set is interrogated at points s_0 to s_3 to find the components of the velocity vector u, v, w at each point. The velocity vector data are in the CFD coordinate system and therefore need to be rotated into the LiDAR coordinate system. If the CFD vector data is defined X positive in the upstream direction then an initial rotation of 180^o is required take the vector data from then CFD to LiDAR coordinate system, figure 7.

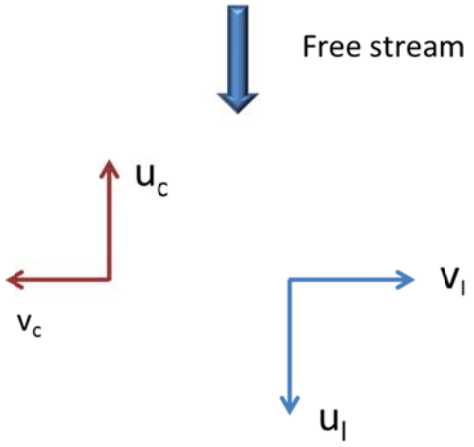


Figure 7, 180 degree rotation of velocity to LiDAR coordinate system

The velocity data then needs to be rotated to take into account any rotation of the platform relative to the wind vector, figure 8.

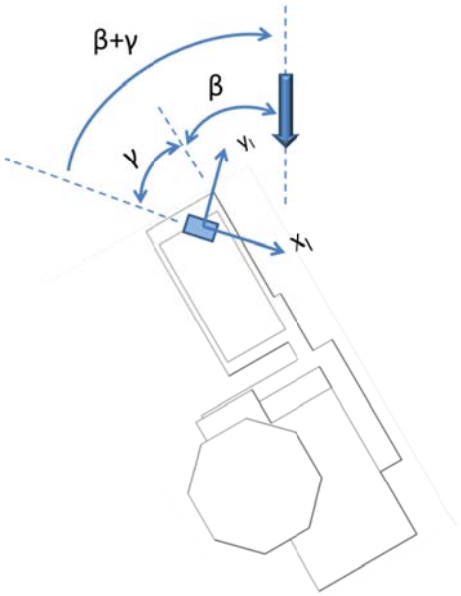


Figure 8: Velocity vector rotation to LiDAR Coordinate system for rotated platform

Note that in figure 8 that the rotation of the free stream vector relative to the LiDAR is clockwise and therefore negative in the right hand coordinate system used in the rotation matrix shown in equation 13.

If the LiDAR is not aligned with north there is a software setting that can be applied that will rotate the measured velocity vector to a coordinate system aligned with north. If this angle is defined as ε, shown in figure 9, then the measured velocity vector is rotated by an angle –ε using the rotation matrix defined in equation

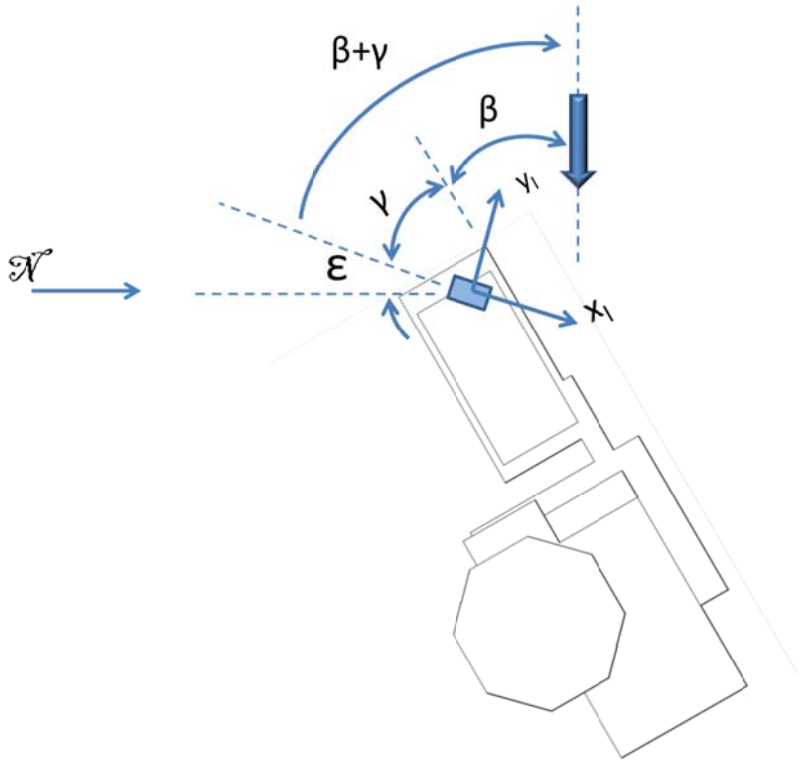


Figure 9: Software velocity vector rotation to north Coordinate system.

Applying the rotations described above to the CFD vector data produces the u, v and w components in the LiDAR coordinate system. From equations 4 to7 the line of sight velocities s_0 to s_3 may then be determined and the velocity vector that would be returned from the LiDAR calculated.

3.4 Correction factors

If it is taken that the CFD data were simulated in a free stream of known magnitude and direction aligned then any change in the magnitude of any component in the u, v and w directions, and therefore a change in the azimuth angle θ of the flow, must be due to interference caused by the presence of the platform. Using the CFD data it is possible estimate the magnitude of the interference and therefore derive a correction factor to

compensate for the effect of the interference. The process by which these correction factors were determined is described below.

To correct the u component of the LiDAR data to point data rearranging equation 14 gives the correction factor in equation 15. The same process can be applied for correction factors for the v and w components

$$u_{point} = cfp_u * u_{lidar} \quad \text{equation 14}$$

$$cfp_u = \frac{u_{point}}{u_{lidar}} \quad \text{equation 15}$$

To correct the u and v LiDAR data to the undisturbed free stream values requires equations 16 and 17 which, when rearranged give the correction factors for these components in equations 18 and 19

$$u_{free\ stream} = cff_u * u_{lidar} \quad \text{equation 16}$$

$$v_{free\ stream} = cff_v * v_{lidar} \quad \text{equation 17}$$

$$cff_u = \frac{u_{free\ stream}}{u_{lidar}} \quad \text{equation 18}$$

$$cff_v = \frac{v_{free\ stream}}{v_{lidar}} \quad \text{equation 19}$$

Because the w component of the free stream will always be zero in the simulation a slightly different correction factor is derived. The correction factor to calculate the w component of the free stream velocity from the LiDAR data is given by equation 20

$$w_{free\ stream} = w_{lidar} + cff_w\, U_{free\ stream} \quad \text{equation 20}$$

In the simulations the w component is zero hence the correction factor is given by equation 21

$$cff_w = -\frac{w_{lidar}}{U_{free\ stream}} \quad \text{equation 21}$$

The correction addends for the azimuth angle are calculated by equations 22 and 23

$$\theta_{point} = \theta_{lidar} + cfp_\theta \quad \text{equation 22}$$

$$\theta_{free\ stream} = \theta_{lidar} + cff_\theta \quad \text{equation 23}$$

4 UDF verification

In order to verify the UDF two calculations were carried out for a data point above the Horns Rev2 platform. The first calculation was carried out by hand and may be found in appendix A. The second calculation was carried out by the UDF. The results of the two calculations may be seen in table 1.

	UDF	Hand Calc
	m/s	m/s
S_0	-6.75	-6.75
S_1	1.94	1.94
S_2	8.31	8.40
S_3	-0.31	-0.32
U	2.25	2.26
V	-15.06	-15.15
W	0.90	0.93
θ	81.484	

Table 1: Comparison between hand calculation and UDF

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

5 Program Listing

The following pages contain a listing of the code for the LiDAR simulation UDF.

```
#include "point_in_cell.h"
#include "math.h"
#include "string.h"
#include "stdlib.h"

/* note that phi is the cone angle */
/* note psi is the asimuth angle */

DEFINE_ON_DEMAND(windcube_comp_sim)
{

void z_rotation(float *ptr, float theta);

int yorn, zone;
float xloc, yloc, zloc, height, dh, h, nh, ver, gamma, beta, cfd_psi, epsilon;
float dumnum1, dumnum2, dumnum3, dumnum4;
float x, y, z, hmeas[50], s[4][3], U[3],U_magl, U_magp, Ul[3],Up[3];
float phi_deg=30, phi_rad, lid_rig_deg, rig_vel_deg, theta_rad, thp, thl, nth_sth_deg;
float cff_U, cff_u, cff_v, cff_w, cff_th, cfp_U, cfp_u, cfp_v, cfp_w, cfp_th;
float u_ref, v_ref, w_ref, U_mag_ref;
float npoints=50;
float pi=3.14159;
float psi_deg, psi_inc, psi_rad;
float ploc[50][3], Vlos[4], asangle[50];
int i,j,nj,b,result, np;

char dummy;
char *header={"h     U_mag_lidar U_lidar V_lidar W_lidar Th_lidar U_mag_point U_point V_point W_point th_point cff_u
cff_v cff_w cff_U cff_th cfp_u cfp_v cfp_w cfp_U cfp_th \n"};
char string1[40];
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
/*define the free stream reference condition from the CFD boundary condition */

float U_ref[3]={-15.0,0,0};

FILE *exp_file_ptr;
FILE *inp_file_ptr;

Domain *d;
Thread *ct;
cell_t c;
real point[3];

ver=1.06;

Message("WINDCUBE full simulation program version %.2f started \n\n", ver);

U_mag_ref=sqrt(U_ref[0]*U_ref[0]+U_ref[1]*U_ref[1]+U_ref[2]*U_ref[2]);

/* open a file to read data */

Message("opening settings.dat \n");

inp_file_ptr=fopen("settings.dat","r");

Message("reading settings.dat \n");

/* read the data in from the settings file */

if (inp_file_ptr !=NULL)

            {
            fscanf(inp_file_ptr, "%s %s %s %s %s %s %s %s \n", &dummy, &dummy, &dummy, &dummy, &dummy, &dummy,
&dummy, &dummy);
            Message("read header \n");
            fscanf(inp_file_ptr, "%d %d %f %f %f %f %f %f \n", &zone, &yorn, &height, &nh, &gamma, &beta, &cfd_psi,
&epsilon);
            Message("read data \n");
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
            }

      else

            {
            Message("no file name \n");
            }

np=beta/30.0;

for (i=0;i<np; i++)

            {
                  fscanf(inp_file_ptr, "%f %f %f %f \n", &dumnum1, &dumnum2, &dumnum3, &dumnum4);

            }

fscanf(inp_file_ptr, "%f %f %f %f ", &beta, &xloc, &yloc, &zloc);

fclose(inp_file_ptr);

Message("finished reading settings.dat \n\n");

Message("x location %6.4f \n", xloc);
Message("y location %6.4f \n", yloc);
Message("z location %6.4f \n", zloc);
Message("gamma %6.4f\n", gamma);
Message("beta %6.0f\n", beta);
Message("psi %6.0f\n", cfd_psi);
Message("epsilon %6.0f\n", epsilon);

/* open a file to write data */

b=beta;

result=sprintf(string1, "windcube simulation %d degrees.dat",b);

Message("opening %s \n", string1);
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
exp_file_ptr=fopen(string1,"w");


/* inputs required for the fluent UDF get domain defines the entire domain and lookup_thread get the thread
number for the specified zone in the given domain */

d = Get_Domain(1);
ct = Lookup_Thread(d, zone);

/* define the angle of the free stream realtive to the lidar */

rig_vel_deg=-1.0*(beta+gamma);

/* change software rotation because rotation os wind vector is clockwise hence negative */

nth_sth_deg=-1.0*epsilon;

/* define the angle of the lidar relative to the cfd axis system */

lid_rig_deg=beta+gamma;

/* calculate the distance between measurement heights */

Message("\n %.2f heights to be calculated \n", nh);

dh=height/nh;

/* convert the lidar cone angle to degrees */

phi_rad=phi_deg*pi/180;

/* there are two types of analysis: The verbose version writes info to the screen during calculation
and the output file columns are annotated. The other just shows the number of steps on screen and just
outputs the data to file without headers  */

if (yorn == 1)
{
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
      Message("verbose output version %.2f \n", ver);
/* write a header to the file */
fputs(header,exp_file_ptr) ;
/* rotate the velocity from the CFD coordinate system into the Lidar Cordinate system */
z_rotation(U_ref, cfd_psi);
/* rotate the velocity from the lidar coordinate system into the rig Cordinate system */
z_rotation(U_ref, rig_vel_deg);
/* rotate the velocity into the north-south Cordinate system */
z_rotation(U_ref, nth_sth_deg);
/* for loop to calculate the velocity vector at each height where nh is the number of heights */
for (j=1; j<=nh; j++)
{
      Message("measurement height %d being calculated \n", j);
      /* calculate the height */
      h=dh*j;
      /* assign to the point array the measurement point location above the lidar in CFD coordinate */
      point[0]= xloc;
      point[1]= yloc;
      point[2]= zloc + h;
      /* find the cell in the computational domain containing the point */
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
c = cell_containing_point(point, ct);

/* find the u, v and w components of the velocity vector at this point */

Up[0]=C_U(c,ct);
Up[1]=C_V(c,ct);
Up[2]=C_W(c,ct);

/* rotate the velocity from the CFD coordinate system into the Lidar Cordinate system */

z_rotation(Up, cfd_psi);

/* rotate the velocity from the lidar coordinate system into the rig Cordinate system */

z_rotation(Up, rig_vel_deg);

/* rotate the velocity to the north south coordinate system*/

z_rotation(Up, nth_sth_deg);

Message("point velocity in rig coords %4.2f %4.2f %4.2f \n", Up[0],Up[1],Up[2]);

/* calculate the magnitude of the horizontal component */

U_magp=sqrt(Up[0]*Up[0]+Up[1]*Up[1]);

Message("Vel magnitude in horiz plane %4.2f \n", U_magp);

/* calculate the flow angle */

if (Up[0]>0)
{
      if(Up[1]>0)
      {
            thp=360.0-atan(Up[1]/Up[0])*180/pi;
      }
      if(Up[1]<0)
      {
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
                thp=-atan(Up[1]/Up[0])*180/pi;
        }
}
else
{
        thp=180.0-atan(Up[1]/Up[0])*180/pi;

}

Message("flow azimuth angle %4.2f \n", thp);

/* for loops to simulate the windcube lidar */

/* first calculates the velocity vector at the four measurement points */

psi_inc=pi/2.0;

for (i=0;i<=3; i++)

        {

        /* calculate the measurement points of the wind cube at the measurement height in CFD coordinate system
        note that the measurement points are S0 on the +ve y axis incrementing positive clockwise */

        psi_rad=i*psi_inc;

        /* first at the origin */

        point[0]= h*sin(phi_rad)*sin(psi_rad)/cos(phi_rad) ;
        point[1]= h*sin(phi_rad)*cos(psi_rad)/cos(phi_rad) ;
        point[2]= h;

        /* rotate 180 degrees from CFD into Lidar coordinate system */

        z_rotation(point, cfd_psi);

        /* rotate the measurement location into the rig coordinate system */
```

```
z_rotation(point, lid_rig_deg);

/* add the offset to translate the points to be centred on the lidar location */

point[0]= point[0]+xloc;
point[1]= point[1]+yloc;
point[2]= zloc + h;

Message("Windcube measurement point S%1d %6.3f %6.3f %6.3f \n",i,point[0],point[1],point[2]);

/* find the velocity vector at the point */

c = cell_containing_point(point, ct);

/* find the u, v and w components of the velocity vector at this point */

U[0]=C_U(c,ct);
U[1]=C_V(c,ct);
U[2]=C_W(c,ct);

/* rotate the velocity vector from CFD to Lidar coordinate system */

z_rotation(U, cfd_psi);

/* rotate the velocity vector from the Lidar coordinate system into the rig coordinate system */

z_rotation(U, rig_vel_deg);

Message("velocity at point S%1d %6.3f %6.3f %6.3f \n",i, U[0], U[1], U[2]);

/* put the velocity vector into the s array */

s[i][0]=U[0];
s[i][1]=U[1];
s[i][2]=U[2];


}
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
/* next derives the Vlos at the four s locations */

for (i=0;i<=4; i++)

      {

      /* calculate the angle of the measurement point from the y axis, positive clockwise */

      theta_rad=i*pi/2.0;

      /* calculate the Vlos at the point */

      Vlos[i]=s[i][0]*sin(phi_rad)*sin(theta_rad)+s[i][1]*sin(phi_rad)*cos(theta_rad)+s[i][2]*cos(phi_rad);

      }

Message("Vlos at the four points are; %6.3f %6.3f %6.3f %6.3f \n", Vlos[0],Vlos[1],Vlos[2],Vlos[3]);

/* calcualte the velocity vector based on the four Vlos values */

Ul[0]=(Vlos[1]-Vlos[3])/(2*sin(phi_rad));
Ul[1]=(Vlos[0]-Vlos[2])/(2*sin(phi_rad));
Ul[2]=(Vlos[0]+Vlos[2])/(2*cos(phi_rad));

/* apply the software rotation to the velocity vector to align with the North-South coordinate system */

z_rotation(Ul, nth_sth_deg);

/* calculate the magnitude in the horizontal plane */

U_magl=sqrt(Ul[0]*Ul[0]+Ul[1]*Ul[1]);

/* calculate the azimuth flow angle */

if (Ul[0]>0)
{
      if(Ul[1]>0)
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
            {
                  thl=360.0-atan(Ul[1]/Ul[0])*180/pi;
            }
            if(Ul[1]<0)
            {
                  thl=-atan(Ul[1]/Ul[0])*180/pi;
            }
      }
      else
      {
            thl=180.0-atan(Ul[1]/Ul[0])*180/pi;
      }


      /* write the lidar velocity to screen */

      Message( "Lidar velocity, magnitude and azimuth angle; %6.3f %6.3f %6.3f %6.3f %6.3f \n", Ul[0],Ul[1], Ul[2],
U_magl, thl);

      /* calculate the correction factors */

      /* correction to free stream */

      cff_u=U_ref[0]/Ul[0];
      cff_v=U_ref[1]/Ul[1];
      cff_w=U_ref[2]/U_mag_ref-Ul[2]/U_mag_ref;
      cff_U=U_mag_ref/U_magl;
      cff_th=(beta+gamma+epsilon)-thl;

      /* correction to point */

      cfp_u=Up[0]/Ul[0];
      cfp_v=Up[1]/Ul[1];
      cfp_w=Up[2]/Ul[2];
      cfp_U=U_magp/U_magl;
      cfp_th=thp-thl;

      /* write the lidar and point velocity to file */
```

```
      fprintf(exp_file_ptr, " %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f \n", h, U_magl, Ul[0], Ul[1], Ul[2], thl, U_magp, Up[0], Up[1], Up[2], thp, cff_u, cff_v, cff_w, cff_U, cff_th, cfp_u, cfp_v, cfp_w, cfp_U, cfp_th);


      }


}
else
{
Message("abreviated output version %.2f \n", ver);

/* fprintf(exp_file_ptr, "%.2f \n",nh); */

/* Calculate the reference velocity vector */

/* rotate the velocity from the CFD coordinate system into the Lidar Cordinate system */

z_rotation(U_ref, cfd_psi);

/* rotate the velocity from the lidar coordinate system into the rig Cordinate system */

z_rotation(U_ref, rig_vel_deg);

/* rotate the velocity to the north south coordinate system*/

z_rotation(U_ref, nth_sth_deg);

/* for loop to calculate the velocity vector at each height where nh is the number of heights */

for (j=1; j<=nh; j++)

{
      Message("measurement height %d being calculated \n", j);

      /* calculate the height */
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
h=dh*j;

/* assign to the point array the measurement point location above the lidar in CFD coordinate */

point[0]= xloc;
point[1]= yloc;
point[2]= zloc + h;

/* find the cell in the computational domain containing the point */

c = cell_containing_point(point, ct);

/* find the u, v and w components of the velocity vector at this point */

Up[0]=C_U(c,ct);
Up[1]=C_V(c,ct);
Up[2]=C_W(c,ct);

/* rotate the velocity from the CFD coordinate system into the Lidar Cordinate system */

z_rotation(Up, cfd_psi);

/* rotate the velocity from the lidar coordinate system into the rig Cordinate system */

z_rotation(Up, rig_vel_deg);

/* rotate the velocity to the north south coordinate system*/

z_rotation(Up, nth_sth_deg);

U_magp=sqrt(Up[0]*Up[0]+Up[1]*Up[1]);

/* calculate the flow angle */

if (Up[0]>0)
{
      if(Up[1]>0)
```

```
            {
                  thp=360.0-atan(Up[1]/Up[0])*180/pi;
            }
            if(Up[1]<0)
            {
                  thp=-atan(Up[1]/Up[0])*180/pi;
            }
      }
      else
      {
            thp=180.0-atan(Up[1]/Up[0])*180/pi;

      }

      /* for loops to simulate the windcube lidar */

      /* first calculates the velocity vector at the four measurement points */

      psi_inc=pi/2.0;

      for (i=0;i<=3; i++)

            {

            /* calculate the measurement points of the wind cube at the measurement height in CFD coordinate system
*/
            /* note that the measurement points are S0 on the +ve y axis incrementing positive clockwise */

            psi_rad=i*psi_inc;

            /* first at the origin */

            point[0]= h*sin(phi_rad)*sin(psi_rad)/cos(phi_rad) ;
            point[1]= h*sin(phi_rad)*cos(psi_rad)/cos(phi_rad) ;
            point[2]= h;

            /* rotate 180 degrees from CFD into Lidar coordinate system */
```

```
z_rotation(point, cfd_psi);

/* rotate the measurement location into the rig coordinate system */

z_rotation(point, lid_rig_deg);

/* add the offset to translate the points to be centred on the lidar location */

point[0]= point[0]+xloc;
point[1]= point[1]+yloc;
point[2]= zloc + h;

/* find the velocity vector at the point */

c = cell_containing_point(point, ct);

/* find the u, v and w components of the velocity vector at this point */

U[0]=C_U(c,ct);
U[1]=C_V(c,ct);
U[2]=C_W(c,ct);

/* rotate the velocity vector from CFD to Lidar coordinate system */

z_rotation(U, cfd_psi);

/* rotate the velocity vector from the Lidar coordinate system into the rig coordinate system */

z_rotation(U, rig_vel_deg);

/* put the velocity vector into the s array */

s[i][0]=U[0];
s[i][1]=U[1];
s[i][2]=U[2];

}
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
/* next derives the Vlos at the four s locations */

for (i=0;i<=4; i++)

      {

      theta_rad=i*pi/2.0;

      Vlos[i]=s[i][0]*sin(phi_rad)*sin(theta_rad)+s[i][1]*sin(phi_rad)*cos(theta_rad)+s[i][2]*cos(phi_rad);

      }

Ul[0]=(Vlos[1]-Vlos[3])/(2*sin(phi_rad));
Ul[1]=(Vlos[0]-Vlos[2])/(2*sin(phi_rad));
Ul[2]=(Vlos[0]+Vlos[2])/(2*cos(phi_rad));

/* apply the software rotation to the velocity vector to align with the North-South coordinate system */

z_rotation(Ul, nth_sth_deg);

U_magl=sqrt(Ul[0]*Ul[0]+Ul[1]*Ul[1]);

/* calculate the flow angle */

if (Ul[0]>0)
{
      if(Ul[1]>0)
      {
            thl=360.0-atan(Ul[1]/Ul[0])*180/pi;
      }
      if(Ul[1]<0)
      {
            thl=-atan(Ul[1]/Ul[0])*180/pi;
      }
}
else
{
      thl=180.0-atan(Ul[1]/Ul[0])*180/pi;
```

```
	}


	/* calculate the correction factors */

	cff_u=U_ref[0]/Ul[0];
	cff_v=U_ref[1]/Ul[1];
	cff_w=U_ref[2]/U_mag_ref-Ul[2]/U_mag_ref;
	cff_U=U_mag_ref/U_magl;
	cff_th=(beta+gamma+epsilon)-thl;

	cfp_u=Up[0]/Ul[0];
	cfp_v=Up[1]/Ul[1];
	cfp_w=Up[2]/Ul[2];
	cfp_U=U_magp/U_magl;
	cfp_th=thp-thl;


	/* write the lidar and point velocity to file */

	fprintf(exp_file_ptr, " %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f
%6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f \n", h, U_magl, Ul[0], Ul[1], Ul[2], thl, U_magp, Up[0], Up[1], Up[2],
thp, cff_u, cff_v, cff_w, cff_U, cff_th, cfp_u, cfp_v, cfp_w, cfp_U, cfp_th);

}
}

fclose(exp_file_ptr);

	Message("lidar simulation ended data written to file %s \n", string1);

}
```

5.1 Functions

The subroutines (functions) called by the UDF are listed below

```
cxboolean outward_face(face_t f, Thread *ft, cell_t c, Thread *ct)
{
  Thread *ct1;
  cell_t c1;
  ct1 = THREAD_T1(ft);
  if(NULLP(ct1))return TRUE; /* face on boundary */
  c1 = F_C1(f,ft);
  if ((c==c1)&&(ct==ct1))return FALSE; /* face's c1 is c so f inward to c */
  return TRUE;
}
cxboolean point_in_cell(real point[ND_ND], cell_t c, Thread *ct)
{
  int i;
  real dist;
  real p_rel[ND_ND], f_cen[ND_ND], A[ND_ND];
  face_t f;
  Thread *ft;
  cxboolean inside = TRUE;
  c_face_loop(c, ct, i)
      {
        if(inside)
            {
              f=C_FACE(c, ct, i);
              ft=C_FACE_THREAD(c, ct, i);
              F_CENTROID(f_cen,f,ft);
              F_AREA(A,f,ft);
              NV_VV(p_rel,=,point,-,f_cen); /* point relative to f_cen */
              dist=NV_DOT(p_rel,A);
              if (outward_face(f,ft,c,ct)) /* Count as inside if dist == 0.0 */
                  {if (dist > 0.0) inside = FALSE;}
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
              else
                  {if (dist < 0.0) inside = FALSE;}
            }
      }
  return inside;
}
cell_t cell_containing_point(real point[ND_ND], Thread *ct)
{
  cell_t c;
  cell_t c_in = NULL_CELL;
  begin_c_loop(c,ct)
      {
        if(c_in == NULL_CELL) /* if cell not found yet */
            {
              if (point_in_cell(point, c, ct)) c_in = c;
            }
      }
  end_c_loop(c,ct)
  return c_in;
}
void z_rotation(float *ptr, float theta)

{
      float x, y, z;
      float xr, yr, zr;
      float pi=3.14159;

      theta=theta*pi/180.0;

      x=*ptr;
      y=*(ptr+1);
      z=*(ptr+2);

      xr=x*cos(theta)-y*sin(theta);
      yr=x*sin(theta)+y*cos(theta);
      zr=z;

/*    printf("matrix after multiplication is %4.2f %4.2f %4.2f \n",xr, yr, zr);*/
```

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00

```
	*ptr=xr;
	*(ptr+1)=yr;
	*(ptr+2)=zr;

	return;

}
```

6 Settings file format

The simulation program reads a settings.dat file which contains information used by the UDF, table 2.

zone	verbose	height	nh	gamma	beta	psi	epsilon
2	0	1.0	50	0.0	330.0	180.0	21.0
0	3.9316	3.4358	0.355				
30	3.9235	3.4343	0.355				
60	3.929	3.9255	0.355				
90	3.7933	3.9954	0.355				
120	3.5076	3.9197	0.355				
150	3.5076	3.9197	0.355				
180	3.15	3.705	0.355				
210	3.2308	3.4255	0.355				
240	3.2303	3.2861	0.355				
270	3.3691	3.1432	0.355				
300	3.6541	3.1466	0.355				
330	3.9358	3.1483	0.355				

Table 2: Settings.dat file data

The first line of text describes the value below

Zone (2) the definition of the fluid zone from fluent. If unknown then needs to be determined by selecting the fluid region from the Surface/zone surface menu and then selecting manage. The number shown in the ID box is the zone name.

Verbose (0) if set to 0 the abbreviated version of the UDF is run, if set to 1 the verbose version is executed.

Height (1.0) the height above the LiDAR location over which the interrogations will be made.

Nh (50) the number of discrete steps over the range set by “height” that will be interrogated.

Gamma (0.0) the angle in degrees between the LiDAR and rig coordinate systems

Beta (330.0) the angle between the free stream wind vector and the rig coordinate system

Psi (180) the rotation between the CFD and the LiDAR coordinate system.

Epsilon (21.0) software rotation to rotate measured wind vector into a coordinate system aligned with north-south

The groups of four digits below give the location of the LiDAR on the platform in the CFD coordinate system for each of the wind directions in the order; beta, x, y, z. linear dimensions are in metres and rotation angles are in degrees

7 Output file format

The LiDAR simulation program writes the calculated data into an output file the format of which depends on whether the verbose or abbreviated version of the program was used.

The numerical data from the LiDAR simulation is the same for both file formats the only difference being that the verbose version has a header line written to the file which describes the contents of the column below, table 3.

h	U_mag_lidar	u_lidar	v_lidar	w_lidar	th_lidar	U_mag_point	u_point	v_point	w_point	th_point	cff_u	cff_v	cff_w	cff_U	cff_th	cfp_u	cfp_v	cfp_w	cfp_U	cfp_th

table 3;output data file format

The name of the output file is defined as

windcube simulation *beta* degrees.dat

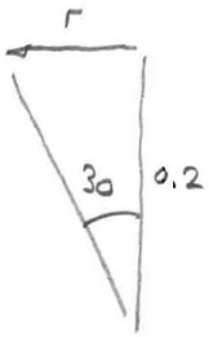
where *beta* is the angle defined in the settings file.

8

hand Calc $h = 0,2\,m$ $\gamma = 21°$ $\beta = 60°$

$\bar{U} = 15\,m/s$ $u = -15$ $v = 0$ $w = 0$

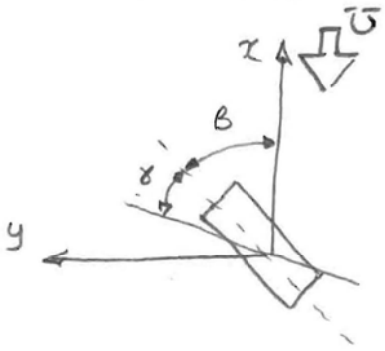
$h = 0,2\,m$



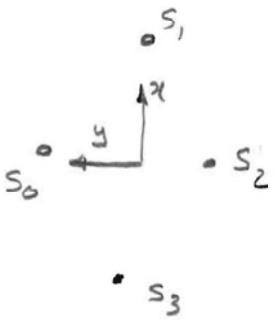
$r = 0,2 \tan 30$

$= 0,1155$

CFD coord



sodar



coords of S in CFD coords

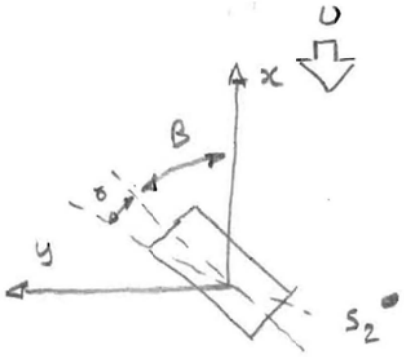
S_0 $(0, 0.1155)$

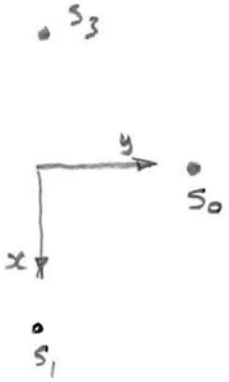
S_1 $(0.1155, 0)$

S_2 $(0, -0.1155)$

S_3 $(-0.1155, 0)$

rotate sodar to x positive in U direction





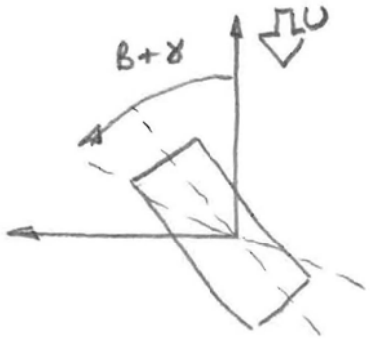
S_0 $(0, -0,1155)$

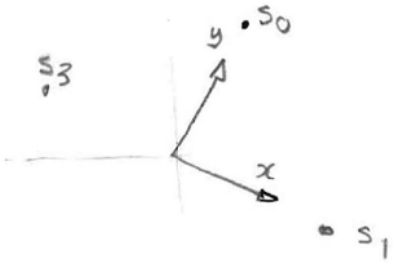
S_1 $(-0.1155, 0)$

S_2 $(0, 0.1155)$

S_3 $(0,1155, 0)$

rotate sodar to align with installation on rig





rotate (B+8) about z
and add offset (3,929, 3.9255, 0,355)

S_0	(0.114, -0.018)	S_0 (4,043, 3,908, 0.555
S_1	(-0.018, -0.114)	S_1 (3,911, 3.812, 0.555
S_2	(-0.114, 0.018)	S_2 (3,815, 3.944, 0.555)
S_3	(0.018, 0.114)	S_3 (3.947, 4.039, 0.555)

Find velocity vector at each location by interrogating fluent

	U	V	W
S_0	-14.81	0,36	0.68
S_1	-15.15	0.49	1.15
S_2	-15.83	0,44	0.64
S_3	-15.18	0,57	0.68

NB slight diff in U, V, W from code values as above are facet average whereas code uses vertex average.

rotate the velocity from CFD to lidar coordinate system $\theta = 180°$

	U	V	W
S_0	14.81	-0.36	0.68
S_1	15.15	-0.49	1.15
S_2	15.83	-0.44	0.64
S_3	15.18	-0.57	0.68

rotate the lidar coord data relative to the rig instalation $\theta = -81°$

	U	V	W
S_0	1.96	-14.68	0.68
S_1	1.88	-15.04	1.15
S_2	2.04	-15.70	0.64
S_3	1.81	-15.08	0.68

all rotations using

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

rotation define +ve anticlockwise about Z

to calculate V_{los}

$$V_{los} = u \sin\phi \sin\psi + v \sin\phi \cos\psi + w \cos\phi$$

S_0; $\phi = 30°$ $\psi = 0$ $S_0 = -6.75$

S_1; $\phi = 30°$ $\psi = 90°$ $S_1 = 1,94$

S_2; $\phi = 30°$ $\psi = 180°$ $S_2 = 8.40$

S_3; $\phi = 30°$ $\psi = 270°$ $S_3 = -0.32$

to calc u, v, w from V_{los}

$$u = \frac{S_1 - S_3}{2 \sin\phi} \qquad v = \frac{S_0 - S_2}{2 \sin\phi} \qquad w = \frac{S_0 + S_2}{2 \cos\phi}$$

$u = 2.26$ $v = -15.15$ $w = 0,93$

Calc of u, v and w from boundary conditions

$u = 15$ $v = 0$ $w = 0$

rotate $-81°$

$u = 2.34$ $v = -14.81$ $w = 0$

User Defined Function: WindCube_comp_sim

UoSNW018 revision 00