THESIS

CASA REAL-TIME MULTI-DOPPLER RETRIEVAL SYSTEM

Submitted by

Sean X. Zhang

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2011

Master's Committee:

Advisor: V. Chandrasekaran

V.N. Bringi
Anura P. Jayasumana
Paul W. Mielke, Jr.

ABSTRACT

CASA REAL-TIME MULTI-DOPPLER RETRIEVAL SYSTEM

Doppler synthesis of 3D wind products is of great practical importance to observing and understanding severe weather features such as tornadoes and microbursts. It becomes very effective for severe weather events if this modeling can be performed in real-time. A real-time multi-Doppler retrieval system forms an important product of modern weather radar networks. Challenging constraints exists between computing performance, high data resolution, and other quality issues. This Thesis describes the implementation of the operational Real-time Multi-Doppler Retrieval System (R-MDRS) of the Center for Collaborative Adaptive Sensing of the Atmosphere Engineering Research Center (CASA ERC). The R-MDRS is seamlessly integrated into CASA's Distributed Collaborative Adaptive Sensing (DCAS) operational framework and exhibit robust performance that strikes balance between high resolution and real-time processing speeds. A detailed technical description of the CASA R-MDRS implementation is given, including design approach that builds around two core components of the tool: interpolation to a common grid and Doppler synthesis. The R-MDRS generates 3D Wind products in step with network scanning modes and has been effective at detecting convective cells and tornadic activities. Data from 2009 and 2010 weather events are presented and analyzed for evaluating processing time as well as factors that effect data accuracy. These factors include Dual-Doppler candidate pair selection, advection correction, and variations in wind calculation techniques.

ACKNOWLEDGEMENTS

First I want to express my deep appreciation to Dr. Chandra for his guidance and advice, both in matters of research and everyday life. His subtle methods and unspoken words have been his most effective tools for helping me grow.

I also sincerely thank Drs. V.N. Bringi, Anura P. Jayasumana, and Paul W. Mielke for serving on my graduate committee.

I am grateful to my colleagues for their academic assistance and insights, as well as to Andy Crane and the rest of the Engineering Network Services for their invaluable technical support. I especially want to thank Minda Le for her friendship and for bringing me into a close network of friends. They have all been invaluable to me during these last few arduous years.

Finally and most profoundly, I want to thank my fiancee Zheng Wang for her tireless love and support. She has pushed me over my greatest obstacles and lifted me from my deepest troubles. She is the light to my dark, the strength to my weakness.

This research is supported by the National Science Foundation via the CASA ERC program (EEC-0313747).

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 CASA Background

The Center for Collaborative Adaptive Sensing of the Atmosphere, is a National Science Foundation Engineering Research Center (CASA ERC) dedicated to developing next generation weather-sensing radar networks. Its goals are to overcome current limitations with new technologies and to improve the paradigm of weather sensing.

Conventional weather radar networks such as NEXRAD utilize data from high-power, long-range radars usually operating in the S and C bands. The choice for low frequency was necessitated by a period when high frequency attenuation at long range was a severe and yet unresolved issue. However at long ranges, these radars are limited at observing lower parts of the atmosphere due to Earth's curvature, leading to under-sampling of meteorological conditions in the lower troposphere where most weather activities occur. Compounding the problem is the low resolution that these radars provide, with sample volumes extending to many cubic kilometers as range increases. These factors severely limit current weather sensing capabilities, especially in regards to observing small features such as tornadoes. Today, tornadoes often go undetected and the rate of false alarms is high.

CASA tries to overcome these limitations by employing a networked sensing approach using many small radars. These CASA radars operate at X-band over short ranges. Located just few tens of kilometers apart, they form a high frequency network that can see the lower troposphere in finer detail. The higher resolution of CASA radar observations consequently results in higher resolution Doppler retrievals and thus better detection of severe weather features. X-band has become a viable option for weather sensing due significant advances in

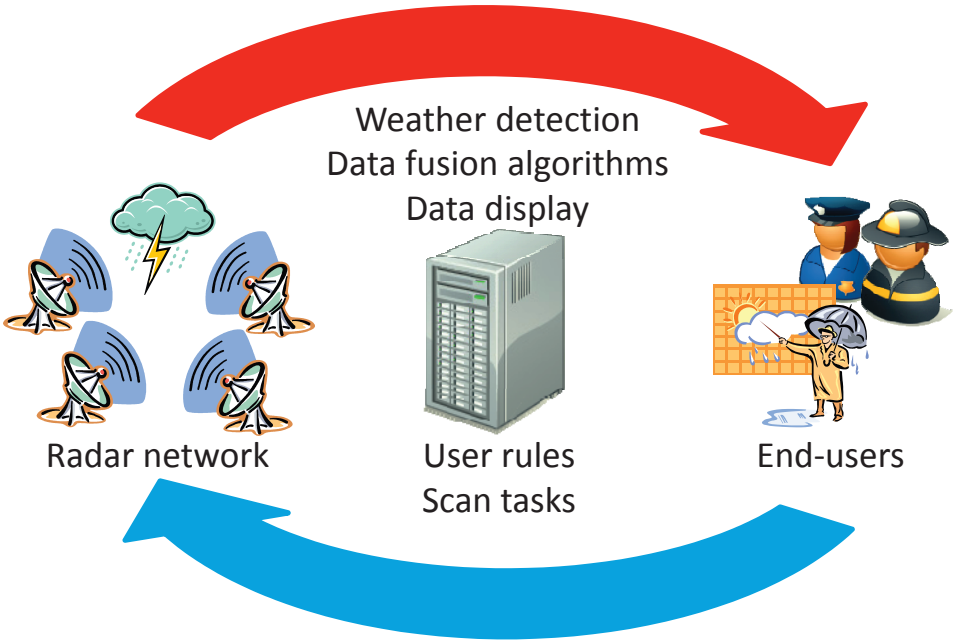


Figure 1.1: CASA DCAS framework

attenuation correction over long ranges. In addition, the smaller antenna required for these higher frequency radars significantly reduces the costs of manufacturing and deployment.

Besides networking small high frequency radars to overcome Earth curvature and resolution issues, CASA also employs an on-demand adaptive operation architecture to improve weather sensing and warning systems. This new operating paradigm is termed Distributed Collaborative Adaptive Sensing, or DCAS for short. 'Distributed' refers to the use of large number of small radars. 'Collaborative' and 'Adaptive' refer to the dynamic interactions between the radars, the weather, the radars' information technology infrastructure, and competing end-user needs. In DCAS, weather detection and end-user demands jointly dictate the radars to scan adaptively to areas of interest (Figure 1.1). This on-demand 'pull' method is a significant shift from NEXRAD's 'push' broadcast paradigm, and is perhaps a better approach for a weather sensing network composed of large numbers of small radars. In regards to measuring air motion, DCAS adaptively scans only where is needed, using optimal sets of Doppler radars. This makes it efficient at addressing the significant computational demands

of high resolution multi-Doppler retrievals and creates the feasibility for real-time air motion tracking using only modest computing resources.

1.2 Real-Time Multi-Doppler Retrieval

Air motion multi-Doppler retrievals has been a persistent pursuit since the birth of Doppler radars. While the mathematical principles behind multi-Doppler retrievals are relatively simple, implementing it in real-time has proven to be difficult due to high sensitivity to errors and computational limitations.

Most of the current day operational NEXRAD Systems are spaced with no significant overlap and as a result multiple Doppler product is not viable. Several research experiments in the past that produced high resolution Doppler wind products were by post-processing. CASA scan update times are on the order of 30 to 60 seconds to keep up with fast movement and formation of tornadoes. Thus reliable and timely warning of these hazards requires a multi-Doppler retrieval system that can operate in real-time while still maintaining high enough resolution to capture the features. Going real-time inevitably places limits on the throughput of the system based on computational power. This compromise of throughput for speed may be readily achieved by lowering the resolution. It is on the premise of this conflicting constraint that CASA's real-time multi-Doppler retrieval system is designed.

CASA's main test bed resides in Oklahoma's "Tornado Alley", code-named Integrative Project One (IP1). Here, fundamental research is being done on electromagnetic wave atmosphere interactions, new information infrastructure to support the DCAS paradigm, and lower atmosphere physics for sensing and forecasting. It is also here in IP1 that CASA's Real-time Multi-Doppler Retrieval System – henceforth known as R-MDRS – is tested and validated.

1.3 Outline of Thesis

The CASA R-MDRS is a convergence of many ideas including past implementations and new designs. This Thesis will detail this comprehensive system over four major sections, organized to provide an intuitive flow of the process behind its development.

Chapter 2 will be background information and a description of the CASA IP1 multi-Doppler problem. It will begin with a technical overview of radars in general, followed by Doppler principles. This will be followed by descriptions of the IP1 physical environment, CASA radar specifications, and operating procedures. The chapter will conclude with the mathematical formulations of multi-Doppler methodologies.

Chapter 3 provides a description of the CASA R-MDRS. A system overview will illustrate the interactions between the integrated parts, the overall execution process flow, and the real-time scheduling. Each subsystems of the R-MDRS will then be examined in detail. The principles and formulations behind each subsystem will be derived, followed by their practical functions and limitations. The chapter will conclude by examining the modularity, reliability, and other design features of the system.

Chapter 4 will showcase some experimental results and analyze the performance of the CASA R-MDRS. Weather events from 2009 and 2010 will be used to present the wind products. The accuracy and quality of the products will be validated with other radar networks, ground sightings, and post-storm damage analysis.

Chapter 5 will be the conclusion and comments on future works for the CASA R-MDRS. An evaluation will be made on the general performance of the current system, including its major limitations of computational throughput. A proposal to evolve the R-MDRS towards parallelism and modularization will be made.

Chapter 2

PROBLEM DESCRIPTION

Designing the R-MDRS for the CASA IP1 test bed requires considerations for the test bed layout, the radar network operation mode, and the methodologies behind Doppler techniques. Multi-Doppler methodology has matured over the decades and the primary challenge is implementing it into the CASA System constraints under which it must operate. To begin understanding these circumstances, the document starts with a brief technical overview of Doppler radars.

2.1 Radar Background

The word "radar" is an acronym for "**ra**dio **d**etection **a**nd **r**anging", which accurately describes its function. Radars operate by radiating electromagnetic beams toward targets to determine their properties based on the return signal. The most common application is to measure the size or intensity of the target based on the strength of the return signal, quantified as reflectivity.

Pulse radars operate by periodically sending out a short pulse and then 'listening' for the echo (Figure 2.1). This 'time-sharing' mode is what allows pulse radars to be compacted into a single antenna. Note that range-time τ is used to describe each period because each temporal instant in τ corresponds to a radial range $r = c\tau/2$. The division by 2 corresponds to the return trip. The time between each transmitted pulse T_s is known as the pulse repetition time or PRT. The reciprocal of the PRT $1/T_s$ is known as the pulse repetition frequency or PRF. The duration of the pulse T_0 is called pulse duration, but also commonly called the pulse width.

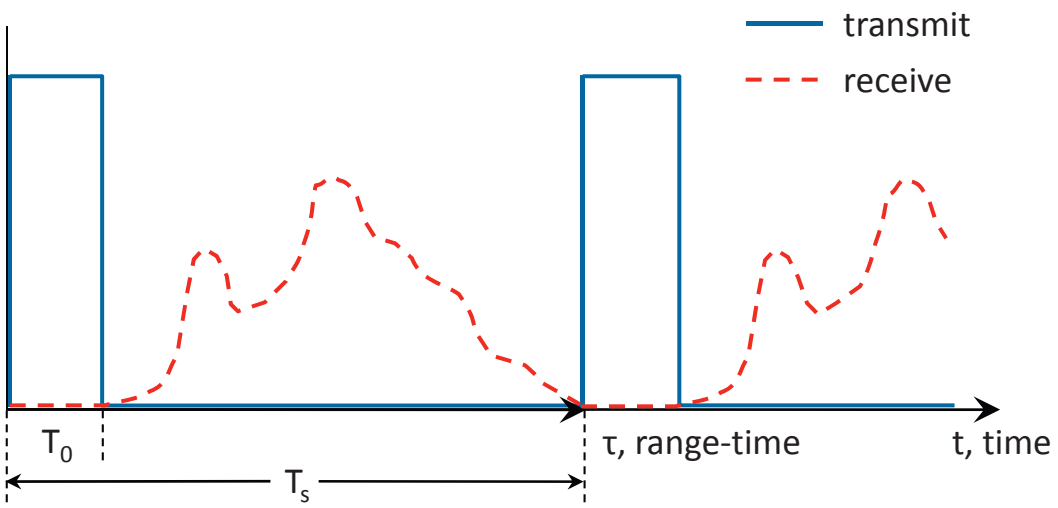


Figure 2.1: Transmitted pulse train and received echoes in range-time

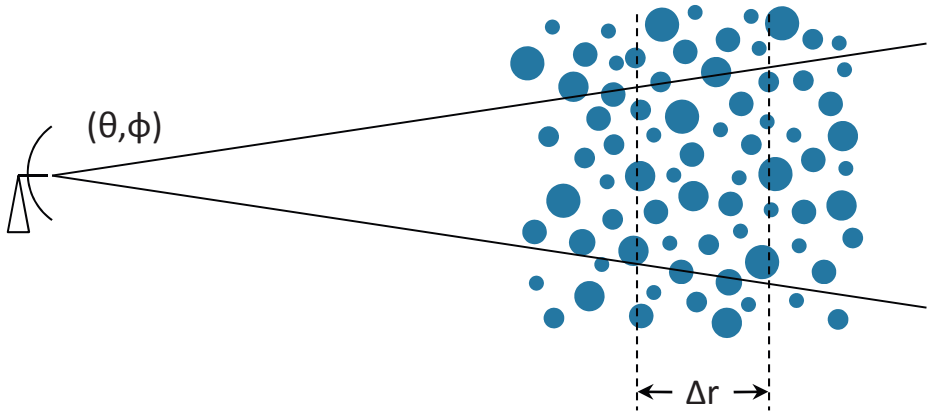


Figure 2.2: Sensing distributed targets within a sample volume [3]

Meteorological targets such as precipitation are composed of large numbers of hydrometeors extending over a large space. Pulse radars sense these as distributed targets within a sample volume. This sample volume is defined by the radar's horizontal and vertical beamwidths θ and ϕ extending a sample range Δr (Figure 2.2). The beamwidths are dimensions based on the physical parameters of the antenna. The sample range dimension Δr is dependent on the pulse width T_0 by

$$\Delta r = \frac{cT_0}{2} \tag{2.1}$$

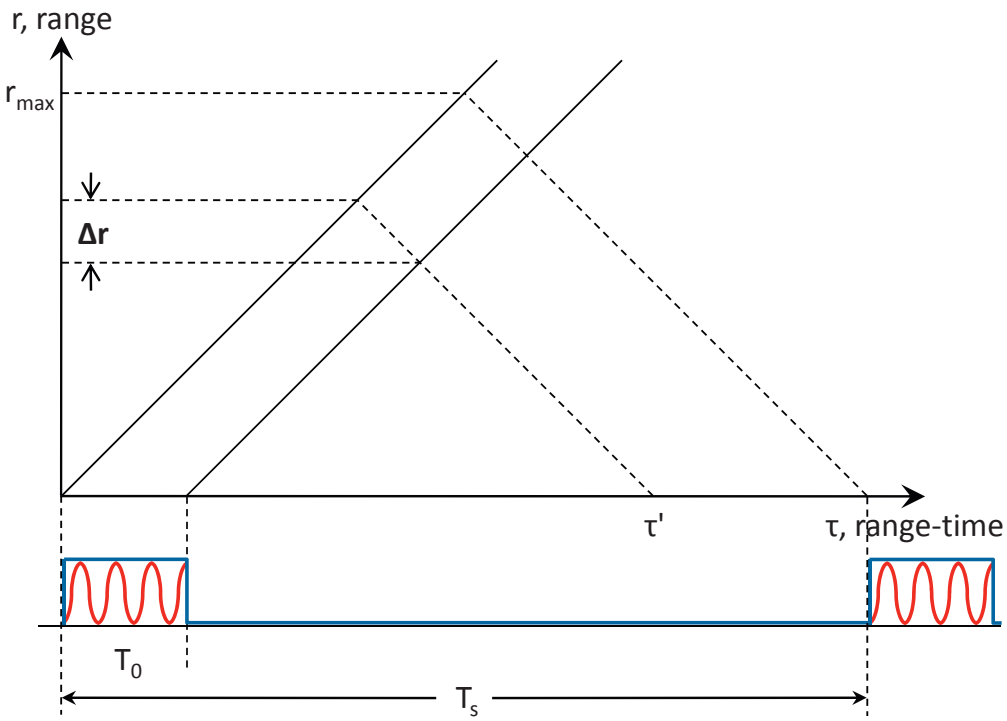


Figure 2.3: Radar return from particles in range resolution Δr [3]

This relationship is explained by the behavior of a finite-duration pulse in the range-time domain. For each instant in τ, when the leading edge of the pulse reaches a distance $r = c\tau/2$, it will have T_0 seconds to travel further out and back again before the trailing edge also reaches r. This way both echoes will return to the radar simultaneously at τ' (Figure 2.3). This additional distance that the leading edge may travel is precisely $\Delta r = cT_0/2$. Thus the received echo at any instant in τ corresponds to the sum of backscatter energies from all particles within the beam extending radially from r to $r + \Delta r$. Sampling in τ is the only way to preserve the spatial information of the sample volume during pulse radar operations. Any other way will irrevocably mix the echo signals of different locations in space, thereby losing the critical spatial information. Because Δr represents the finest detail the radar pulse could 'see', it is often referred to as the range resolution.

Pulse radars inherently time-sample the atmosphere with its PRF as the sampling frequency. Even for radars scanning quickly across a wide arc, the pulse rate is many orders of magnitude faster than the mechanical scanning motion. This results in each sample volume

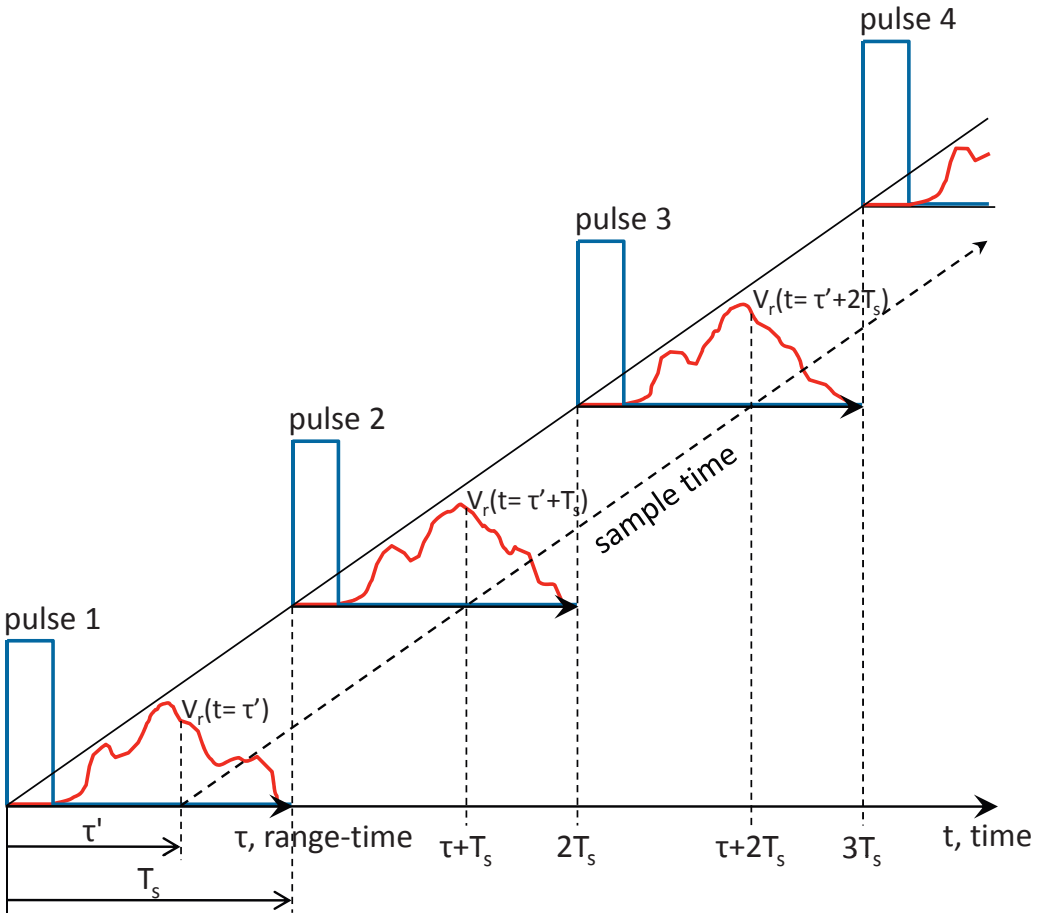


Figure 2.4: Range-time and discrete sample-time space of pulse radar operation [3]

time-sampled hundreds of times at $t = \tau'$, $t = \tau' + T_s$, $t = \tau' + 2T_s$, ... See Figure 2.4. In this discrete time-sample space, each sample is an observation of the underlying stochastic process taking place at the sample volume (Bringi & Chandra 2001). Together, time-samples of a sample volume create the most fundamental unit of radar data, known as a range gate or bin. The many gates of a particular pointing direction form a ray; the many rays of a scan form a sweep; and multiple sweeps across different elevations or azimuths form a volume. In a network of radars, the volume scans of each radar is synchronized to within a time frame known as the system heartbeat.

Another noteworthy characteristic of pulse radars is their maximum unambiguous range r_{max}, illustrated in Figure 2.3. While not directly related to Doppler retrievals, it does effect the Doppler performance of pulse Doppler radars, which will be examined in the next section.

r_{max} essentially indicates the maximum range a radar pulse can travel and return before the next pulse is sent out. It is defined as

$$r_{max} = \frac{cT_s}{2} \tag{2.2}$$

Of course targets beyond r_{max} can still be detected by the radar, but its reflected signal would be aliased with echoes from the next pulse, making its range ambiguous. This phenomenon is known as range aliasing or second-trip. The obvious solution to this problem is to increase r_{max} by lengthening the PRT, but this runs into conflicting constraints when examining the next focal point, Doppler radars.

2.2 Doppler Radars

The Doppler effect can be noted as

$$f_d = -\frac{v}{\lambda} \tag{2.3}$$

Here the subscript d denotes the Doppler frequency shift and λ denotes the wavelength. For a constant wavelength, the Doppler shift becomes solely dependent on the velocity. Backscatter from moving particles will be shifted in frequency based on particles' relative velocities with the radar. This affinity was recognized early on by radar engineers, and its exploitation augmented conventional reflectivity measurements of weather radars with complete kinematics of weather systems. Today, the vast majority of modern weather radars are pulse Doppler radars.

Measuring the frequency shifts in backscatter from moving particles yield the velocity of the particles as

$$v_r = -\frac{f_d \lambda}{2} \tag{2.4}$$

Note the subscript r indicating v_r to be the radial velocity. Radial velocity is the velocity component along the pointing direction of the radar. By convention, particles going away from the radar have positive radial velocity, hence the negative signs in Equations 2.3 and 2.4.

There are however limitations on the maximum velocity that can be resolved unambiguously. Pulse Doppler radars measure the Doppler shift by detecting the change in phase shift of the return signal across sample time. If a pulse is transmitted with an initial phase of ϕ_0, the phase of the return signal from the sample volume at range r will be

$$\phi = \phi_0 + \frac{4\pi r}{\lambda} \tag{2.5}$$

If there is radial movement in the sample volume, this phase will change with time from one pulse to the next. However, being a discrete time series, $d\phi/dt$ can be no greater than

$\pm\pi$ radians per sample. Thus the velocity that produces a phase shift of $\pm\pi$ radians is the maximum velocity that a Doppler radar can unambiguously detect. This is described as

$$v_{max} = \frac{f_{max}\lambda}{2} \tag{2.6}$$

where f_{max} is given by

$$f_{max} = \frac{(\mathrm{d}\phi/\mathrm{d}t)_{max}}{2\pi} = \frac{\pi/T_s}{2\pi} = \frac{\mathrm{PRF}}{2} \tag{2.7}$$

This yields the maximum unambiguous velocity as

$$v_{max} = \frac{\lambda \mathrm{PRF}}{4} \tag{2.8}$$

This result implies that to increase v_{max}, either the wavelength or the PRF must be increased, or both. Most often the PRF is more adjustable as wavelengths are locked into a range defined by the band of the radar.

Care must be taken when trying to achieve higher v_{max}. Recall the maximum unambiguous range r_{max} of pulse radars mentioned in Equation 2.2, restated as

$$r_{max} = \frac{cT_s}{2} = \frac{c}{2\mathrm{PRF}} \tag{2.9}$$

With PRF being in the numerator and denominator of v_{max} and r_{max} respectively, a conflicting constraint forms between range and velocity ambiguity. This is known as the "Doppler dilemma" and requires careful consideration in choosing the PRF during the design and usage of pulse Doppler radars. Compounding the two constraints for highlight, we have

$$v_{max} r_{max} = \frac{c\lambda}{8} \tag{2.10}$$

A larger v_{max} must require a smaller r_{max}, and vice versa. The right side of the equation is essentially constant for a given radar.

A pulse Doppler radar by itself is a powerful tool for weather sensing, but it cannot extrapolate the entire kinematics of weather systems because it can only measure radial velocities. Its true strength lies in working together as Doppler networks, where triangulations of radial velocities can determine true air motion vectors. The CASA IP1 test bed is one such Doppler network.

2.3 IP1 Overview

The CASA IP1 test bed is located approximately thirty miles southwest of Oklahoma City, OK and currently consists of four X-band pulse Doppler radars. The coverage area is an 140 km by 140 km area as illustrated by Figure 2.5. The radars are named based on nearby town names. They are clockwise from top-right, Chickasha (KSAO), Rushsprings (KRSP), Lawton (KLWE), and Cyril (KCYR). The location of the test bed was chosen for its climatology. Being in Tornado Alley, the test bed has a 77% chance to have at least one tornado touchdown in any given year. Severe storms are almost 100% guaranteed every year, with an average of 12 hail days annually [4].

IP1 radars are X-band dual-polarized pulse Doppler radars developed proprietary to CASA's research goals. This means that they are designed with the technical feasibility of future mass deployment in mind. The radars have a range resolution of 75 meters and scan to a maximum unambiguous range of 40 km during normal operations.

The layout of the radars is designed to optimize Doppler operations by maximizing coverage overlap and compensating for minimum beam-crossing angle blind-spots. Beam-crossing angle refers to the angle of intersection between beams of two radars. A small beam-crossing angle corresponds to two radars scanning a region between them that is close along the axis connecting the two radars. In such a case, the two radars would measure Doppler velocities that are approximately equal and opposite, giving very little or no orthogonal component to triangulate the true velocity vector. This creates a blind-spot, which would need to be compensated by another Doppler pair that has highly orthogonal beam-crossing angles at the region. Each of IP1's six radar pairs has their beam-crossing blind-spots covered by at least one other Doppler pair.

From the perspective of beam-crossing angles, each point within IP1's overlap regions has an optimal radar pair for Doppler calculations based on maximizing the orthogonality of the beam-crossing angle. These optimal Doppler pair regions are illustrated in Figure 2.6. The CASA multi-Doppler system, as its name suggests, goes beyond optimizing the choice

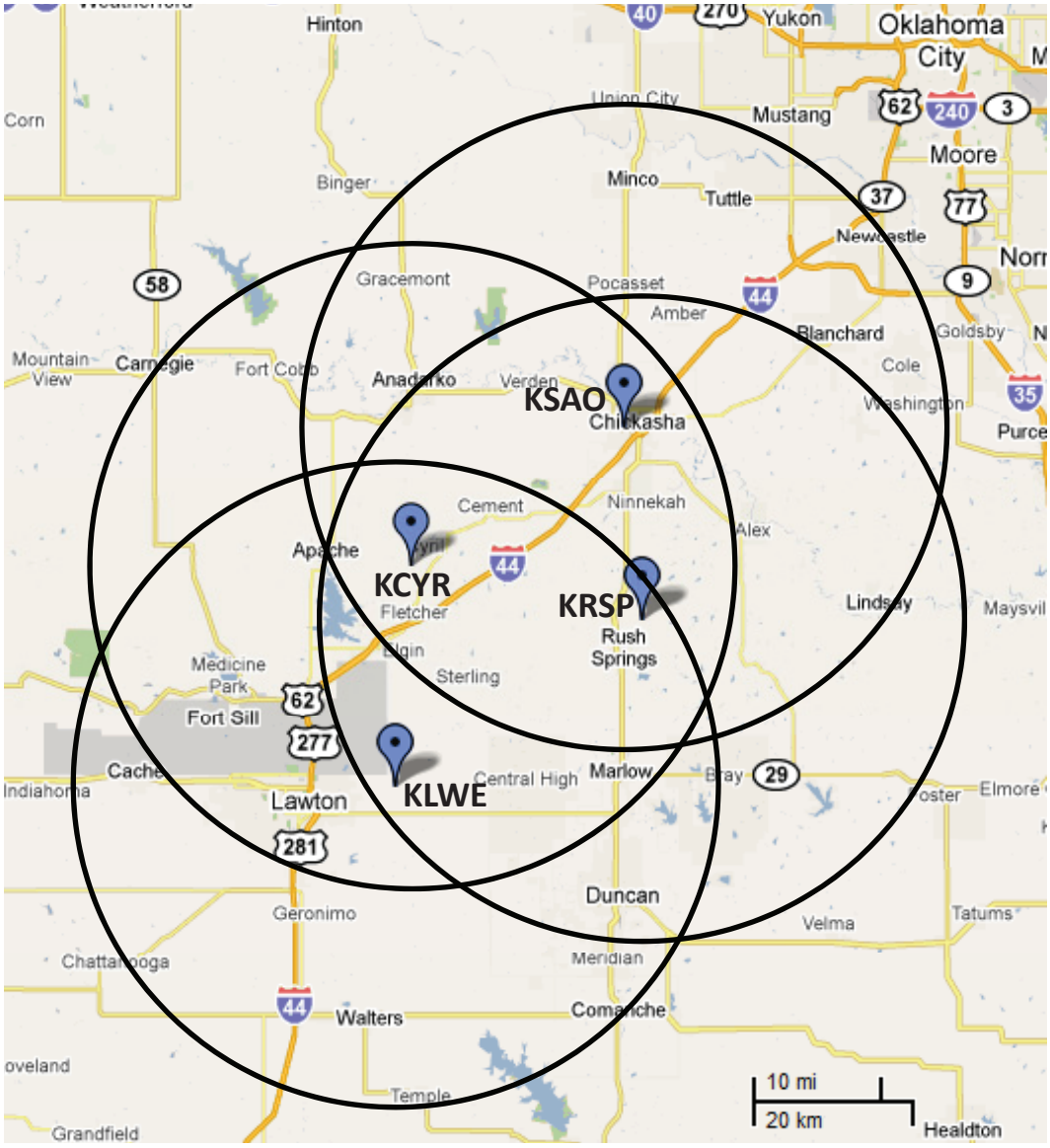


Figure 2.5: CASA IP1 geographic layout (Google Maps 2010)

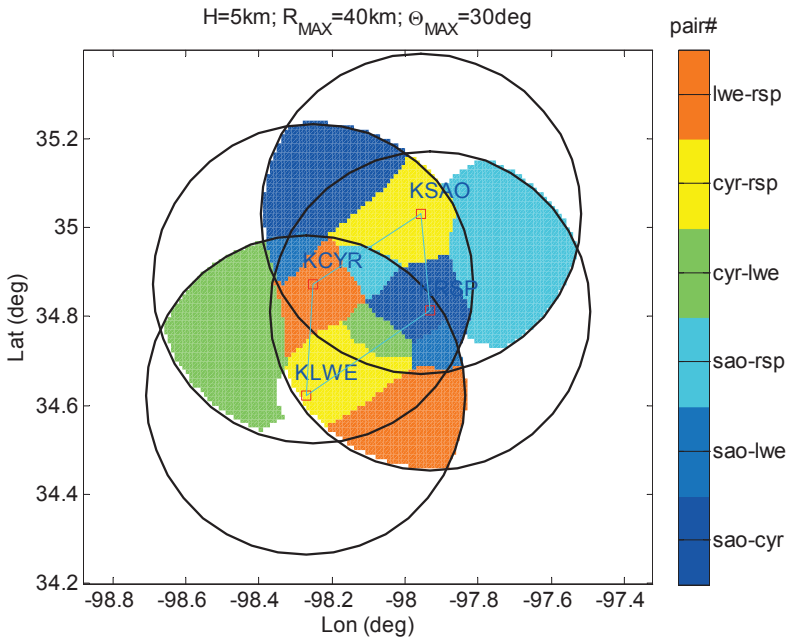


Figure 2.6: CASA IP1 optimal Doppler pair regions [12]

of dual-Doppler pair and uses all radars available in a region. However, dual-Doppler, tri-Doppler, and quad-Doppler regions are all handled separately by the retrieval software. This multi-Doppler approach may create over-determined systems, whose pros and cons will be analyzed later in Chapter 4.

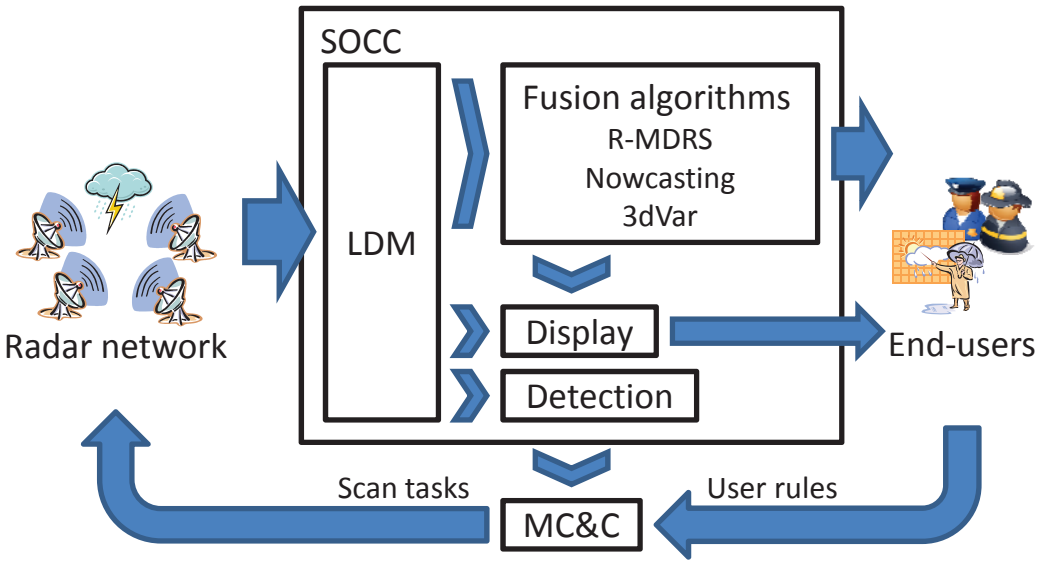


Figure 2.7: CASA IP1 operation loop and R-MDRS

2.4 IP1 Operation

IP1's operation conforms to the DCAS concept as described in Chapter 1. Figure 2.7 illustrates this in more detail. The four IP1 radars scan collaboratively and adaptively under the direction of the Meteorological Command and Control system (MC&C). The MC&C resides in the System Operation & Command Center (SOCC), a compute server which also houses the rest of IP1's processing subsystems. MC&C tries to maintain temporal and spatial synchronization between the radars so that they are scanning roughly the same region at about the same time. With a fast scanning scheme and the relative slow motion of weather systems, this creates an approximation to the ideal "snapshot" of the weather structure. To achieve this synchronized and fast scanning scheme, MC&C dictates a volume scan heartbeat of 1 to 3 minutes depending on scanning modes, with radars initiating new volume scans usually within 15 seconds of each other.

Moment data from each heartbeat constitutes the radial volumes centered on each radar. They are broken down into elevation denominated plan position indicator (PPI) sweeps and sent back to the SOCC via its Local Data Manager (LDM), an event-driven data distribution system. The information level of this moment data is termed Tier2a and is high enough for

direct graphical display to end-users, which is done. It has been adjusted by attenuation correction, clutter filtering, and other processes. This data is also fed into meteorological detection algorithms, which along with end-user rules, are input into MC&C to generate the next set of scan tasks, thereby completing the DCAS loop.

Beside driving the DCAS loop, the SOCC also hosts a variety of data fusion algorithms that uses Tier2a data, including the R-MDRS. More relevantly, Tier2a data contains corrected reflectivity and radial velocity, which are crucial inputs for the calculation of true velocity vectors.

2.5 Multi-Doppler Methodology

The R-MDRS's multi-Doppler retrieval implementation adopts algorithms developed by Jay Miller at the National Center for Atmospheric Research (NCAR). It parallels formulations presented by Armijo (1969), Ray et al (1978), and Ray et al (1980); and is outlined in detail in Miller and Anderson (1991).

Conventional meteorological coordinate system consists of northward axis y, eastward axis x, and upward axis z above mean sea level as illustrated in Figure 2.8. The respective velocity components are u, v, and w. Usually Doppler radars detect particles so there is an additional component of motion, terminal velocity of falling particles w_t, to be considered as well.

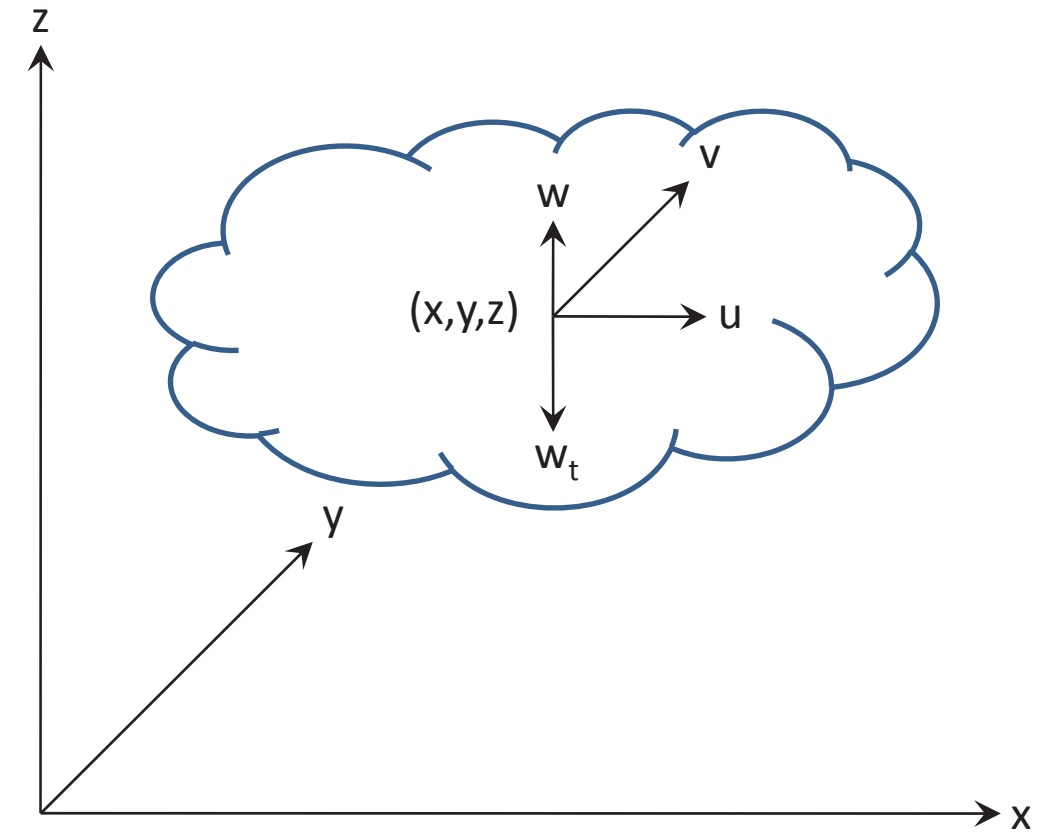


Figure 2.8: Cartesian meteorological coordinate system

The projection of particle motion (u, v, w, w_t) along the Doppler radial direction is

$$v_r = u \sin\phi \cos\theta + v \cos\phi \cos\theta + (w + w_t) \sin\theta \tag{2.11}$$

where ϕ and θ are azimuth and elevation angles of the beam, respectively. The azimuth angle is measured clockwise from zero degree north. True vertical air motion w needs to be separated from the Doppler measurement $W = w + w_t$ through additional information. It should be noted that u, v, and W are average measured values within the sample volume.

It is common to express the radial velocity in terms of Cartesian coordinates because the radial measurements need to be interpolated to a common grid for retrieving air motion vectors. Since radial velocity is the projection of the velocity vector on the vector connecting the observed point and the radar, Equation 2.11 can be rewritten as [**?**]:

$$v_r r = u(x - x_0) + v(y - y_0) + W(z - z_0) \tag{2.12}$$

for a radar at (x_0, y_0, z_0) with slant range

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$$

Further, advection effects due to storm motion need to be taken into account to adjust for the difference between Doppler synthesis time t and the data sample time $t + \Delta t$.

To overcome this problem, Gal-Chen (1982) proposed finding a moving frame of reference i.e. advection, that preserved the stationarity of observations. Non-Cartesian variables such as radial velocity v_r are non-stationary when advected in a moving frame, however the product $v_r r$ is. Consequently Gal-Chen concluded that it would be more accurate to advect the product $v_r r$ rather than v_r solely. Thus for a weather event moving with components (U, V), the above v_r equation is replaced with [5]:

$$\frac{[v_r r]_{t+\Delta t}}{[r]_t} = \left[\frac{u(x - x_0 + U\Delta t)}{r} + \frac{v(y - y_0 + V\Delta t)}{r} + \frac{W(z - z_0)}{r} \right]_t \tag{2.13}$$

Measured radial velocities are first multiplied by slant ranges from the radar at sample time $t + \Delta t$. This product field is then advected at the storm motion to new locations, where it is

divided by slant ranges at the synthesis time t. Coefficients of u and v on the right hand side of 2.13 are modified to account for change in radar pointing direction. For M radars, a linear system of equations can be formed for Doppler synthesis time t,

$$ua_m + vb_m + Wc_m = d_m \qquad m = 1, 2, 3, ..., M \tag{2.14}$$

where a_m, b_m, c_m, and d_m are terms in Equation 2.13.

Approximating the solutions for (u, v, W) by method of least square error, the error equation

$$Q = \sum E_m^2 = \sum (ua_m + vb_m + Wc_m - d_m)^2 \tag{2.15}$$

is minimized with respect to u, v, and W. A proposed solution lies in the following systems of equations:

$$\begin{aligned} u\sum a_m a_m + v\sum a_m b_m + W\sum a_m c_m &= \sum a_m d_m \\ u\sum b_m a_m + v\sum b_m b_m + W\sum b_m c_m &= \sum b_m d_m \\ u\sum c_m a_m + v\sum c_m b_m + W\sum c_m c_m &= \sum c_m d_m \end{aligned} \tag{2.16}$$

or for just solving two unknown quantities u and v with two available radars

$$\begin{aligned} u\sum a_m a_m + v\sum a_m b_m &= \sum a_m d_m - W\sum a_m c_m \\ u\sum b_m a_m + v\sum b_m b_m &= \sum b_m d_m - W\sum b_m c_m \end{aligned} \tag{2.17}$$

Simplifying terms in the three equation system, we can rewrite Equation 2.16 as

$$\begin{aligned} uA_1 + vB_1 + WC_1 &= D_1 \\ uA_2 + vB_2 + WC_2 &= D_2 \\ uA_3 + vB_3 + WC_3 &= D_3 \end{aligned} \tag{2.18}$$

for which the solutions are

$$\begin{aligned} u &= \frac{D_1(B_2C_3 - B_3C_2) - D_2(B_1C_3 - B_3C_1) - D_3(B_1C_2 - B_2C_1)}{D} \\ v &= \frac{D_1(A_3C_2 - A_2C_3) - D_2(A_3C_1 - A_1C_3) - D_3(A_2C_1 - A_1C_2)}{D} \\ W &= \frac{D_1(A_2B_3 - A_3B_2) - D_2(A_1B_3 - A_3B_1) - D_3(A_1B_2 - A_2B_1)}{D} \end{aligned} \tag{2.19}$$

where D is the determinant of the coefficients

$$D = A_1(B_2C_3 - B_3C_2) - A_2(B_1C_3 - B_3C_1) + A_3(B_1C_2 - B_2C_1) \tag{2.20}$$

The vertical component W can be further separated into vertical air motion w and falling speed w_t via reflectivity-fallspeed relations or by incorporating the mass continuity equation into the system.

Similarly for the system of two equations described by Equation 2.17 can be simplified as

$$\begin{aligned} uA_1 + vB_1 &= D_1 - WC_1 \\ uA_2 + vB_2 &= D_2 - WC_2 \end{aligned} \tag{2.21}$$

for which the solutions are

$$\begin{aligned} u &= \frac{D_1B_2 - D_2B_1}{D} + W\frac{B_1C_2 - B_2C_1}{D} = u' + \epsilon_u W \\ v &= \frac{D_2A_1 - D_1A_2}{D} + W\frac{A_2C_1 - A_1C_2}{D} = v' + \epsilon_v W \end{aligned} \tag{2.22}$$

where the determinant D equals

$$D = A_1B_2 - A_2B_1 \tag{2.23}$$

And the terms ϵ_u and ϵ_v are lumped geometric terms. The quantities u' and v' in Equation 2.22 can be used to approximate u and v if ϵ_u and ϵ_v are sufficiently small.

It is necessary to estimate the impact of geometry on the transformation of radial velocities to Cartesian components in order to determine the bounds and validity of the u, v, and W solutions. To begin, it can be seen from Equation 2.19 and 2.22 that u, v, and W are geometrically weighted sums of the interpolated radial velocity that can be described as

$$\begin{aligned} u &= \sum g_{um} v_{rm} \\ v &= \sum g_{vm} v_{rm} \\ W &= \sum g_{Wm} v_{rm} \end{aligned} \tag{2.24}$$

Since radial velocity measurement errors are independent, the variance of the solutions can be written as sums of the radial velocity variances weighted by the squares of the geometric terms

$$\begin{aligned} \sigma^2(u) &= \sum g_{um}^2 \sigma^2 v_{rm} \\ \sigma^2(v) &= \sum g_{vm}^2 \sigma^2 v_{rm} \\ \sigma^2(W) &= \sum g_{Wm}^2 \sigma^2 v_{rm} \end{aligned} \tag{2.25}$$

Assuming all radial velocity variances $\sigma^2 v_r$ are equal, it can be rewritten as normalized variances

$$\begin{aligned} \sigma_N^2(u) &= \sum g_{um}^2 \\ \sigma_N^2(v) &= \sum g_{vm}^2 \\ \sigma_N^2(W) &= \sum g_{Wm}^2 \end{aligned} \tag{2.26}$$

These normalized variances can be compared to determine the geometric impact and consequently the validity of the u, v, and W solutions.

Low elevation scans such as those performed by IP1 radars produce radial velocities that have small vertical components. With the above assessment and from a geometric standpoint, the vertical wind component W solved from Equation 2.19 is usually unreliable. However with u and v solutions being much more accurate, the mass continuity equation can be used to obtain the vertical wind component. For this application, the mass continuity equation becomes

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = 0 \tag{2.27}$$

ρ is the air density whose horizontal gradients and local variations are considered negligible. It is modeled as a function of the altitude

$$\rho = \exp(-zG) \tag{2.28}$$

where G is a user-specified parameter. Integrating Equation 2.27 to solve for the net mass flow in the vertical direction yields

$$\int_{z_k}^{z_{k+1}} \frac{\partial(\rho w)}{\partial z}\,\mathrm{d}z = -\int_{z_k}^{z_{k+1}} \rho\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\mathrm{d}z \tag{2.29}$$

For the discrete grid space, finite difference form is utilized between the previous p and the current c levels

$$(\rho w)_c = (\rho w)_p - \delta\Delta z\overline{\left[\rho\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right]}_{p,c} \tag{2.30}$$

where

$$\delta = \begin{cases} +1 & \text{for updward integration} \\ -1 & \text{for downward integration} \end{cases}$$

The overbar value represents the average divergence of previous and current levels. The boundary condition $(\rho w)_b$ must be specified for every grid point at the bottom of the domain for upward integration, or at the top of the domain for downward integration. A variational integration scheme also exists where both upward and downward integrations are performed. The solution is an averaged value of the two directional integrations, which attenuates the error carry-overs exhibited by single-direction integrations. Downward and variational schemes are viable only when radar measurements top out the storm, otherwise the upper boundary condition becomes ambiguous.

Chapter 3

SYSTEM DESCRIPTION

IP1's R-MDRS is a composition of many programs, scripts, and event-driven processes. These components are required to interact seamlessly under precise timing to ensure robust function of the system. This chapter provides an indepth description of each component, their interactions, the overall process flow, and the design philosophy.

3.1 System Overview

As shown earlier in Figure 3.1, the R-MDRS is a subcomponent of the overarching IP1 DCAS operation loop that enables accurate and real-time production of air kinematics within the IP1 test bed during severe weather events. It shares computing and data resources with other simultaneous operations running on the SOCC, but is an autonomous processes. Once executed, the R-MDRS creates an independent processing branch until finished. It then awaits the execution order of the next DCAS loop iteration. Since IP1's operation loop is strictly tied to the radar network's scanning heartbeat, each R-MDRS execution must also fall within the heartbeat time. In actuality, when overhead processing time is taken into account, the R-MDRS processes need to complete within approximately 75% of the heartbeat time to reliably avoid skipping data sets. Figure 3.1 illustrates an overview of the R-MDRS operation that occurs on each heartbeat.

The overall process is divided into two major sections, a preprocessing phase that integrates the R-MDRS proper with the IP1 DCAS loop, followed by the core R-MDRS itself. Preprocessing consist of data distribution from the radar network via the LDM and ingestor procedures that prepares the data for R-MDRS intake. While the LDM is a system-wide feature used for all of IP1's data products, the ingestor procedures are specifically designed to accommodate volume-based processing applications such as R-MDRS.

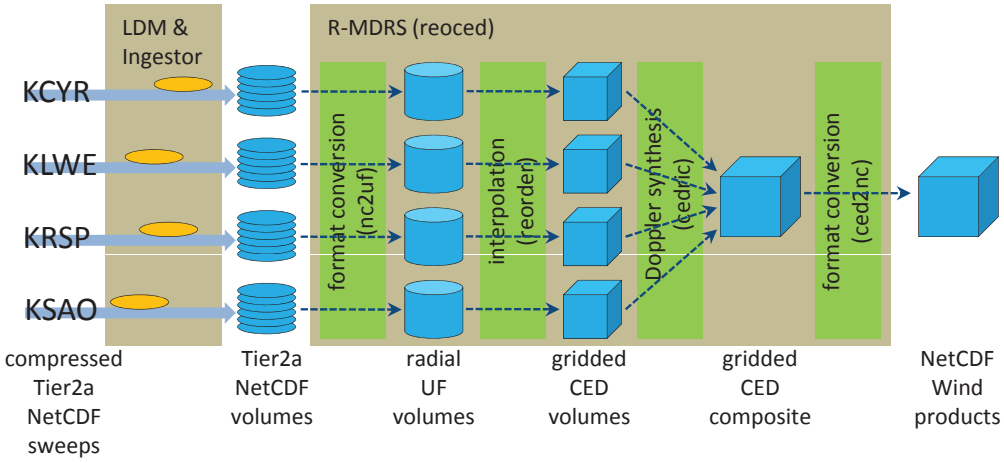


Figure 3.1: IP1 real-time multi-Doppler retrieval operation

The LDM is an event-driven process in the sense that it distributes the radar data to the SOCC as soon as it is generated. These data are compressed Tier2a sweeps in Network Common Data Form (NetCDF) format. NetCDF is a prevalent data format for meteorological research and is the standard for all CASA information exchange. In addition to data distribution, the LDM can also execute programs to operate on the received data. This feature is exploited on every loop to invoke the Ingestor program to decompress the incoming data, extract their scanning information, synchronizing them to respective radar and volumes, and finally executing the R-MDRS itself. Thus by inheritance, the whole R-MDRS is event-driven and automated.

The R-MDRS proper consists of both new and legacy software controlled by a PERL meta-script. This meta-control design is an important feature of the R-MDRS, as it greatly improves the useability of the conglomeration of software that makes up the R-MDRS. Externally, the R-MDRS appears as one functional block that intakes Tier2a NetCDF sweeps and outputs 3D wind products.

In more detail, the R-MDRS revolves around two components, data interpolation and Doppler synthesis. These functions are implemented by legacy software known as `reorder` and `cedric`, respectively. The use of these legacy software to perform core functionalities is justified by several reasons. `reorder` and `cedric` are developed cooperatively alongside each

other at the National Center for Atmospheric Research (NCAR) for the specific purpose of gridding radial data onto a common Cartesian grid and performing multi-Doppler synthesis, albeit not in real-time. This cooperative development results in an integrative multi-Doppler synthesis algorithm that guarantees inter-compatibility and execution fluidity. Consequently the `reorder cedric` duo has been a widely used and well-supported toolset with proven proficiency in terms of both accuracy and speed. For these reasons they are chosen for the R-MDRS in favor of other implementations or new development. These core components are wrapped by peripheral software specifically developed to retrofit them for useability and integration with real-time IP1 applications. The PERL meta-script that controls this entire process is called `reoced`, a conjunction `reorder` and `cedric`. Collectively, these components form the R-MDRS core.

The first R-MDRS subsystem is the `nc2uf` format conversion program that acts as a data adapter for the interpolation software `reorder`. It converts the Tier2a NetCDF files into Universal Format (UF) files. UF is a position-defined ray-based data format that is commonly used by legacy radar processing software, such as `reorder`. The conversion also combines the separate sweep files into single volume files for each radar.

The UF volumes are then interpolated from its radial dimensions to Cartesian space by `reorder`. This process is commonly referred to as "gridding". Mapping to Cartesian space provides a common coordinate system for overlapping scans in the test bed. This is a necessary geometric condition for performing the Doppler retrieval methods outlined in Section 2.5. `reorder` implements radial to Cartesian interpolation through range-weighted averaging of range gates to single grid points. Spheres of influence surrounding each grid points are used to determine the boundaries of range gates being averaged to that point. A variety of schemes for both range-weighting and sphere of influences are possible with `reorder`.

The actual implementation of Section 2.5 is performed by `cedric`. Gridded volumes from each radar is synthesized by `cedric` into a single composite grid with resolved u, v, and w

fields. `cedric` outputs the products in NetCDF, but requires a final format conversion process provided by the program `ced2nc` to convert results to CASA's NetCDF standards. R-MDRS ends by cleaning its memory and intermediary files before awaiting to be invoked again for the next loop iteration. This practically implies that the R-MDRS is constrained by IP1's heartbeat. While falling behind due to large data sets or strained computing resources is not fatal to R-MDRS processing, it does lead to skipping of data sets.

3.2 LDM and Ingestor Preprocesses

SOCC LDM's real-time data streaming from the four radars is not natively compatible with the volume-based processing of the R-MDRS. The LDM is an event-driven collection process that gathers meteorological data in small and potentially out-of-order chunks. While the R-MDRS is a batch process that simultaneously synthesizes entire volumes of data from multiple radars. To address these two incompatible operating models, the Ingestor program is implemented and embedded into the LDM. This section provides a description of the LDM and Ingestor preprocesses.

A brief overview of the LDM, the Local Data Manager (LDM) is developed and maintained by the Unidata Program Center. Unidata is a NSF sponsored program to promote IT and software technologies to make the best use of atmospheric and related data for higher education and research. The LDM is created as a software system for efficient and reliable distribution of arbitrary but finite-sized data via the internet. It operates on a client-server model with the data source being the servers and the data sinks being clients. Thus for the IP1 test bed, each radar hosts a LDM server, and the LDM residing on the SOCC is technically a client downloading data from each radars' LDM servers. Other LDM clients can also be set up on any arbitrary internet-connected device to stream from the IP1 radars, creating an efficient data-pull distribution system. Throughput is limited by the upload speeds of the radar nodes.

An important feature of the LDM is its ability to execute user-defined processes on all incoming data. As mentioned in the previous section, this is the basis of making the entire SOCC operation event-driven, beginning with the Ingestor preparing the incoming data for R-MDRS processing. Figure 3.2 illustrates the entire LDM Ingestor preprocess in detail.

Data streams via the LDM is dependent on the data generation rate at the radars as well as network latency. For this reason it is not guaranteed that data acquisition times will be simultaneous for simultaneous data sampling times, although it is required to be within a reasonable time frame. Latency issues may also cause radars' sweeps to arrive out of order,

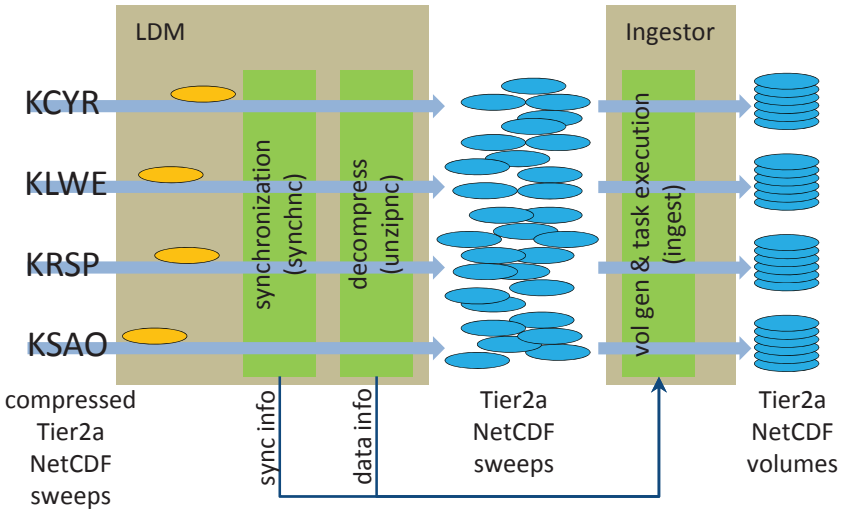


Figure 3.2: LDM and ingest pre-processes

or be lost altogether. In essence, IP1's LDM tries its best to pull any available data from the radars, but it cannot be aware of the completeness and the temporal sequence of the data. That function falls to the Ingestor.

The Ingestor technically consist of three separate processes that share an inter-process communication (IPC) channel. They are in order of execution, `synchnc`, `unzipnc`, and `ingest`. `synchnc` parses incoming file name and passes this information to the IPC channel. As IP1 sweep file names contain the radar name and time stamp of the data sampling time, `synchnc` serves as a method to enforce the correct grouping and sequence of the radar data sweeps.

`upzipnc`, as its names suggest, decompresses the incoming data sweeps and archives it to disk. Data from the radars are compressed for the sake of preserving fidelity and increasing speed during transmission over the internet. Coincidentally, NetCDF formatted files are also very compressible due to its constant dimensioned arrays; sometimes achieving compression ratios approaching an order of magnitude. In addition to decompression, `unzipnc` also extracts the scan information from each data sweep, including scan id's, elevation angles, and scan times. This information is also passed to the IPC channel.

`ingest` processes the data information from the IPC channel left by `synchnc` and `unzipnc`, and organizes the sweeps into respective volumes for each radar. `ingest` does not physically change the sweep data, but instead create meta-files that lists all the sweeps that make up all the radar volumes of a heartbeat. These volume lists organize the incoming data streams into batches that are suitable R-MDRS's volume-based processing model. Note the following volume list example:

```
4 2 3 3 2
/net/makalu/radar/CASA/LDM/20090514/cyril/1.00/KCYR_20090514-023108.nc
/net/makalu/radar/CASA/LDM/20090514/cyril/2.00/KCYR_20090514-023116.nc
/net/makalu/radar/CASA/LDM/20090514/lawton/1.00/KLWE_20090514-023107.nc
/net/makalu/radar/CASA/LDM/20090514/lawton/2.00/KLWE_20090514-023126.nc
/net/makalu/radar/CASA/LDM/20090514/lawton/3.00/KLWE_20090514-023146.nc
/net/makalu/radar/CASA/LDM/20090514/rushsprings/1.00/KRSP_20090514-023104.nc
/net/makalu/radar/CASA/LDM/20090514/rushsprings/2.00/KRSP_20090514-023123.nc
/net/makalu/radar/CASA/LDM/20090514/rushsprings/3.00/KRSP_20090514-023143.nc
/net/makalu/radar/CASA/LDM/20090514/chickasha/1.00/KSAO_20090514-023105.nc
/net/makalu/radar/CASA/LDM/20090514/chickasha/2.00/KSAO_20090514-023112.nc
```

The first line of numbers indicates the number of radars and the respective number of sweeps from each radar in the batch, thus in the above example, the batch contains data from four radars, with two sweeps from Cyril, three from Lawton, three from Rushsprings, and two from Chickasha. Subsequently, the paths of the data sweeps are listed.

The generation of volume lists is both a simple and versatile method for providing complete definitions of the data set. The versatility refers to the arbitration in how the volume list is populated. For R-MDRS processing, the batches are defined by a combination of the heartbeat and elevation angles. In the volume list example listed above, note that sweeps from each radars are in order of ascending elevation angle. Since the IPC channel contains

elevation angle information of each sweep, `ingest` can detect the beginning of a new volume scan when the elevation angle suddenly drops back down to the lowest, in this case 1.00 degrees.

In addition to volume initialization, an arbitrary time delineation is also needed to mark the end of the volume. This time range should approximate the time it takes for the radars to complete a full volume scan. For normal IP1 operations, this volume scan time is tied to the system heartbeat of one minute. Lastly, to synchronize the multiple volumes of each radar, all the volumes are required to begin within a short time frame of each other. This ensures a sufficiently 'instantaneous' snapshot of the weather space. Note that while the timing restrictions pertain to sample time, a sweep that arrives too late due to network latency or other issues will nevertheless be excluded from a volume list. These are the rules for populating the volume lists for R-MDRS operations. For other applications, they can be customized to generate other suitable data batches.

3.3 Data Formats and Conversion

The utilization of legacy software in the R-MDRS necessitates the `nc2uf` format conversion tool. CASA data exchange such as IP1 Tier2a data are formatted in Network Common Data Form (NetCDF). Legacy radial-to-Cartesian interpolation software `reorder` intakes data formatted in UF. While both formats are prevalent in meteorological research, they are separated by several generations in development and are seldom used together. This section aims to provide an understanding of the mapping performed by `nc2uf` between these two very different data formats.

3.3.1 NetCDF structure

The '**Net**work **C**ommon **D**ata **F**orm', or NetCDF, is a data format also developed and maintained by the Unidata Program Center. Unidata created NetCDF to provide a common data platform for all Unidata applications and information exchange. Eventually however, it also became widespread beyond Unidata applications and meteorological research due to its comprehensive support and I/O libraries including C, C++, FORTRAN, PERL, and others.

In the meteorological research environment where mass amounts of volumetric data is shared across multitude of sensors, networks, and computer architectures, NetCDF successfully adopts a self-describing, machine-independent data structure tailored for network distribution of array-based data. "Self-describing" means that datasets include header information that define the data they contain. Machine-independent" or "portable" means datasets can be accessed by different computer architectures regardless of the memory system they use store floats, integers, and various other data types. A NetCDF dataset generated on one computer in FORTRAN can be accessed on another computer in C without any intermediary conversions. To accommodate array-based data, a NetCDF dataset consists of three components: dimensions, variables, and attributes, all of which have a name and an id number. These components are used together to capture the meaning of the data and relations among data fields. The example below illustrates a simple NetCDF product produced by R-MDRS.

```
netcdf COMP_20090514-023004 {
dimensions:
        Alt = 20 ;
        Lat = 281 ;
        Lon = 281 ;
variables:
    float Alt(Alt) ;
        Alt:Units = "Km" ;
    float Lat(Lat) ;
        Lat:Units = "Degrees" ;
    float Lon(Lon) ;
        Lon:Units = "Degrees" ;
    float U(Alt, Lat, Lon) ;
        U:Units = "MetersPerSecond" ;
    float V(Alt, Lat, Lon) ;
        V:Units = "MetersPerSecond" ;
    float W(Alt, Lat, Lon) ;
        W:Units = "MetersPerSecond" ;
    float CorrectedReflectivity(Alt, Lat, Lon) ;
        CorrectedReflectivity:Units = "dBZ" ;
global attributes:
    :TypeName = "all" ;
    :DataType = "AltLatLonGrid" ;
    :Latitude = 35.457539850056 ;
    :Longitude = -98.8681012556291 ;
    :Altitude = 500.f ;
    :Time = 1242268203 ;
```

```
        :FractionTime = 0. ;

        :AltGridSpacing = 0.5f ;

        :LatGridSpacing = 0.00449957f ;

        :LonGridSpacing = 0.005481437f ;

        :MissingData = -99900.f ;

        :RangeFolded = -99901.f ;

    }
```

A dimension may be used to model real physical dimensions such as altitude or time, but it may also be used to as a data index such as station number. All dimensions must have an arbitrary positive integer length, except for the first, which can have a length of 'UNLIMITED'. This unlimited dimension acts as a record index that allows NetCDF datasets to be appended indefinitely. For radar data, the record dimension is usually time. In the above example, the data set is defined in in a three dimensional space by `Alt`, `Lat`, and `Lon`.

Variables contain the bulk of the data in a NetCDF dataset. It is made up of an array of values of the same type and is defined by its data type and dimension list. The data type such as float, integer, etc... applies to the values of the array, and the dimension list defines the shape of the array. A scalar variable is not indexed to any dimension and thus does not have a dimension list associated with it. The dimensions are ordered from most significant to least, meaning the last dimensional index varies the fastest when indexing the variable. In the above example NetCDF, variable `U` is defined as `float` with dimension list `Alt`, `Lat`, and `Lon`. This means that the variable `U` is a 20 by 281 by 281 array of 4-byte floating point values. In memory space, it is sequenced by longitudinal indexes, then by latitudinal, and lastly by altitude.

A variable can have the same name as a dimension, it makes no difference to the NetCDF environment. However it is conventionalized to mean that the eponymous variable is a 'coordinate variable' that defines the physical coordinate corresponding to that dimension. In the above example, the variables `Alt`, `Lat`, and `Lon` are coordinate variables.

Attributes are used to store supplemental information about data and is largely optional. Information such as units of measure, project names, user comments, etc... are most conveniently conveyed as attributes. They are attached either globally to the NetCDF or to specific variables. Note that since they have no dimensions, they can be used to store scalar data. For the R-MDRS, vital information regarding timing, interpolation parameters and test bed attributes are conveniently packaged as global attributes in the NetCDF data flow.

3.3.2 UF structure

The **U**niversal **F**ormat, or UF, is a much older data format originally proposed by Barnes (1980). It is specifically created for Doppler radar data, and is structured around scalable records that correspond to rays - data acquired for a given pointing direction. Being a widespread data format for radar information exchange in the 80s and early 90s, many legacy radar data processing software utilize UF data standard. Due to memory-saving priority of the past, UF field names must be no more than two ASCII character. It can be arbitrary, but usually conforms to SIGMET standard field mnemonics listed below:

```
DZ - Reflectivity factor (dBZ)

CZ - Corrected reflectivity factor (dBZ)

DR - Differential reflectivity, ZDR (dB)

PH - Differential phase, PhiDP (deg)

KD - Specific phase, KDP (deg/km)

RH - Cross-polar correlation, RhoHV (0 to 1)

VR - Raw velocity (m/s)

VT, VE - Velocity thresholded on NC (m/s)

VF - Velocity with good/bad flag on least significant bit (m/s)

VP - Velocity thresholded on received power (m/s)
```

SIGMET, or Significant Meteorological Information, is a weather advisory that contains meteorological information concerning the safety of all aircraft.

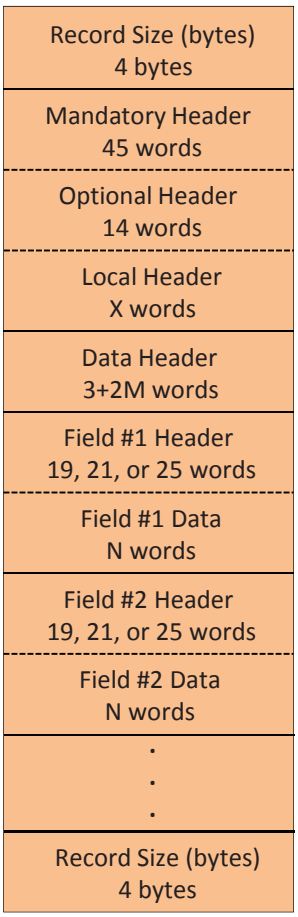


Figure 3.3: UF record strucure for a ray with M fields and N range gates

Each record in an UF dataset completely defines the data values, the radar information, and the scanning parameters of a ray. Large rays with many variables may be broken down into several records, but this is a feature seldom used in modern computer systems. Thus for R-MDRS applications, a UF record strictly corresponds to an entire ray. Records are organized into two-byte aligned, byte-swapped (big Endian) format illustrated by Figure 3.3. The exception is the beginning and end of the record, which are 32 bit memory spaces indicating the number of bytes in the record, these values are still byte-swapped. All data values are also stored as short integers (2 bytes), which means floating point values are scaled by an integer multiplier to accomplish this. This scaling obviously truncates and reduces precision, but gain a memory advantage.

Unlike NetCDF, UF does not have any active development environment or official support. As a result, a UF library called `ufsxz` is implemented specifically for the R-MDRS. The library provides a C interface for generation and manipulation of UF files and adheres as much as possible to the original definitions of the UF data standard. While developed for the R-MDRS, `ufsxz` is not proprietary to it, and can be used by any software requiring manipulation of UF data. For a detailed user guide of `ufsxz`, please consult Appendix D.

3.3.3 NetCDF to UF mapping

As illustrated in Figure 3.1, the NetCDF to UF conversion process is performed by the `nc2uf` program. `nc2uf` is a C program built on the Unidata NetCDF and `ufsxz` UF libraries described in previous sections. Like `ufsxz`, `nc2uf` is created for the R-MDRS, but is a general tool that can be used to convert any NetCDF to UF. However note that while not required, the input NetCDF should be a radially-coordinated data structure, since UF format is inherently radial. Tier2a data streaming from the IP1 network are radial sweeps, and thus meet this criterion.

`nc2uf` is versatile because its execution is driven by configuration file. Most configuration parameters set UF ray header information such as radar names and ray time to corresponding fields in the NetCDF file. From previous examination of the two data formats, it is clear that UF has a lot more redundant header information than NetCDF. For example, the radar name parameter would be repeated in the header of every record of the UF data file, while in the NetCDF data file it is specified once as a global attribute. This leads to single-to-multi-point mapping as NetCDF data are converted to UF. The configuration options for `nc2uf` must be able to accommodate these rather complex parameterizations. When integrated into the R-MDRS, `nc2uf` configuration are auto-generated by the overarching `reoced` meta-script. For a detailed user guide of `nc2uf` and the `uf2sxz` library, please consult Appendix D and C.

Aside from header information, the majority of `nc2uf`'s compute cycles are dedicated to the transformation of the data itself. Conventionally this is a straightforward and computationally efficient process. However `nc2uf`'s task is complicated by the byte-swapped and

scaled nature of UF data values. The conventional copy instruction is now expanded to a sequence of multiplication, floating point to short integer conversion, byte swapping, and then copying. In compute cycles, this represents an increase by about an order of magnitude. Thus for R-MDRS's real-time performance, `nc2uf` is parameterized to only transform radial velocity and reflectivity variables.

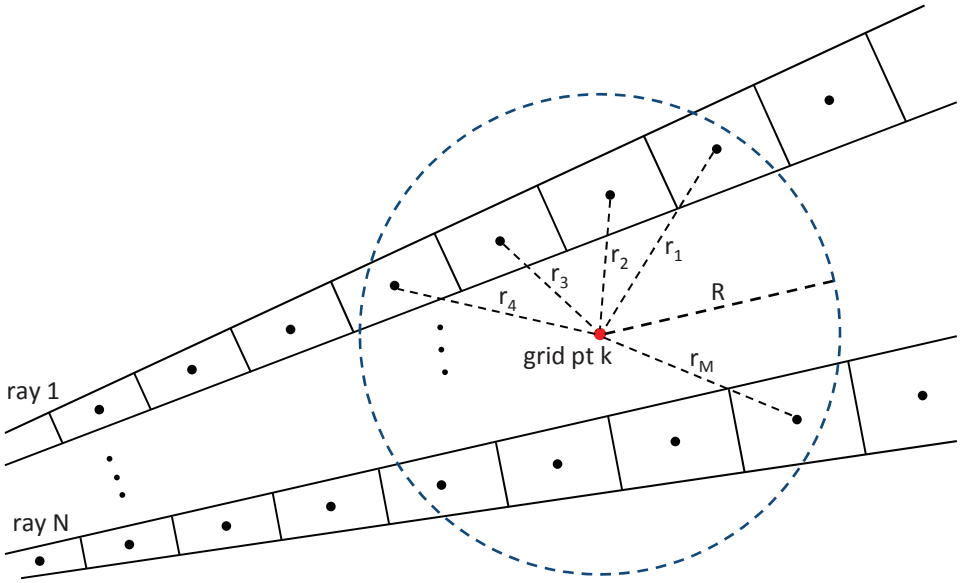


Figure 3.4: Interpolation via range-weighted averaging within sphere of influence

3.4 Data Interpolation

Interpolating radial volume to Cartesian space is a pre-requisite of the final Doppler synthesis and retrieval, as is evident in the equations outlined in Section 2.5. For the R-MDRS, this process is accomplished with the `reorder` program developed by NCAR. This section will examine the methodologies, implementation, and limitations of this interpolation process.

3.4.1 Gridding principles

The R-MDRS interpolation performed by `reorder` is characterized as distance-weighted averaging of range gates to various grid points based on radius of influence criteria. This creates two sets of choices, first for selecting the weighting scheme, and second for choosing the radius of influence function. Figure 3.4 illustrate these geometric relations.

For distance weighting to a certain grid point k, each range gate in the input UF rays is assigned its Cartesian coordinate in (x, y, z). Then its distance r to the grid point is compared to the grid point's radius of influence R. The relationship of this comparison is the weighting scheme. The R-MDRS syetem uses a conventional Cressman weighting scheme, for which the

mth range gate's weight becomes

$$W_m = \frac{R^2 - r_m^2}{R^2 + r_m^2} \qquad m = 1, 2, 3, ..., M \tag{3.1}$$

where the radius of influence R is

$$R^2 = dX^2 + dY^2 + dZ^2 \tag{3.2}$$

The dX, dY, and dZ values that define R are user-specified parameters. For constant dX, dY, and dZ, the weighting volume literally becomes a *sphere* of influence. The final data value at grid point k is then

$$a_k = \frac{\sum_{m=1}^{M} W_m a_m}{\sum_{m=1}^{M} W_m} \tag{3.3}$$

`reorder`'s available weighting schemes include uniform, closest point, and exponential. For exponential, the weighting scheme is

$$W_m = \exp\left[\frac{-G r_m^2}{R^2}\right] \qquad m = 1, 2, 3, ..., M \tag{3.4}$$

where G is a user-specified parameter.

Aside from user-specified dX, dY, and dZ defining a constant radius of influence R, R can also be defined by components of azimuth, elevation, and range. In this case, user specifies ΔAzimuth, ΔElevation, and ΔRange, from which dX, dY, and dZ are calculated for each grid point k. This has the effect of increasing the radius of influence for grid points further away from the radar. For scans in which there are spatial gaps in between, especially further away from the radar, this scheme can help interpolate the otherwise missing grid points. However it also causes blurring of weather features under many circumstances. With the proper selection of beamwidth and scanning elevations, gaps in scanning volumes can be avoided, which also eliminates the need for using this variable radius of influence scheme. IP1 radars are designed and operated to use a combination of beamwidth and scanning elevations that leave no gaps within their volume scans, thus the R-MDRS uses constant radius of influence for interpolation.

3.4.2 Interpolation implementation

During IP1 operation, `reorder` interpolates each radar to a common Cartesian grid centered at $(34.8276\,^{\circ}N, -98.1007\,^{\circ}W)$, a point approximately at the center of the test bed. The common grid extends from -70 to 70 km in the east-west direction and -70 to 70 km in the north-south direction, extending just beyond the full coverage area of the radars. Altitude extends from 0.5 to 10.0 km sea level, ensuring topping out of most storm cells. Resolutions on all three axes are set to 0.5 km, a value chosen to limit the number of grid points to be computationally feasible in real-time. In addition, as the prior NetCDF to UF conversion process only transformed radial velocity and reflectivity values, only these will be interpolated, further reducing computational load.

Each `reorder` execution is driven by a single configuration file that describes the data input, the grid dimensions, and interpolation parameters. The meta-script `reoced` that controls the R-MDRS process dynamically generates new `reorder` configuration files for each volume. For a detailed operation manual of `reorder`, please consult Oye & Case (1995). Since interpolation is done on a per-radar basis, IP1's test bed requires four `reoder` executions every heartbeat. When performed sequentially with the aforementioned grid parameters, the total interpolation process accounts for approximately 60% of total processing time.

`reorder` is integrated with the NetCDF libraries to be able to directly output interpolated results in NetCDF format. This function is useful for verification of interpolation results. However for IP1 operations, `reorder` is parameterized to output interpolated results in a proprietary binary format for the Doppler synthesis program `cedric`. `reorder` is one of very few programs that natively outputs `cedric`'s proprietary format due to their simultaneous and complementary development. This compatibility makes the `reoder-cedric` duo a very convenient choice for performing the entire interpolation-to-Doppler-retrieval process; as opposed to having to cross-adapt separate interpolation and retrieval programs. When factored in along with their respectable capabilities and speeds, the choice for using the `reoder-cedric`

duo becomes clear. This merit will be further supported by the strengths and advantages of `cedric` explained in Section 3.5.

3.4.3 Limitations of interpolation

R-MDRS's implementation of `reoder` is not without issues, this section addresses the limitations of `reorder` that the R-MDRS is not yet able to overcome. The two primary limitations involve computational model and data resolution, both of which ultimately tie into the speed limitation of `reorder` for R-MDRS operations.

As aforementioned, the R-MDRS executes `reorder` for each radar *sequentially*. This means four `reoder` executions per heartbeat for IP1's current radar configuration, accounting for approximately 60% of total R-MDRS processing time. For normal IP1 scanning mode heartbeat of 1 minute, and with SOCC's current computational capabilities, this approximately turns out to be 30 seconds of interpolation out of a total R-MDRS processing time of 50 seconds. With variability in network latency and data size, R-MDRS sometimes surpasses the heartbeat, resulting in skipped data sets. From this assessment, the most obvious and potentially biggest speed gain for the R-MDRS is to parallelize the interpolation process over all the radars. This can potentially quarter the 30 seconds down to 8, providing an ample overhead time to avoid data set loss.

R-MDRS's modular control of subroutines via the meta-script `reoced` makes it very easy to initiate multiple processing threads and execute separate programs over each of them. Utilization of PERL's threads library allows for `reoced` to execute separate instances of `reorder` for each radar. However this is limited by the computational hardware R-MDRS resides on, in this case the SOCC compute server. Truly scalable multi-threaded processing requires physically independent processing cores each with their own level 1 and level 2 caches. With the multitudes of processes running on the SOCC as illustrated in Figure 2.7, it is simply unfeasible to reserve the four required independent processing cores for the R-MDRS alone. However this limitation is not a serious constraint that questions the fundamental design of

the R-MDRS because it can be readily solved by scaling hardware to match IP1's requirements at nominal costs.

The second limitation addresses the resolution limit of the R-MDRS in real-time. In cubic relations, doubling the resolution will multiply the number of grid points by eight, scaling interpolation times by eight as well. Under current SOCC processing capabilities and a 1 minute heartbeat scanning mode, IP1's spatial domain of 140 by 140 by 10 km can be interpolated in real time at a resolution of 0.5 km in each dimension. This equates to 1,568,000 grid points interpolated according to Equation 3.3. However recall from Section **??** that IP1 radars have radial range resolutions of 75 meters. For a Cartesian interpolation to map this data on same scale order, a much finer grid resolution on the order of at least 100 meters would be required. Refining the resolution from 500 meters down to 100 meters would multiply the number of grid points by 125, resulting in 196,000,000 points. As with any brute-force method, this is unfeasible and unnecessary. It is also impossible as `reoced` has a 500 by 500 by 500 grid size limit to control memory usage, though this limitation can be readily removed in the code to accommodate modern computing capabilities.

For any moment in time, significant weather features such as squall lines and tornadic spins would reasonably occupy only a small portion of the IP1 test bed. Encapsulating the interpolation domain to only areas with significant weather can cut out a great majority of unnecessary grid points. The R-MDRS already does this to a degree with the assistance of the LDM and Ingestor. When no significant weather occurs within the domain of a particular radar, that radar's data is not included in the input volume list, thereby cutting out the interpolation of that radar altogether. An even more aggressive approach is to minimally encapsule the domain of interpolation based on significant weather echoes.

High resolution measurement and plotting of wind and tornadic events in IP1 require significant zooming into the test bed. Staying within the 1,568,000 grid point throughput previously calculated, a nominal 100 meter resolution grid would be 12.5 km by 12.5 km by 10km. This is about 0.8% of the area of IP1's test bed, thus the biggest dilemma with

this approach is knowing where to zoom into. Currently, with no reliable way of pinpointing locations of significant fast-moving weather features, real-time Multi-Doppler retrieval can only be performed on a macro-scale. Finely detailed air motion in max resolution can only be produced as a post-process, not in real-time. Unlike the previous limitation of compute cores, this issue is not simply addressed by merely scaling computing capability. Aside from unfeasible brute force using world-class super computers, a smarter approach needs to be developed to dynamically encapsulate the interpolation domain to minimize the number of necessary grid points. A plausible approach for vortex detection lies in sensing regions of high vorticity in the test bed at low resolutions, and spinning off processing threads for further high resolution Doppler analysis at candidate regions. This will be discussed in more detail in Chapter 5 as a part of future works.

3.5 Doppler Synthesis and Retrieval

The core Doppler retrieval function of the R-MDRS is applied on the output results of `reorder`. The program that performs this task is known as **C**ustom **E**diting and **D**isplay of **R**educed **I**nformation in **C**artesian space, or `cedric` for short. This section will explore the features and operations of the Doppler retrieval process.

3.5.1 `cedric` Usage

The `cedric` program is developed by NCAR's Mesoscale and Microscale Meteorology Division in the early 90's with the primary purpose of multi-Doppler wind analysis. `cedric` provides a wide and in-depth variety of options for calculating physical quantities, editing the data, and filtering. In addition, users have flexibility in manipulating the data by stacking functions to operate on an entire field. These factors extends `cedric` beyond its original objectives and intended lifespan, making it a very versatile tool for manipulating and analyzing any kind of gridded numerical data. Though it has been around for a very long time, `cedric` still maintains very competitive capabilities and speeds.

`cedric` possesses a peculiar execution model that is inherited from its original development on punch card computers. In its oldest form, each operation is explicitly defined by a series of card images. A sequence of these card images operating on the same data set performs the desired algorithms and calculations. The card images have since then been digitized into text configuration files, however it maintains its stringent formatting and spacing requirements. Below is an example a `cedric` configuration segment containing the `CHANGE` operation and a `COMMENT`.

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
CHANGE  NAMFLD  ISPCNX  XLOC    YLOC    ZLOC    VALUE
```

Each line in the configuration is exactly 80 characters long and corresponds to one of the original "card image", thus they are simply referred to as such. The `COMMENT-COMEND` comment segment delineates each card image into 10 8-character wide parameter fields labeled P1 through P10. The first field P1 always contains the keyword that identifies the card image. In the above example, the first card image is identified as a `COMMENT` segment, indicating all subsequent content to be a comment string until terminated by a `COMMEND` card image. The next card image is the `CHANGE` command and its parameters, dictating the replacement of the `NAMFLD` field value at (`XLOC`,`YLOC`,`ZLOC`) with `VALUE`. Commands such as `CHANGE` occupy one single card image. More complex commands with more than 9 parameters require multiple card images and are referred to as stacks. Stacks must be terminated by an `END` card image.

The configuration format is archaic, but retains exact and indepth control of function parameters. It also clearly illustrates the process flow as data is being manipulated. When integrating `cedric` into the R-MDRS, this cumbersome configuration process is conveniently overcome by having the overarching `reoced` meta-script auto-generate the configuration files. This drastically improves the useability and tuning capabilities of `cedric` while reducing the chance of critical operation errors caused by likely typos in the configuration.

`cedric` directly implements the Doppler retrieval methodologies outlined in Section 2.5, taking appropriate approximations and discretizations when necessary. Piecewise, `cedric` implements multi-Doppler analysis in the following order,

1. Intake Cartesian coordinated data sets that contains radial velocity fields. The data sets must be in `cedric` binary format, either produced by `reorder` or otherwise through the `cedric` I/O library.

2. Set appropriate coordinate boundaries. If proper housekeeping is done during interpolation by `reorder`, this will be done automatically.

3. Perform Doppler synthesis with the `SYNTHES` operation. This step corresponds to solving the least square error solution system of equations outlined in Section 2.5 for (u, v, W).

4. Calculate horizontal divergence values for (u, v) solutions using the `DIVR` function. These form the terms of the mass continuity relation outlined in Equation 2.27.

5. Integrate the divergence terms using the `INTEGR` operation. This solves the discrete mass continuity equation described by Equation 2.30.

3.5.2 Data preparation

The `cedric` program utilizes its own proprietary data format and a sequential execution model. Its proprietary format, similar to UF, is a position-dependent, non-self-describing legacy format prioritized for memory conservation rather than ease of use. However as aforementioned, `reorder` outputs this format specifically for feeding into `cedric`, and `cedric` itself has been augmented to output products in NetCDF format. There is also an entire `cedric` input and output library for integrating user applications with `cedric`. As a result of these factors, `cedric`'s proprietary data format becomes largely invisible to the end-user, and thus unimportant to the discussion of the R-MDRS. For technical details about `cedric` data structure and function calls, please consult Appendix D of NCAR's `cedric` manual (Miller & Fredrick 1998).

In regards to coordinate definitions, it is mentioned in Section 3.4.2 that `reorder` interpolates each radar to a common Cartesian grid representing the entire test bed. This means data sets from each radar are already offset to correct locations in the test bed and relative to each other. During `cedric` operations, they simply need to be superimposed to correctly create the regions of dual- and multi-Doppler overlap as illustrated in Figure 3.5. This superposition inherently imposes resolution requirements on the data grids. `cedric` can handle data grids of mixed resolutions, but they must be integer multiples of each other. The R-MDRS is configured with one common resolution for all its interpolated data sets to avoid complications. Within IP1's geographic boundaries, compromising between covering the entire test bed and keeping a sustainable number of grid points for real-time performance results in a nominal resolution of 0.5 km in x, y, and z directions.

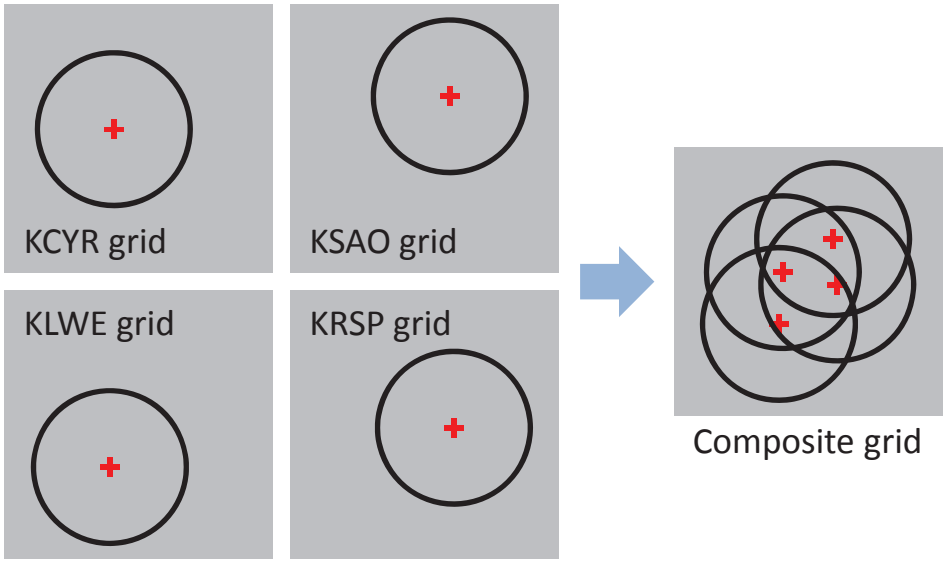


Figure 3.5: Grid synthesis via superposition for `cedric` analysis

3.5.3 Synthesis

Synthesis of radial velocity information is performed by the `SYNTHES` operation. This is the core process in `cedric` that implements the lease square error solution presented in Equations 2.15 to 2.22. Optionally, the `SYNTHES` operation also performs the advection correction detailed in Equation 2.13 as well as importing up to five additional data fields from input data volumes. `SYNTHES`'s command structure is illustrated as follows:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
SYNTHES NSNVOL  NRADS   DTEST1  DTEST2  DTEST3  ITWOEQ  ITANAL  STMDIR  STMSPD
        LUNIT   NAMVOL  IBEGTM  IENDTM  IREWND  NAMEVEL LADTYP  NAMTIM  NADFSW
        INPUT   NAMINP  NAMINP  NAMINP  NAMINP  NAMINP
        OUTPUT  NAMOUT  NAMOUT  NAMOUT  NAMOUT  NAMOUT
END
```

`P2 NSNVOL` Name of the SYNTHES output field.

`P3 NRADS`	Number of radar input volumes included in the synthesis.
`P4 DTEST1`	User specified constraint on the component contribution of W to (u, v) described by Equation 2.22. ϵ_u and ϵ_v must both be less than DTEST1 for an acceptable two equation solution.
`P5 DTEST2`	User specified constraint on the quality of the (u, v) solution. Both $\sigma_N u$ and $\sigma_N u$ from Equation 2.26 must be less than DTEST2 for an acceptable two or three equation solution.
`P6 DTEST3`	User specified constraint on the quality of the W solution. $\sigma_N W$ from Equation 2.26 must be less than DTEST3 for an acceptable three equation solution
`P7 ITWOEQ`	**YES**: Always use two equation solution. **NO**: Use two equation solution when there is only data from two radars, otherwise use three equation solution.
`P8 ITANAL`	Reference (synthesis) time for advection correction. Advection correction is optional and can be skipped by leaving P8 through P10 blank.
`P9 STMDIR`	Storm direction for advection correction, in degrees clockwise from North.
`P10 STMSPD`	Storm speed for advection correction, in meters per second.

The other parameters are additional card images specifying the import of additional data fields. For their detailed description, please consult the `cedric` manual. The `SYNTHES` operation produces several fields depending on whether two- or three-equation solution is selected:

U	Component of air motion in x direction, can be u in Equation 2.19 or u' in Equation 2.22.
V	Component of air motion in y direction, can be v in Equation 2.19 or v' in Equation 2.22.
W	Component of air motion in z direction, W in Equation 2.19.
USTD	Normalized standard deviation, square root of normalized u-variance $\sigma_N^2(u)$ in Equation 2.26.
VSTD	Normalized standard deviation, square root of normalized v-variance $\sigma_N^2(v)$ in Equation 2.26.

WSTD	Normalized standard deviation, square root of normalized W-variance $\sigma_N^2(W)$ in Equation 2.26, only available when three-equation solution is selected.
EWU	Geometric factor ϵ_u that scales W as a contribution to u in Equation 2.22.
EWV	Geometric factor ϵ_v that scales W as a contribution to v in Equation 2.22.

For the two-equation solution, if EWU and EWV are both less than DTEST1, and USTD and VSTD are both less than DTEST2, then the output U and V are given by u' and v in Equation 2.22, respectively. For the three-equation solution, if USTD and VSTD are both less than DTEST2, then the output U and V are given by u and v in Equation 2.19, respectively. In addition, if WSTD is less than DTEST3, the output W is given by W in Equation 2.19. If only two radial velocities are present at a grid point, the procedure defaults to two-equation solution.

3.5.4 Advection

The R-MDRS incorporates two stages of storm advection correction. The first stage accounts for non-simultaneous observations inherent in the sweeping motion of radar scans and across radar networks. This correction is performed during the synthesis process by the `SYNTHES` operation outlined previously. Mean motion of the storm over the system network heartbeat results in an increasing spatial displacement of meteorological values as sampling time progresses. A fast-moving storm within IP1's 1 minute heartbeat scanning may exhibit significant displacement. Thus to attain the desired 'snapshot' at the specified synthesis time, these displacements must be corrected. To reiterate Equation 2.13, a storm system moving with components (U, V) should be adjusted by the following advection correction:

$$\frac{[v_r r]_{t+\Delta t}}{[r]_t} = \left[\frac{u(x - x_0 + U\Delta t)}{r_t} + \frac{v(y - y_0 + V\Delta t)}{r_t} + \frac{W(z - z_0)}{r_t} \right]_t \tag{3.5}$$

Radial velocities are first multiplied by slant ranges from the radar at sample time $t+\delta t$. This product field $[v_r r]_{t+\delta t}$ is then shifted at storm motion to new spatial location coordinates,

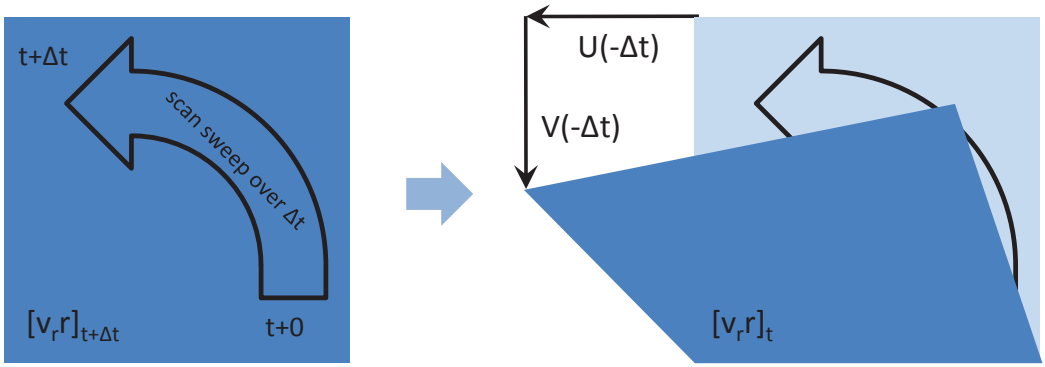


Figure 3.6: Sample time lag advection for storm moving at (U,V)

where it is divided by slant ranges at the synthesis time t. This spatial advection to correct for sample time lag is illustrated by Figure 3.6. The illustration suggests that for a storm moving at (U, V), the radial velocity v_r at (x, y) sampled at time $t + \Delta t$ is actually the v_r at $(x - U\Delta t, y - V\Delta t)$ at time t. Thus to adjust v_r to synthesis time t, the data field must be spatially advected by $(-U\Delta t, -V\Delta t)$.

Adjusting the velocity-range product field $v_r r$ instead of simply v_r is due to v_r being a non-Cartesian field undergoing Cartesian transformation. The product accounts for change in radar pointing direction during the advection, thereby improving the accuracy of the advection process. The detailed advection methodology is beyond the scope of this Thesis, but is detailed in Gal-Chen (1981).

The second stage of advection correction performed by the R-MDRS is the removal of mean storm motion altogether from the velocity products. This is done to isolate the internal air dynamics of the storm from the moving frame of reference of the storm itself. This can reveal features such as vortices that are otherwise obscured by the storm movement, and is a reflection on the fact that even if there is no spin in the absolute air motion relative to ground, an observer within the storm may still feel a spin. Mathematically, this correction removes the constant components from areas of convergence or divergence, leaving the pure derivative. Subtracting the mean storm motion is implemented by `cedric` with a relatively simple operation:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
FUNCTIONNAME          1.0                                          Z       FULL
        U       P       LINEAR  U                    1.0ADVX
        V       P       LINEAR  V                    1.0ADVY
END
```

The critical factor in both stages of advection correction is knowing the storm motion. Currently the R-MDRS obtains this from sounding data archived at University of Wyoming. Sounding data from the nearest National Weather Service WSR-88D radar at Norman, OK (KOUN) is gathered every 12 hours. From this data, air motion at mid-level atmosphere altitude corresponding to 700 mbar is used for mean storm motion. Being a single-point constant not centrally-located in the IP1 test bed and with a long sampling time, this value is inherently limited at accurately describing the mean storm motion. Generally speaking, larger, slower-moving, and longer-lasting storm systems tend to be more accurately portrayed by the sounding data.

For real-time operations, a lack of better alternatives has made sounding data the only solution for calculating mean storm motion. Post-analysis can of course use abundant posteriori information to accurately calculate the mean storm motion. A viable ongoing improvement of the RTMDS system is to use nowcasting products to determine mean storm motion accurately in real-time. Nowcasting is another development product of the IP1 project focusing on very short term forecast of weather events.

3.5.5 Field import and merging

The optional feature of `SYNTHES` to import fields into the `cedric` editing space is convenient for incorporating additional desired data fields from each of the radars. Further analysis and manipulations can be done with the data using the vast array of `cedric` operations. Merging reflectivity fields from the radars is one common application, as Doppler wind

products are almost always superimposed on top of another scalar data such as reflectivity to visualize the physical correlation between hydrometeors and air motion.

The R-MDRS is configurable to import any data field. However, conventionally it only imports reflectivity to save on processing time. From the very beginning, reflectivity is processed alongside radial velocity starting with the `nc2uf` data format conversion, through `reorder` interpolation, and finally to the current `SYNTHES` operation in `cedric`. Once imported, the following merging operation is performed within `cedric`:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
FUNCTIONNAME          1.0                                                  FULL
        BADMSK  T       CON                       -200.0
        CZ1     P       ORELSE  CZ1     BADMSK
        CZ2     P       ORELSE  CZ2     BADMSK
        CZ3     P       ORELSE  CZ3     BADMSK
        CZ4     P       ORELSE  CZ4     BADMSK
        CZ1     P       MAX     CZ1     CZ2
        CZ1     P       MAX     CZ1     CZ3
        CZ1     P       MAX     CZ1     CZ4
        CZ1     P       ONLYIFC<CZ1     CZ1       -200.0
END
DELETE  CZ2     CZ3     CZ4
```

The Corrected Reflectivity "CZ" fields from each radar refers to reflectivities that have been corrected for attenuation and other quality controls. These "CZ" fields first have their bad values masked with a known constant using the `ORELSE` function and a constant `BADMASK` field. They are then compared and merged with each other using the `MAX` function, which

eponymously merges using a maximization criteria. Finally the BADMASK is filtered back out, leaving blank elements to correspond to bad values.

More complex merging algorithms are avoided in favor of a simple maximizing scheme. This is to optimize speed and real-time performance of the R-MDRS. The merged reflectivity values are intended to merely provide a visual correlation background to the wind products, they are not intended to be high quality products suitable for rigorous analysis.

3.5.6 Reflectivity thresholding

After synthesis and data import, various data quality measures are taken by the R-MDRS before further processing. The first step usually entails thresholding reflectivity to realistic physical bounds between 15.0 to 80.0 dBZ using the operations outlined below:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
FUNCTIONNAME          1.0                                          Z        FULL
        CZ1      P        ONLYIFC<CZ1      CZ1          15.0
        CZ1      P        ONLYIFC>CZ1      CZ1          80.0
        U        P        ONLYIFC<U        CZ1          15.0
        V        P        ONLYIFC>V        CZ1          80.0
END
```

Velocity products are also thresholded on the same reflectivity thresholds to eliminate them from places where no significant hydrometeors exist. This operation is based on the assumption that without significant reflective particles, velocity products could not have been measured. Whatever velocity products that do exist in non-reflective areas are likely anomalies caused by noise or aliasing.

3.5.7 Beam-crossing angles

The issue of beam-crossing angle mentioned in Chapter 2 is the next problem resolved by `cedric`. To reiterate, beam-crossing angle refers to the angle of intersection between beams of two radars. A small beam-crossing angle corresponds to two radars scanning close along the axis connecting them. The two radars would measure radial velocities that are nearly equal and opposite, yielding a very small orthogonal component that results in great velocity error variances. The error variances corresponding to certain beam-crossing angles are known as the angle constant. Angle constants for some common minimum beam-crossing angles are:

Min beam-crossing angle	**Angle constant**
20.0 deg	4.135
22.0 deg	3.75
30.0 deg	2.83

Eliminating high error wind products within the minimum beam-crossing angle is then achieved by thresholding velocity variance against the angle constant, as outlined below:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
FUNCTIONNAME        -1.0                                          Z       FULL
        TEMP1   T       SQ+SQ   USTD    VSTD
        TEMP2   T       SQRT    TEMP1
        U       P       ONLYIFC>U       TEMP2       2.83
        V       P       ONLYIFC>V       TEMP2       2.83
END
```

The above operations threshold velocity against a velocity variance of 2.83, corresponding to 30.0 degrees minimum beam-crossing angle. This is the typical value used by the R-MDRS. For cases with very limited data, this constraint can be loosened to include data within narrow beam-crossing angles.

3.5.8 Other quality control processes

A series of quality control processes have been implemented throughout the maturity of the `cedric` stage of the R-MDRS. They begin with noise reduction and de-spiking operations, followed by a crosscheck between U and V values, and concluding with a final smoothing by Leise filters.

The decimation and de-spiking operations are to eliminate noisy features within the wind products.

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
PATCHER UD      U       DECIGLO      5.0                        Z       FULL
PATCHER VD      V       DECIGLO      5.0                        Z       FULL
FIELDSETPRI     UD      VD      U       V
PATCHER UD      UD      DECILOC        2       4     7.5        Z       FULL
PATCHER VD      VD      DECILOC        2       4     7.5        Z       FULL
FIELDSETPRI     UD      VD      U       V
PATCHER UD      UD      FILLCON        3       4       7        Z       FULL
PATCHER VD      VD      FILLCON        3       4       7        Z       FULL
```

The above procedure first decimate spurious velocity data globally using the `DECIGLO` operaton to set all values outside 5.0 standard deviations from the global mean to `BAD`. Then local decimation is performed with the `DECILOC` operation. This is done by taking a mean of all local points within 2 grid points of current grid point. If current grid point deviates by more than 7.5 from local mean, or if there are less than 4 `GOOD` points in the vicinity, the value is set to `BAD`.

Lastly, `BAD` values are attempted to be filled using the `FILCON` operation. Going out one grid interval from a `BAD` grid point, if at least 4 quadrants surrounding the grid point has `GOOD`

values and there are at least 7 GOOD points in the vicinity, the BAD grid point is interpolated with a linear least-square fit solution. If conditions are not met, the program goes out one more grid interval and repeats the procedure until it has gone out 3 grid intervals.

The u v crosscheck procedure is to ensure that every grid point has a valid u and v value, or none at all. If u or v are BAD, then the whole grid point will be labeled BAD. The procedure is as follows:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
FUNCTIONNAME
        UD      P       ONLYIF  UD      VD
        VD      P       ONLYIF  VD      UD
END
```

As a final pass, wind and reflectivity products are smoothed by a 1-step Leise filter described as follows:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
FILTER  UDF     UD      LEI             1                       Z       FULL
FILTER  VDF     VD      LEI             1                       Z       FULL
FILTER  CZ1     CZ1     LEI             1                       Z       FULL
```

These three procedures have the common goal to improve the integrity of the wind products across the spatial domain. In particular, they focus on low-pass filtering to remove discontinuities and improve the derivative properties of the u and v wind products. This is to prepare them for the final mass continuity calculations to solve for W outlined in Equation 2.29.

3.5.9 Mass Continuity

The W solved by the SYNTHES operation is the solution from the least mean square error solution outlined in Equation 2.19 and 2.22. As aforementioned, this value usually has very high errors due to the small vertical component of measured radial velocities used to perform the Doppler calculations. Practical and reliable results lie in utilizing the mass continuity solution outlined in Equation 2.29. For the discrete grid space, finite difference form is utilized between the previous p and the current c levels as described by Equation 2.30, reiterated here:

$$(\rho w)_c = (\rho w)_p - \delta \Delta z \overline{\left[\rho\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right]}_{p,c} \tag{3.6}$$

where

$$\delta = \begin{cases} +1 & \text{for updward integration} \\ -1 & \text{for downward integration} \end{cases}$$

The divergence terms are first calculated using the following procedure:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
FUNCTIONNAME          1.0                                                  FULL
        DIVR    T       DDI+DDJ U       V
        CONV    P       LINEAR  DIVR                -1.0     0.0
END
```

They are then integrated following Equation 2.30 using:

```
COMMENT
P1......P2......P3......P4......P5......P6......P7......P8......P9......P10...
COMEND
INTEGR  Wup     CONV    U*0.1   FRACT       0.25
INTEGR  Wvar    CONV    V*0.1   FRACT       0.25FRACT       0.25D=ZMIN  D=ZMAX
```

Two W products are produced by the two integration operations, W_{up} and W_{var}. The first integration specifies a P4 parameter of `U*0.1`, which means an upward integration with $\rho = e^{-0.1*z}$. Its P5 and P6 parameters set the bottom boundary condition to be based on a fraction of the field being integrated, specifically $W_{bot} = \texttt{CONV}*0.25*\delta*\Delta z$. The P5 parameter may also set a constant or a field as the boundary values of W, with constant value 0 being a common choice.

The second integration performs a variable integration requiring each vertical column to have both a bottom and an upper boundary condition as specified by the P4 parameter `V*0.1`. All else being equal, this method reduces the accumulated errors exhibited by W_{up} as the integration climbs higher upward. However, if the scan does not top out the storm, the assumed top boundary condition of $W_{top} = \texttt{CONV} * 0.25 * \delta * \Delta z$ would be invalid, and the integration fails. In such a case where the storm is not topped out, W_{up} is the only viable solution despite its upward error accumulation.

Chapter 4

EXPERIMENTS AND RESULTS

The R-MDRS has been in successful operation for the past two years across the IP1 DCAS network. During this time, it has successfully captured high resolution wind kinematics of every major weather event within the IP1 test bed. Operationally, it has also exhibited excellent robustness by maintaining reliable and stable operation under real-time demands.

Figure 4.1 illustrates R-MDRS's deployment within the CASA framework. R-MDRS's main operational platform is the SOCC at the University of Oklahoma (OU). Additional SOCCs at Colorado State University (CSU) and the University of Massachusetts Amherst (UMass) run parallel instances of R-MDRS as development and test platforms. Since R-MDRS operates on a local level at each SOCC, new instances of it are readily generated with modest IT resources.

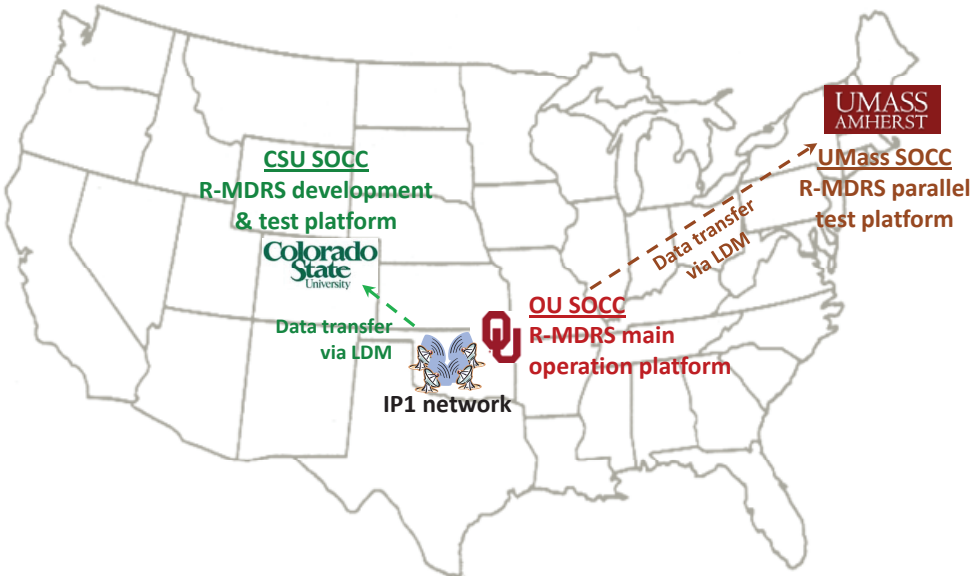


Figure 4.1: R-MDRS deployment across CASA

The following sections provide critical analysis of the R-MDRS by first examining some of its products from significant case events over the past two years. This will then be followed by a technical evaluation of R-MDRS's speed performance as well as various analysis focused on quality-effecting factors of the process. These include but are not limited to:

Advection Effects - Evaluate the quality of the advection corrections and its impact on the wind products. If the changes are insignificant, or if inadequate corrections are made based on unsound assumptions, it may be better to leave the data unadvected.

Vertical Product Comparisons - Compare the quality of W_{up}, and W_{var} vertical wind products. In conjunction with the physical circumstances under which the data is observed, a decision can be made over which is the better product.

4.1 Case Events

May 14, 2009. A cold front moved into the test bed from the north at approximately 0100 UTC and generated severe winds and at least one tornado that was captured by R-MDRS. Figure 4.2 is a display of the storm that was generated in real time. The domain stretches from -70 to 70 km in both X and Y directions, corresponding to the geographic layout of the IP1 test bed. During operation, this display would be updated every minute as each volume scan undergoes R-MDRS processing. It provides a real-time mesoscale indicator but cannot represent the data at full resolution, as that would create wind vector fields that are too dense to visualize. Instead, microscale features are fully resolved by zooming into specific regions on the order of 10 by 10 km. Figure 4.3 illustrates this with a full-resolution display of an EF2 tornado touchdown during the event.

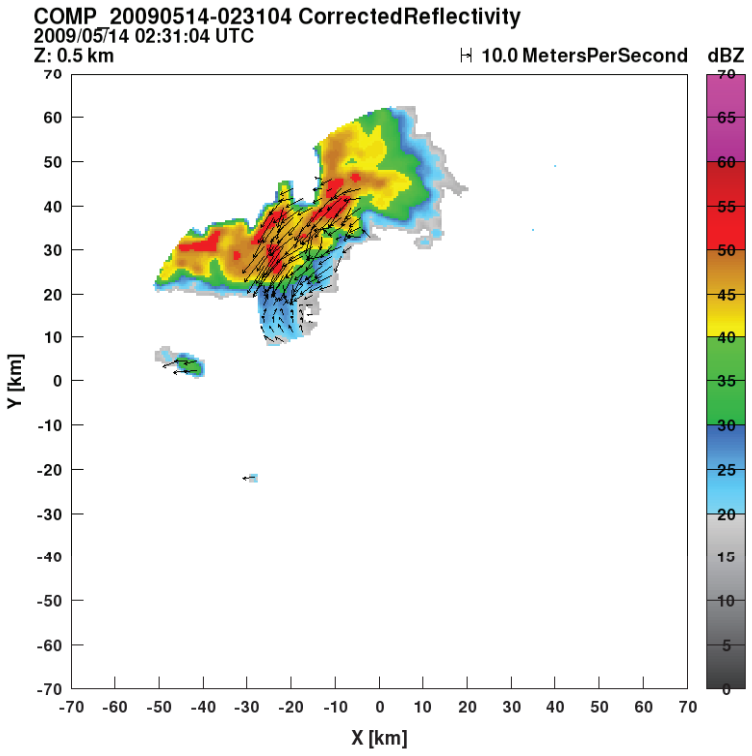


Figure 4.2: 2009-05-14 storm event at 02:31:04 UTC

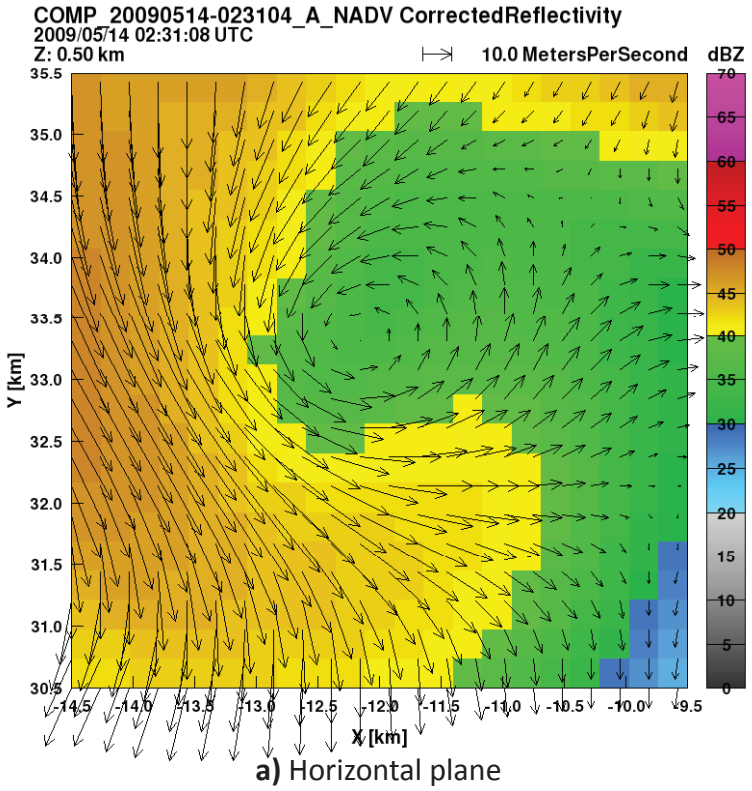


Figure 4.3: 2009-05-14 02:31:04 UTC - EF2 tornado touchdown

April 2, 2010. A severe squall line moving northeast entered the test bed at approximate 0830 UTC. The system produced heavy precipitation, hail, and minor wind damage. Figure 4.4 is the real-time mesoscale display that shows the most severe time frame at approximately 1100 UTC. Due to the extent of the squall line, most scans were very wide, typically more than 200 degrees. This translated to a more strenuous data load, which the R-MDRS handled without issue. Time performance was naturally more marginal but no crashes or time-outs occurred.

Numerous convective cells were exhibited all along the squall line. These produced strong updrafts at the front of the storm that were clearly visible in the vertical wind profile such as those illustrated in Figure 4.5.

Although no tornado touchdowns were reported for this event, strong circulations were detected by the R-MDRS as shown in Figures 4.6 and 4.7. Being recorded at 0.5 km altitude, it is likely that these vortexes made contact with the ground.

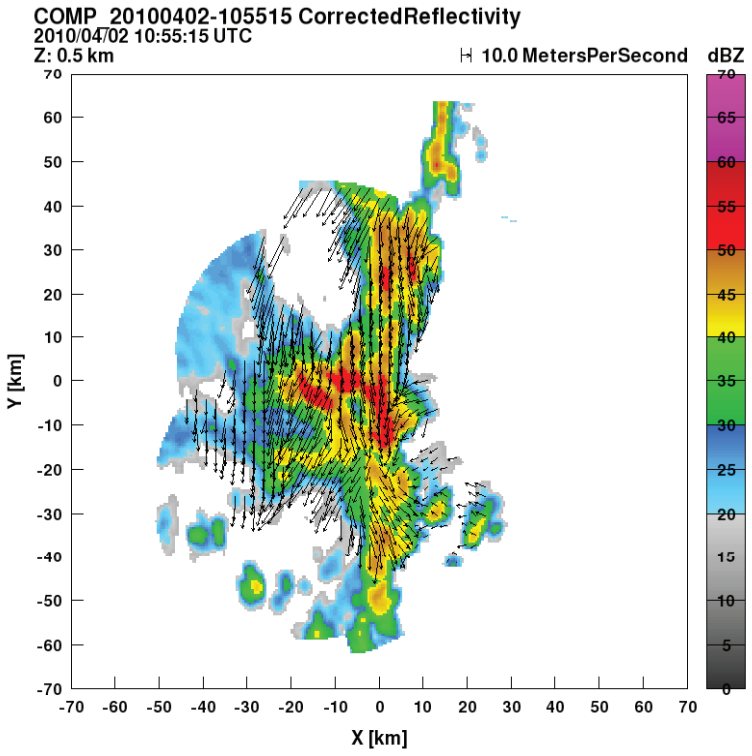
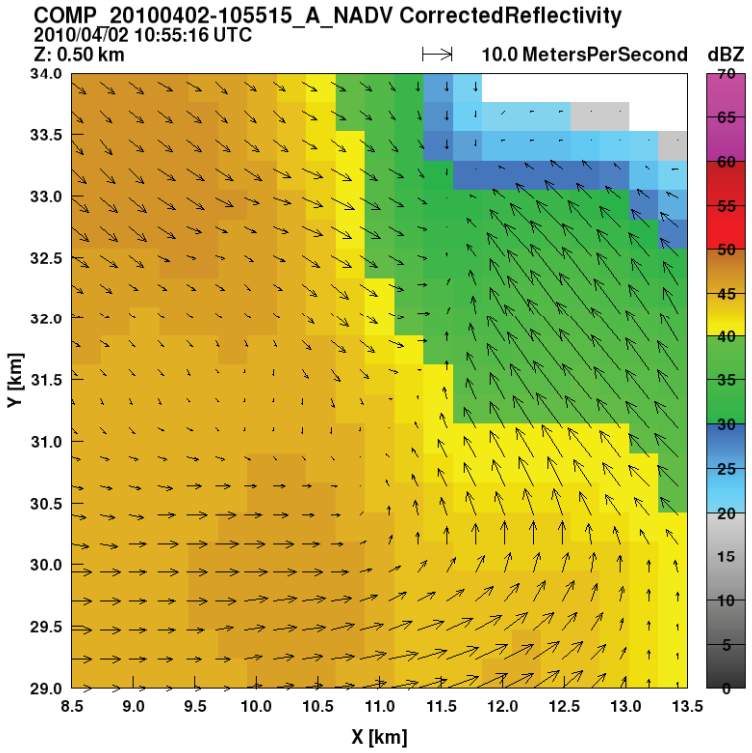
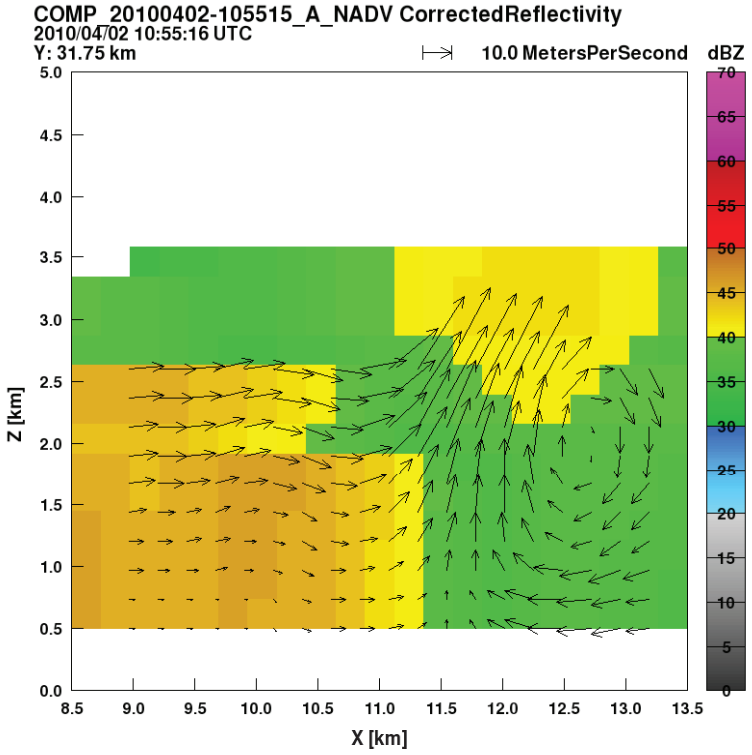


Figure 4.4: 2010-04-02 fast moving squall line at 10:55:15 UTC

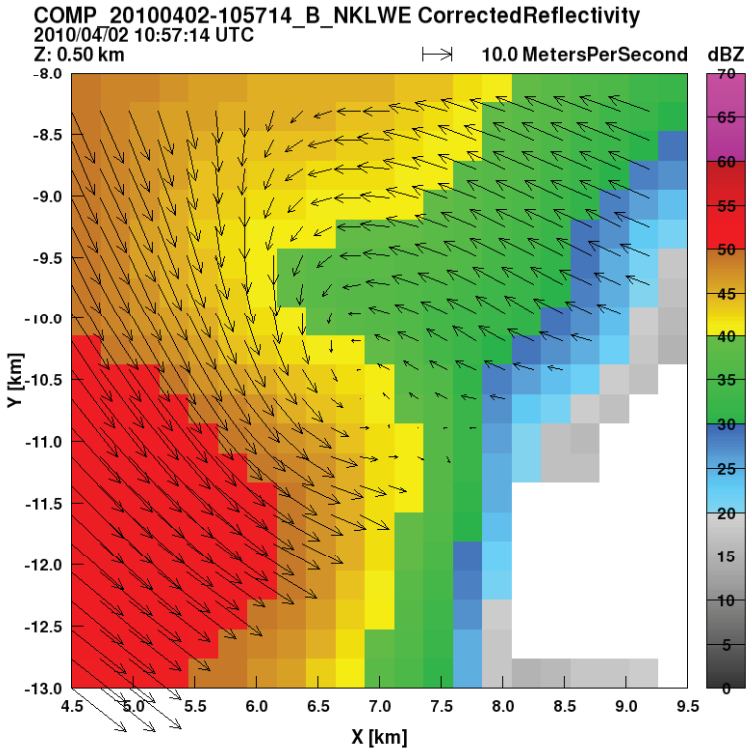


a) Horizontal plane

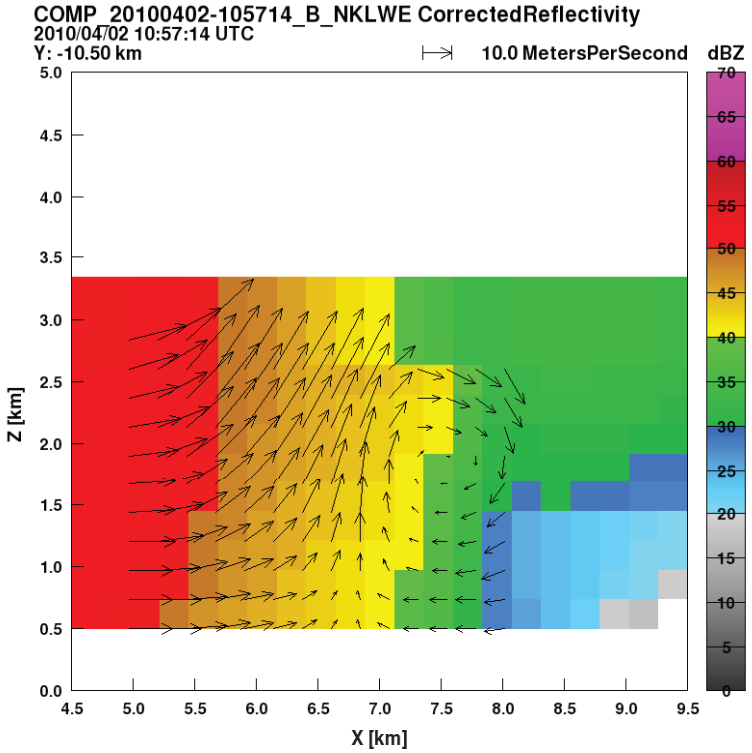


b) Vertical plane

Figure 4.5: 2010-04-02 10:55:15 UTC - squall front convection and updraft

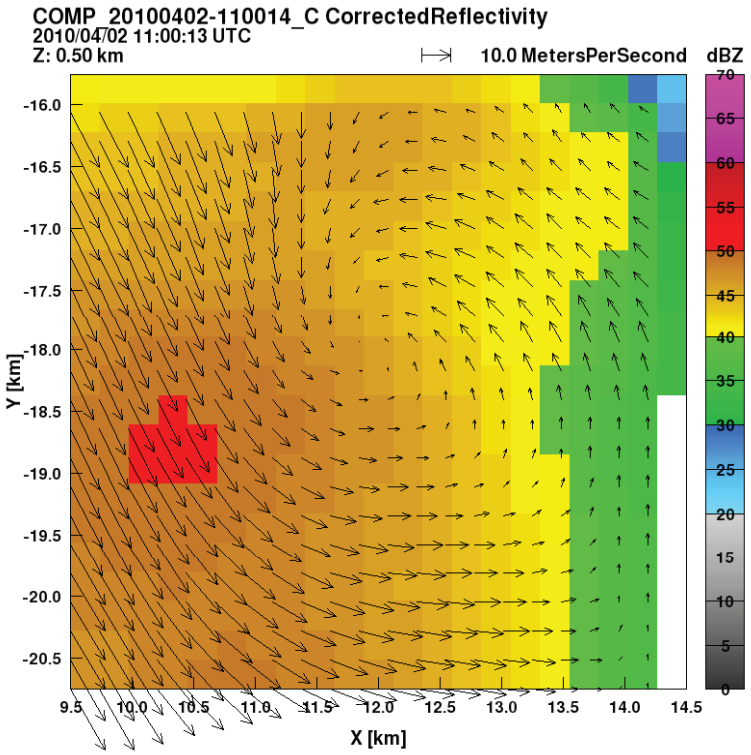


a) Horizontal plane

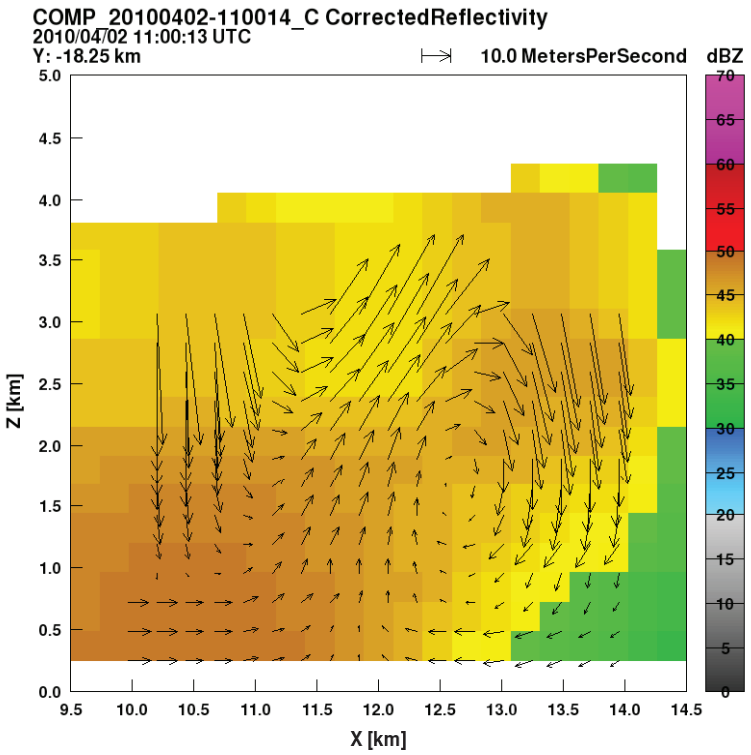


b) Vertical plane

Figure 4.6: 2010-04-02 10:57:14 UTC - circulation at edge of Doppler region



a) Horizontal plane



b) Vertical plane

Figure 4.7: 2010-04-02 11:00:14 UTC - circulation with strong convection

May 19, 2010. A string of tornado outbreaks occurred between 2100 UTC and 0200 the next day within a cluster of scattered thunderstorms. As seen from Figure 4.8, precipitation and other high reflectivity regions were relatively sparse. This limited velocity measurement regions due to reflectivity thresholding explained in Section 3.5.6. Consequently, R-MDRS was limited in its ability to capture the comprehensive air motion dynamics of the weather system.

17 tornados were reported for the event, with several originating within the test bed. However only one circulation occurred within Doppler regions, as illustrated in Figure 4.9. This circulation occurred in a region of the test bed where radar scans fully topped out the storm. This allowed variably-integrated vertical wind product W_{var} to be used to display the vertical wind profile.

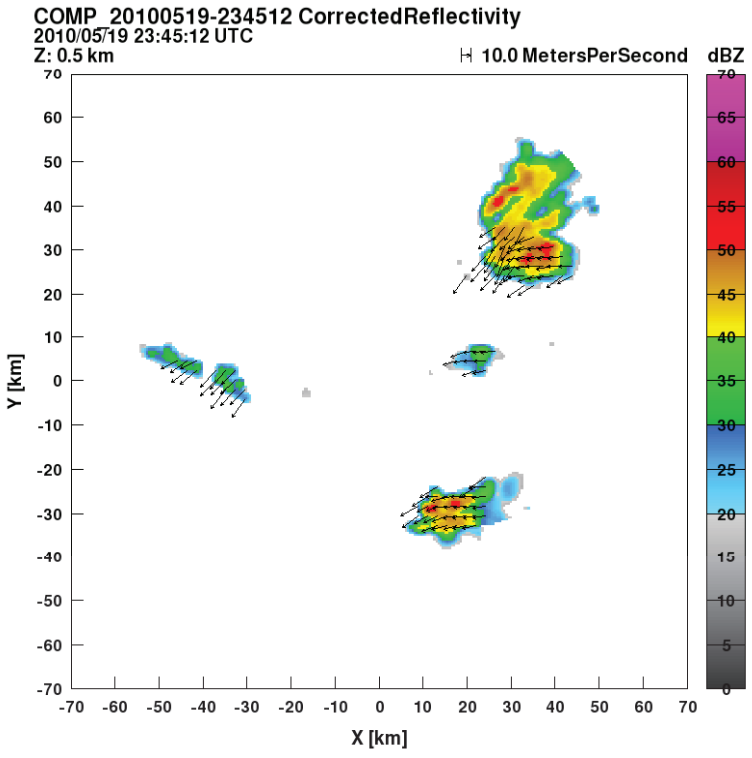
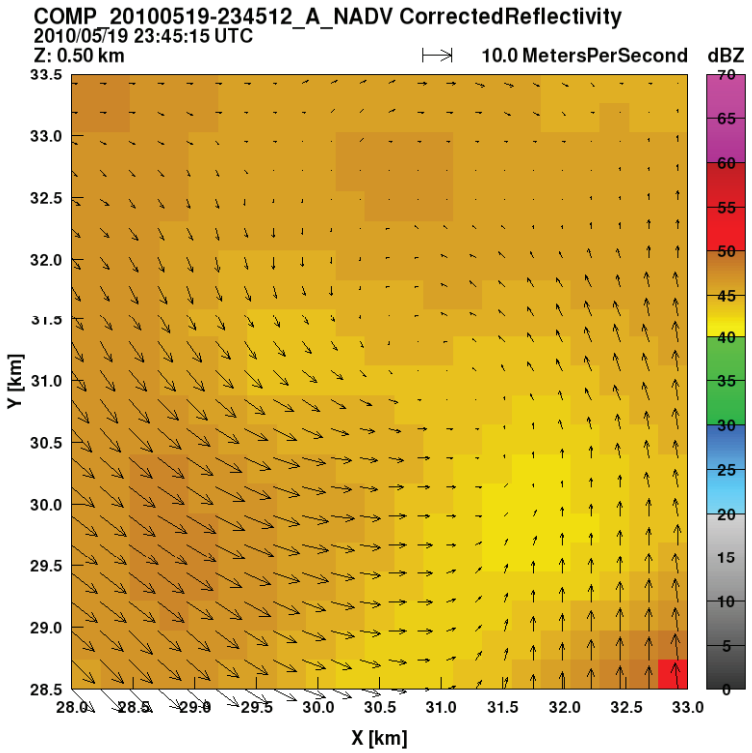
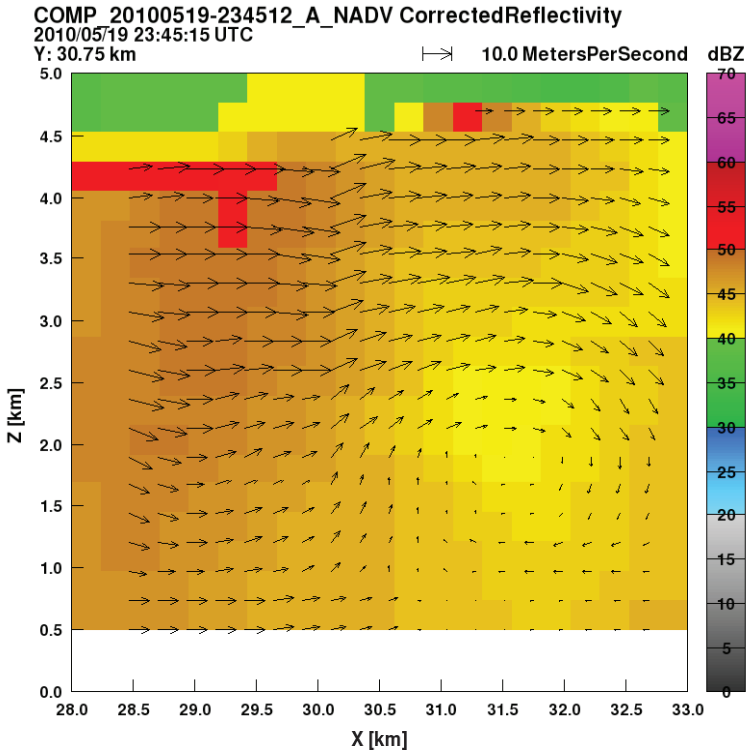


Figure 4.8: 2010-05-19 scattered thunderstorms at 23:45:12 UTC



a) Horizontal plane

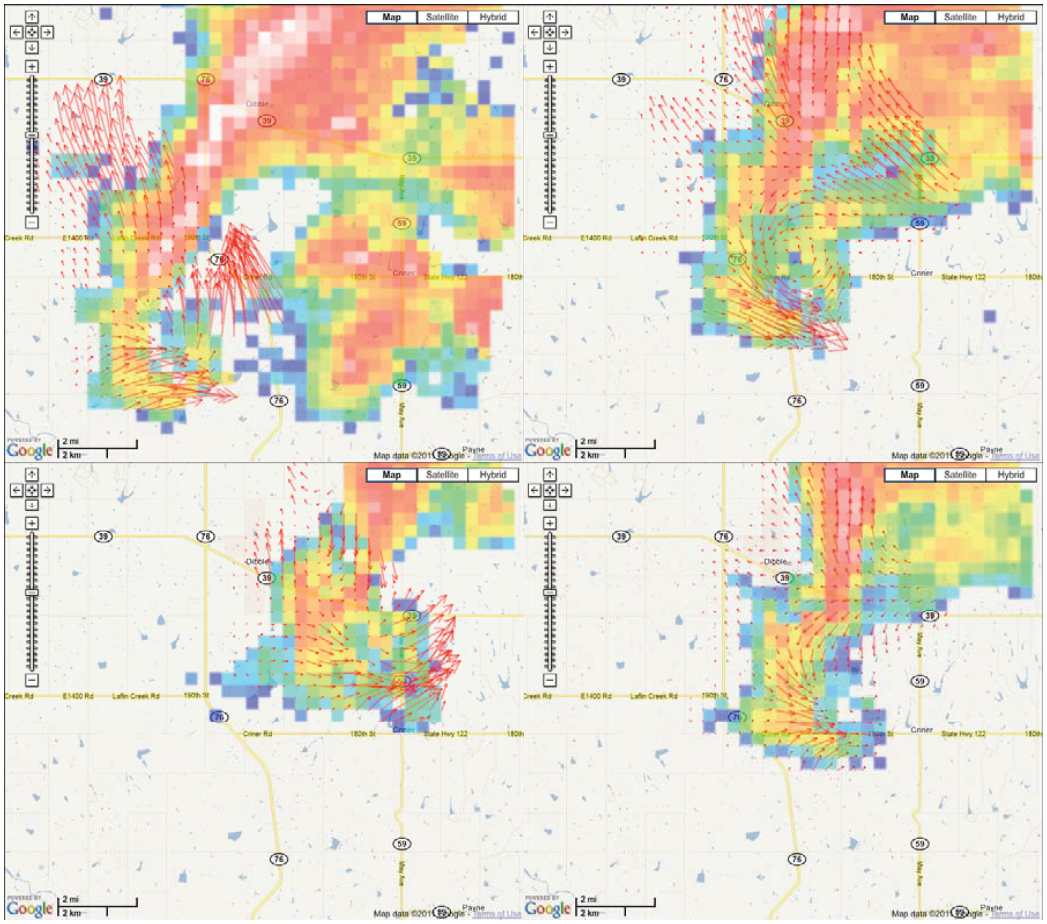


b) Vertical plane

Figure 4.9: 2010-05-19 23:45:12 UTC - circulation with W_{var} vertical wind product

May 24, 2011. A strong tornado touched down east of Chickasha near Criner, OK. A time-lapse of the tornado evolution is illustrated in Figure 4.10.

This event showcased a new R-MDRS real-time display that overlaid wind products onto Google Earth. This leap provides two feature advantages. The first is a geo-spatial reference for tracking the location and movement of storm features. The second is the integration with a mature commercial product such as Google Maps. This integration provides a streamlined web interface for end-users, forming the crucial practical link between CASA and society. Combined, both these factors have taken R-MDRS one step further to being a mature early weather warning system.



Clockwise from top left: Google Earth time lapse of tornado east of Chickasha near Criner, OK. May 24, 2011

Figure 4.10: 2011-05-24 - time-lapse of tornado touchdown near Criner, OK

4.2 Real-time Performance

The raison-d'etre of the R-MDRS is to perform IP1 Doppler retrievals in real-time. This section provides an analysis of R-MDRS's real-time performance collected over hundreds of executions. Further, a breakdown and comparison of R-MDRS individual process times will also be examined.

Time logging is collected from both R-MDRS's development test platform `rainier` at Colorado State University as well as the SOCC application server at the University of Massachusetts. At both locations, the R-MDRS is configured to process IP1 test bed as a 140 by 140 km grid extending from 0.5 to 10.0 km altitude with a grid resolution of 0.5 km. This configuration results in a total of 1,658,181 grid points in the composite radar product. It is chosen after calibrating `rainier` to be able to consistently complete a single R-MDRS synthesis within IP1's 1 minute scanning heartbeat. As UMass SOCC is a much more powerful computer, its time performance is expected to be better.

Figure 4.11 below illustrates the distribution of R-MDRS total process time over 689 executions on `rainier` during actual case events. As expected for a calibration reference, the majority of synthesis completed within a safe margin of 35 to 45 seconds. Less than 1% of cases extended beyond 1 minute, which in operational sense would have resulted in skipping data of the next heartbeat.

In more detail, Figure 4.12 below illustrates the time consumption of each major process of the R-MDRS as it executes on `rainier`. Interpolation by `reorder` occupies the majority of compute cycles, followed in descending order by `cedric` Doppler synthesis, miscellaneous tasks, and `nc2uf` format conversion. Miscellaneous tasks include but are not limited to: obtaining sounding data, conforming to WDSSII format, and clean up.

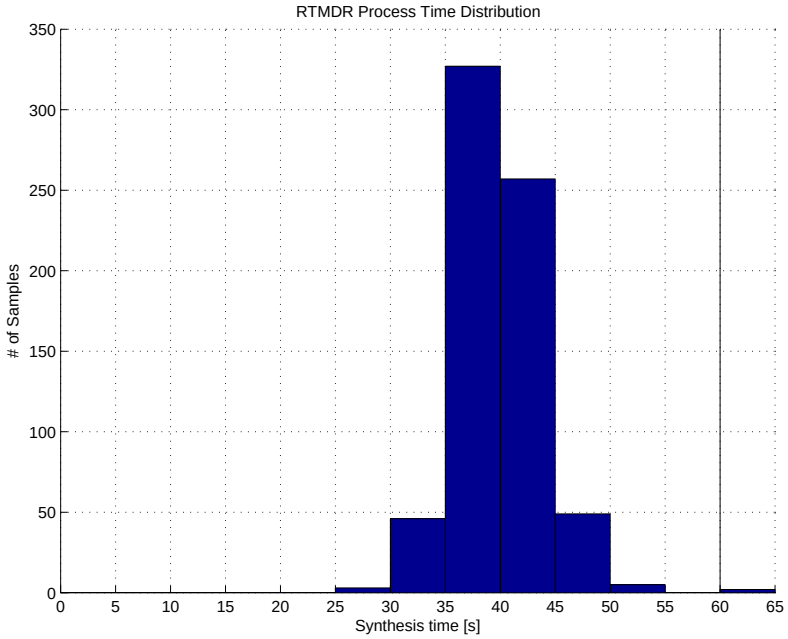


Figure 4.11: R-MDRS process time distribution on Rainier

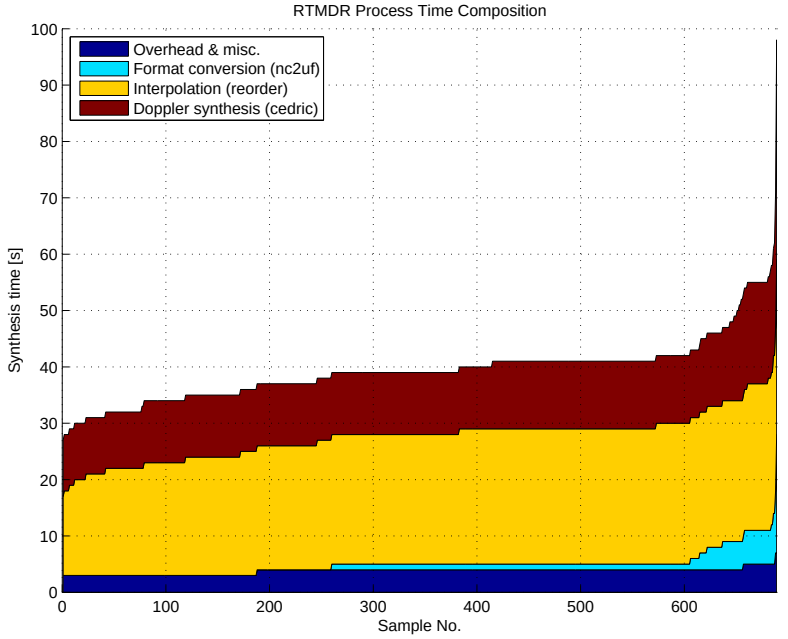


Figure 4.12: R-MDRS process time composition on Rainier

Figures 4.13 and 4.14 below illustrates expectedly better time performances for UMass SOCC server. With a sufficiently large sample of 1,090 R-MDRS executions, over 90% completed within 35 to 40 seconds. Similar to `rainier`, about 1-2% of cases extended beyond 1 minute, resulting in data loss.

While UMass SOCC is faster than `rainier` as expected, visually comparing the time performance of the two servers does point to some glaring disparities. Specifically, the two servers does not have the same time composition as one would expect from running identical software. Comparing Figures 4.12 and 4.14 reveals that all subroutines **except** `reorder` takes approximately one third of the time it does on UMass SOCC as it does on `rainier`. For `reorder` interpolation, UMass SOCC actually takes longer than `rainier` with an average of more than 30 seconds as compared to `rainier`'s consistent 20 seconds. The cause of this is still under investigation.

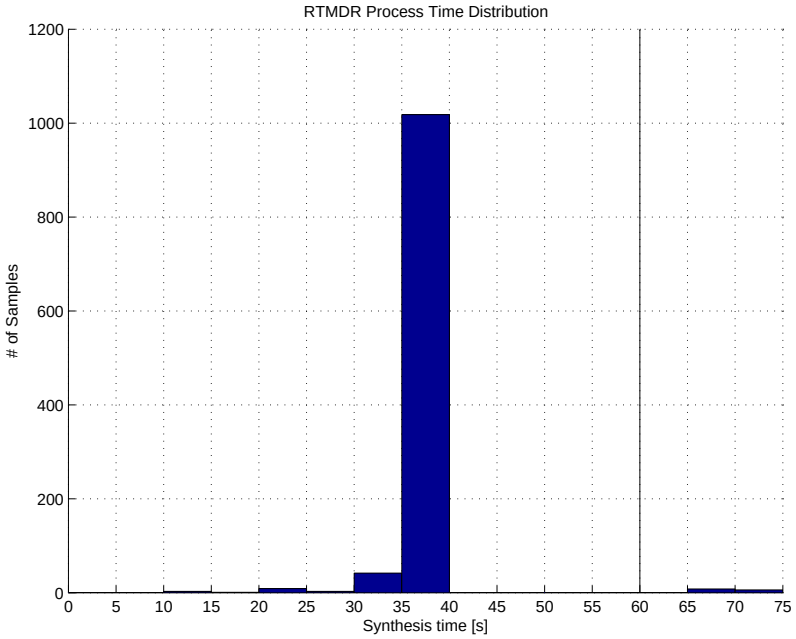


Figure 4.13: R-MDRS process time distribution on UMass SOCC

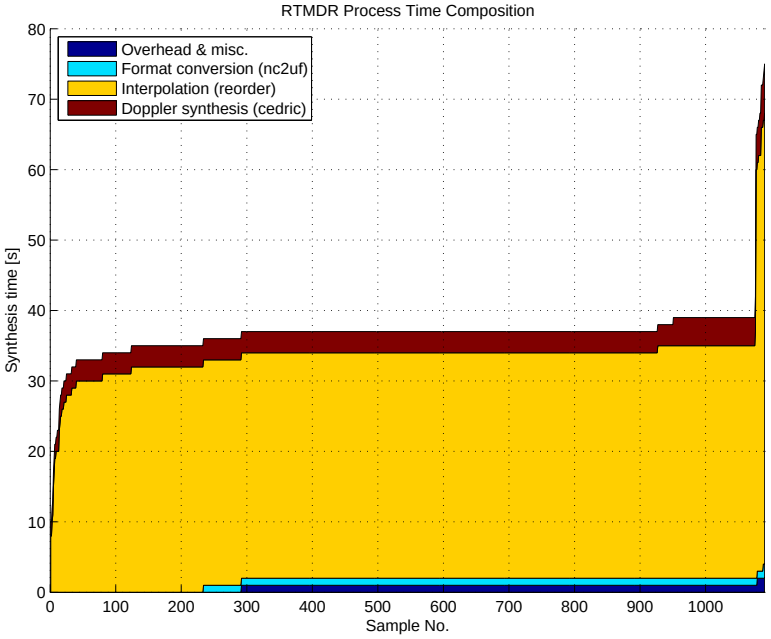


Figure 4.14: R-MDRS process time composition on UMass SOCC

Figure 4.15 illustrates the overall breakdown of R-MDRS process times into its subroutines. It combines both `rainier` and UMass SOCC time data to provide a complete process breakdown characteristic of the R-MDRS. By far the greatest demand on computing resources is the interpolation of each radar from radial to Cartesian coordinates, thus it is insightful to predict that the greatest performance gain is likely obtained by improving `reorder` executions. Chapter 5 will provide further theories on how this may be accomplished.

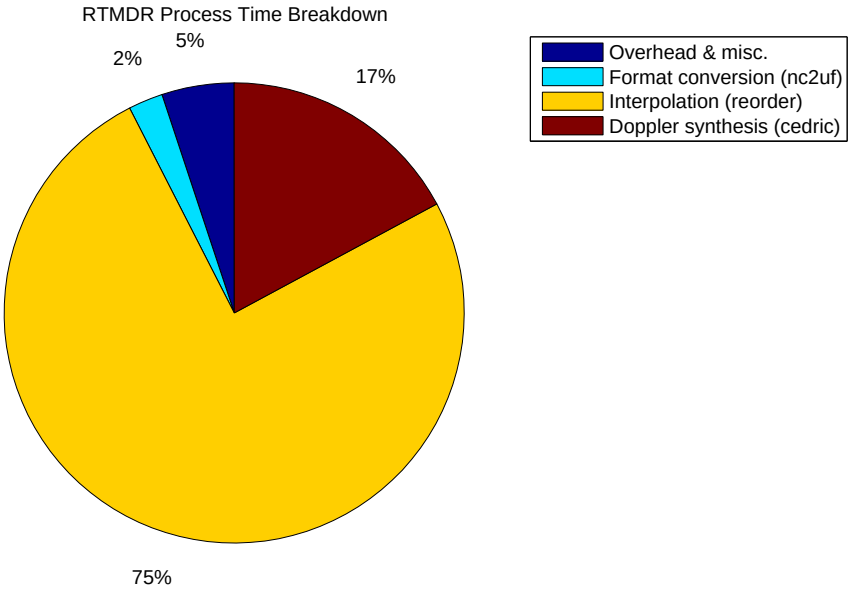


Figure 4.15: R-MDRS process time breakdown

4.3 Advection Effects

Advection correction during R-MDRS operation is detailed in Section 3.5.4. To reiterate, two stages of advection corrections are performed during real-time synthesis. The first stage accounts for displacement caused by sampling time lag. The second stage subtracts the mean storm motion altogether to isolate internal air dynamics from the moving frame of reference of the storm itself.

Currently, during real-time operations, the only means by which the R-MDRS can procure mean storm motion information is from sounding data measured by National Weather Service stations. The closest sounding station to the IP1 test bed is at the Norman WSR-88D radar (KOUN). Sounding data is gathered by balloons once every twelve hours. Several shortcomings immediately become apparent. First, the sampling time of twelve hours may be too sparse to describe entire storm events. Secondly, the sounding is about 100 km north east of the test bed and may be too far for the measurements to be applicable to the test bed. Lastly, sounding data provides a single point measurement, applying this to the entire span of the test bed may be an oversimplified assumption. Regardless, sounding data has proven to be reliable at modeling daily trends of mesoscale air motion in the atmosphere. And as of yet, sounding data is the only tool available to R-MDRS for real-time determination of mean storm motion.

R-MDRS's experimental results have been inconclusive about the benefits and disadvantages of advection correction. The only conclusive observation is that subtracting mean storm motion has much more impact on wind products than sampling time displacement corrections. Figure 4.16 illustrates an instance where advection correction significantly improves the wind products. With both stages of advection corrections applied, Figure 4.16 clearly reveals the air circulation about a developing hook echo. Figure 4.16b shows wind products with no sampling time displacement corrections, while Figure 4.16c shows the results with both advection correction stages ignored. In both cases, the spinning vortex is obscured.

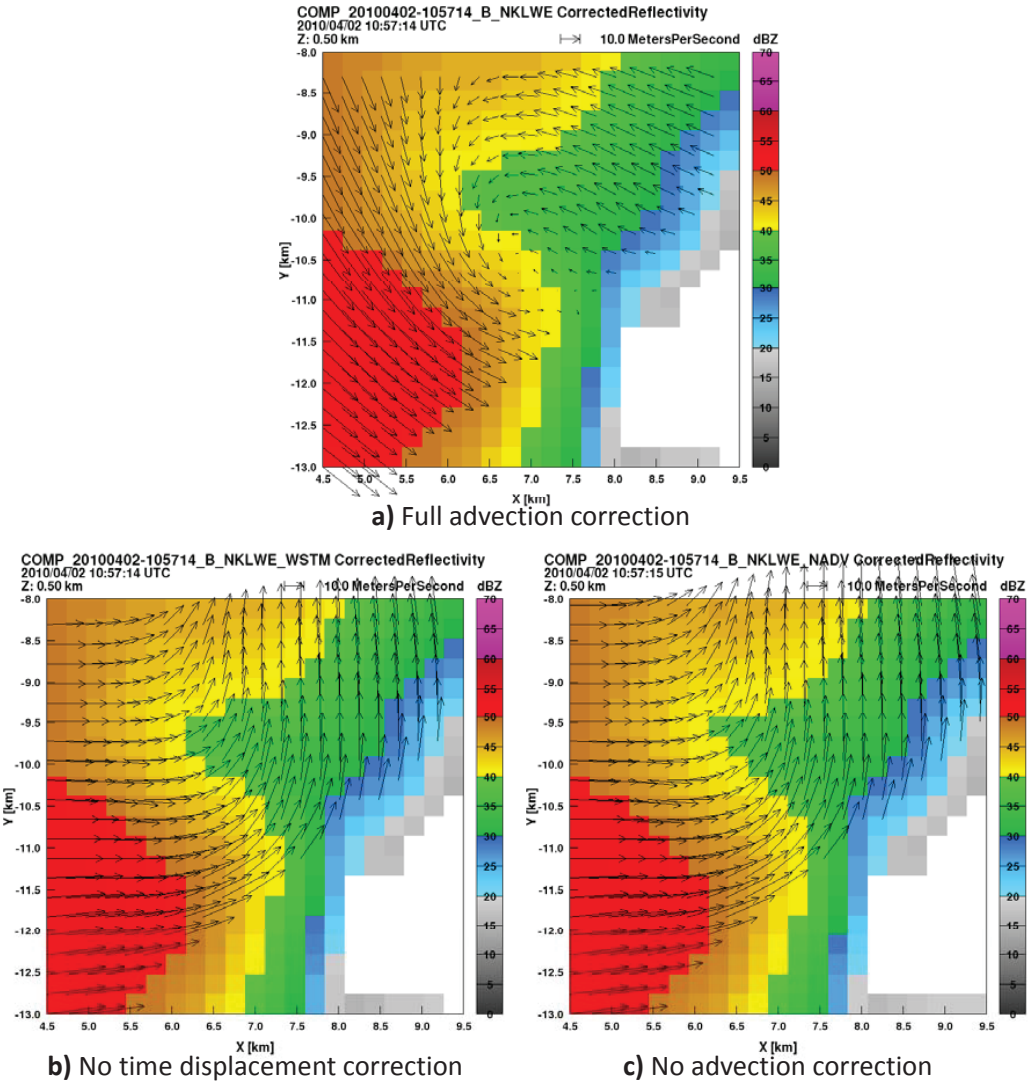


Figure 4.16: 2010-04-02 10:57:14 UTC - beneficial advection correction

It is worth noting, comparing Figures 4.16b with 4.16c reveals very minute differences. This supports the generally minute effects of sampling time displacement. Only for very fast-moving storms does the sampling time lag of radar scanning mechanics began to have noticeable effects on the Doppler wind products.

The two case examples aforementioned represents the general ambiguity of how beneficial advection correction is to the R-MDRS. It is reasonable to conclude that due to the nature of how sounding data is gathered, it is wholly inadequate for modeling real-time storm motions.

Consequently, without reliable storm motion information, no effective advection corrections can be made.

4.4 Vertical Wind Analysis

Several vertical wind products are generated according to the methodologies outlined in Section 2.5. To reiterate, they are:

W - Direct trigonometric result from system described by Equation 2.19. This product only exists for Doppler regions covered by three or more radars. Geometrically, since radars scan at low slant angles, very little vertical component exists. Consequently, W product is very error-prone and ignored.

W_{up} - Result of upward vertical integration as a part of the mass continuity solution approach outlined in Equation 2.30. Setting the bottom boundary of the storm as $W_{up} = 0$ simulates the physical boundary conditions of storm air motion. Integrating upward within limitations of mass continuity derives the rest of vertical wind products. This solution inevitably suffers from carry-over errors as altitude increases.

W_{var} - Similar to W_{up}, with the difference being that both bottom and top boundary of the storm are set as $W_{var} = 0$, and integration is performed in both directions. While this reduces carry-over errors as altitude increases, its basis assumption is only valid if Doppler region tops out the storm, which is rare during IP1 operations.

Figure 4.17 provides a comparison of the two products that is representative of the majority of R-MDRS syntheses. The vertical profile bisects the vortex of an air circulation near the front of the storm. The radars do not top out the storm, as can be seen in the sudden cutoff in reflectivity echoes. Due to not topping out the storm, W_{var} is not a viable product. This leaves W_{up} as the only valid product in the vast majority of R-MDRS synthesis.

Disregarding the validity of the products for one second and simply comparing Figure 4.17b and c confirms some of the characteristics of W_{up} and W_{var} outlined above. W_{up} exhibits increasing vertical magnitudes as altitude increases. While this may be true in some cases, its occurrence in nearly all syntheses indicates that it is a direct visual result of the

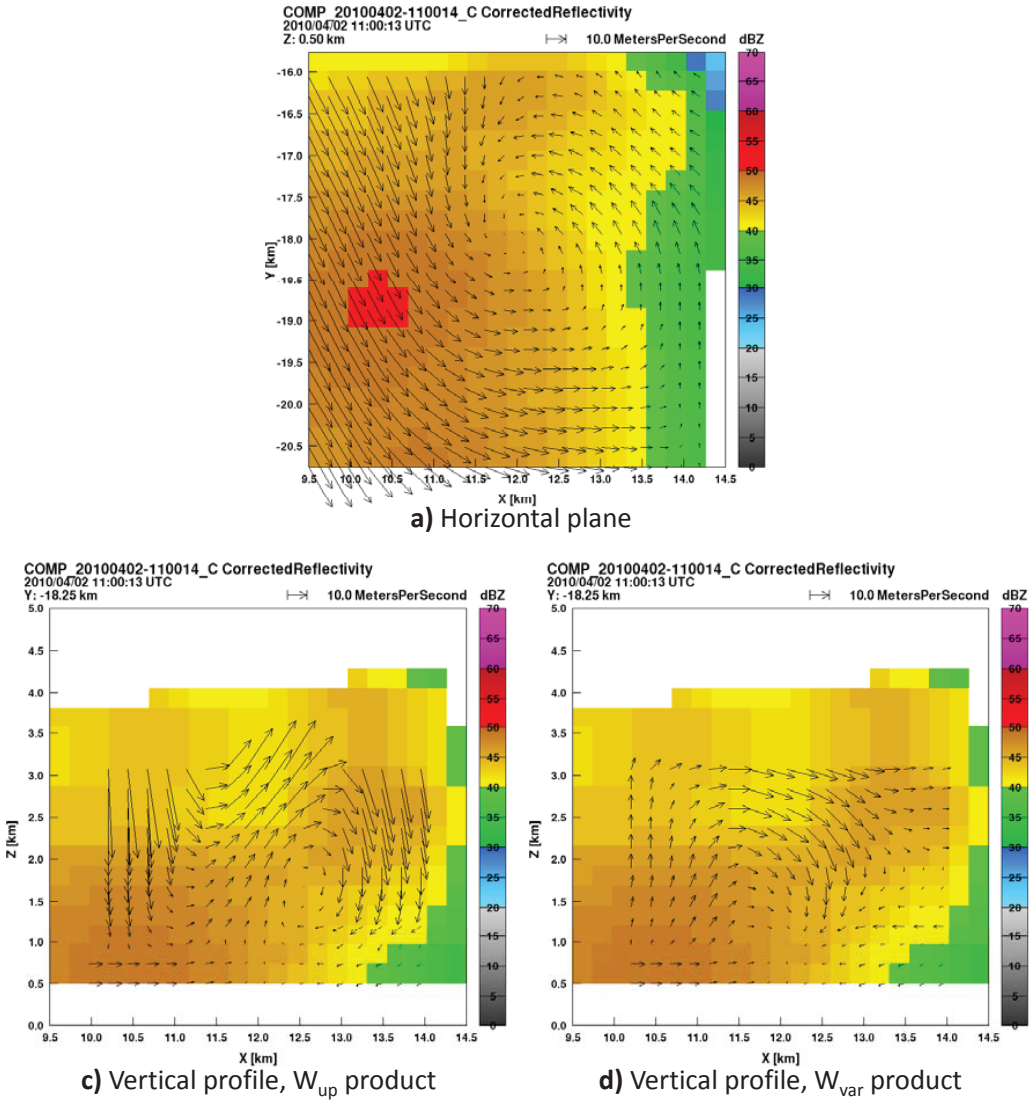


Figure 4.17: 2010-04-02 11:00:14 UTC - vertical wind products comparison

carry-over integration error. W_{var} has a much more controlled vertical variability as a result of its top and bottom bounding. However in this particular situation, its 0 vertical wind value in the middle of the storm is clearly nonsensical.

The primary reason the R-MDRS is limited to only using W_{up} product for the vast majority of syntheses is because of the low elevation scanning modes that IP1 radars typically operate in. This scanning mode makes sense for the operational goals of IP1, which is to detect acute low level weather features such as tornadoes. Scanning higher elevations will allow more prevalent overtopping of storms, which provides another known boundary condition that W_{var}

can utilize to reduce errors. In short, optimizing scans for better low level weather monitoring will naturally conflict with optimizing for better vertical wind profiling.

Chapter 5

SUMMARY AND CONCLUSION

5.1 Achievements

The most important practical achievement of this work is the successful implementation of a real-time multi-Doppler retrieval tool in CASA's DCAS network. This achievement is a culmination of overcoming a sequence of challenges.

The first challenge to be overcome is the integration of existing but fragmented data processing tools into a robust process flow that is fast and efficient enough to meet real-time demands. Much of R-MDRS's components such as `reorder` and `cedric` have existed for many years and been widely used. However they have never been linked and executed as a cohesive real-time unit. R-MDRS does that and customizes this integration into a specialized tool to meet CASA's needs.

This integration encompasses a variety of peripheral applications built around the few core components. They range from simple conversions to entire code libraries. On top of all these core components and their interlinks is a governing framework to dictate the process flow. Complex technical issues such as compatibility, file structures, function methods, etc all have to be carefully considered and resolved in order to create the conglomerate R-MDRS tool that robustly performs its designed functions.

Once the process flow of the R-MDRS is operational and products are being generated, the next challenge overcome by the R-MDRS is the comprehensive effort to eliminate errors, variations, and uncertainty to improve product integrity. This is done by adjusting processing parameters such as reflectivity-thresholding or beam-crossing angle criteria to produce variations in the final wind products and forming comparisons. This exploration of different combinations of settings continues until a high quality product is achieved.

This effort of perpetually tweaking R-MDRS settings to explore a great variety of results is possible due to the tool's superb useability. A comprehensive tool such as R-MDRS encounters the dilemma of creating a control interface that provides easy accessibility to all its features. Short of developing a full graphical user interface, R-MDRS overcomes this challenge through the implementation of a simple but effective text-based user configuration interface.

The final achievement of note is not the overcoming of any particular challenge, but rather a preemptive foresight that makes R-MDRS a promising tool to continue developing. From the beginning, R-MDRS has been designed with scalability and expansion in mind. This is in part due to knowing that version 1.0 of R-MDRS will not be perfect and will need significant adjustments. But more importantly it is the vision to expand the capabilities of R-MDRS beyond its core functions as a fast Doppler synthesizer.

The most important future-proofing element of R-MDRS is the script-based governing framework that dictates the process flow of all R-MDRS subcomponents. This avoids over-integration of processes and grant each of them autonomy and modularity to be modified or interchanged. As a further supplement, the PERL-based scripts and native UNIX core applications are portable across nearly all computing platforms. Last but not least, R-MDRS's development has been parallelled with detailed and concise documentations, making any future maintenance and expansion much more easier.

5.2 Suggestions for Future Works

As in the case of any version 1.0 products, R-MDRS still has much room to grow. This section will peruse some of the most relevant potentials for improvement.

While R-MDRS currently meets real-time demands for CASA IP1 network, there are still many significant advantages to be gained from increasing processing speeds even further. IP1 radars' native scanning resolution allow for much finer grid resolutions, down to approximately 100 meters. From a purely computational standpoint, this represents 125 times the current R-MDRS processing load, which is done at 0.5 km resolutions.

Timing performance analysis shows the greatest potential speed increase lies in the interpolation stage. When viewed in conjunction with the fact that R-MDRS interpolates each radar in sequence, it becomes immediately obvious that the interpolations need to be parallelized. This must be achieved via recoding the `reoced` PERL meta-script to initiate each radars' interpolations on separate processing threads. This is a simple procedure in itself, however complexities quickly arise when resolving computer resource conflicts with the rest of CASA applications running on the SOCC server. A more global thread management scheme will first need to be implemented on the SOCC level before stable parallelization can be implemented.

Another potential to increase speed is to analyze the data flow through the R-MDRS. From beginning to end, the radar data undergo several format conversions, which serve no benefit to the final wind products and are thus a waste of computing process. They are only necessary for adapting to proprietary formats of core applications such as `reorder` and `cedric`. The apparent solution here is to eliminate those core applications and re-code their functionalities in a completely NetCDF-native environment. This obviously embodies a tremendous amount of design and coding work, but the result would be a straightforward and efficient process that waste no overhead with conversions and translations. In fact, this improvement goes beyond mere speed increase, it would thoroughly reform the R-MDRS into

a fluent application suite built upon a unified framework consisting of NetCDF data structure, C applications and PERL control scripts.

Functionally, what the R-MDRS needs most is significant improvement to its advection correction methods. As results have shown, current utilization of sounding data gives no consistent outcome, and as such should be abandoned altogether. Another approach that shows great promise for solving this problem is short-term prediction of trending weather, also known as nowcasting. CASA has developed a functional nowcasting implementation detailed in Ruzanski (2009). As this nowcasting tool runs alongside the R-MDRS on the same SOCC hardware and DCAS operational framework, it is entirely plausible to integrate nowcasting results into the R-MDRS process to estimate real-time storm movement. This real-time value would be much more suitable for reliable advection corrections.

Appendix A

REOCED MANUAL

Manual for reoced

A real-time multi-Doppler retrieval system

==========

Sean X. Zhang

Colorado State University

sxzazn@engr.colostate.edu

Last updated 2011 June 28

Introduction

==========

reoced is the real-time multi-Doppler retrieval system developed for CASA IP1 test bed. It is an integrated interface built around two core components: reorder, which interpolates radial coordinated radar data into Cartesian; and cedric, which synthesizes the multi-Doppler wind products. Both programs are developed by the National Center for Atmospheric Research at Boulder, CO. Their detailed descriptions can be found at:

Oye, D., and M. Case, 1995: REORDER a program for gridding radar data, installation and use manual for the Unix version". National Center for Atmospheric Research, Atmospheric Technology Division.

```
  Miller, L.J., and S.M. Frerick, 1998: CEDRIC custom editing and display
    of reduced information in Cartesian space. National Center for Atmospheric
    Research, Mesoscale and Microscale Meterology Division.

reoced is solely developed by Sean X. Zhang (2009) at Colorado State
University.



Installation
=============
1) Dependencies - reoced requires the following software to be installed.
  Unidata NetCDF http://www.unidata.ucar.edu/software/netcdf/
  Universal Format provided by "ufsxz".
  reorder provided by NCAR
  cedric provided by NCAR

2) reoced.pl is a PERL script that is portable to all PERL-compatible computing
   platforms. Simply execute from shell command line.



Usage
======
reoced.pl VOLLISTPATH CFGPATH OUTPATH
  VOLLISTPATH: volume list file path.
  CFGPATH: config file path.
  OUTPATH: output NetCDF file path.
```

```
run_reoced_event.pl VOLLISTDIR CFGPATH OUTDIR

  VOLLISTDIR: volume list file directory.

  CFGPATH: config file path.

  OUTDIR: output NetCDF file directory.



Configuration

==============


General parameters

-------------------

General parameters are delineated by the "[General]" tag.


advection_mode (manual|auto)

  Advection mode, manual uses advx and advy values below, auto uses sounding

  data.


advx (float)

  Advection in X


advy (float)

  Advection in y


sounding_dir (str)

  Directory of sounding files


qc (0|1)
```

Perform quality control (Not)

Variable list

Variable list is delineated by the "[Variables]" tag.

<var1>

<var2>

.

.

.

Grid definition

Grid definition parameters are delineated by the "[GridDefinition]" tag.

glat (float)

Grid origin latitude

glon (float)

Grid origin longitude

galt (float)

Grid origin altitude

xmin (float)

x minimum

```
xmax (float)

  x maximum


xres (float)

  x increment


ymin (float)

  y minimum


ymax (float)

  y maximum


yres (float)

  y increment


zmin (float)

  z minimum


zmax (float)

  z maximum


zres (float)

  z increment


xrad (float)

  Radius of influence x component
```

yrad (float)

  Radius of influence y component

zrad (float)

  Radius of influence z component

azrad (float)

  Radius of influence delta azimuth component

elrad (float)

  Radius of influence delta elevation component

rgrad (float)

  Radius of influence delta range component

weighting_function (CRESSMAN|EXPONENTIAL|UNIFORM|CLOSEST POINT)

  Weighting scheme (consult reorder manual)

weighting_value (float)

  Weighting value (for EXPONENTIAL, consult reorder manual)

Appendix B

VOLLISTGEN MANUAL

Manual for vollistgen

A volume list generator for events

==========

Sean X. Zhang

Colorado State University

sxzazn@engr.colostate.edu

Last updated 2011 June 28

Introduction

==========

vollistgen is a PERL script for generating volume lists for the reoced real-time Multi-Doppler retrieval system. It compiles data file paths through parsing UNIX shells, after which it sorts and prints the volume lists into ASCII texts.

Installation

==========

vollistgen.pl is a simple PERL script that is portable to all PERL-compatible computing platforms. It has no dependencies. Simply execute from shell command line.

Usage

======

vollistgen.pl EVENTDIR CFGPATH OUTDIR

  EVENTDIR: event data directory.

  CFGPATH: config file path.

  OUTDIR: output volume list directory.

Configuration

==============

Parameters

----

begin (YYYYMMDDhhmms)

  The event start time in YYYYMMDDhhmmss format

end (YYYYMMDDhhmms)

  The event end time in YYYYMMDDhhmmss format

threshold (int)

  The threshold time in whole seconds. Sites must begin their volume scans

  within threshold time of the eachother in order for those sites to be

  included in the volume

Appendix C

NC2UF MANUAL

Manual for nc2uf

A tool to convert NetCDF to Universal Format

==========

Sean X. Zhang

Colorado State University

sxzazn@engr.colostate.edu

Last updated 2011 June 28

Introduction

==========

nc2uf is a program to convert NetCDF files to Universal Format (UF) files. UF is a radar data format originally proposed and documented in:

Barnes, S.L., 1980: Report on a meeting to establish a common Doppler radar data exchange format. The Bulletin of the American Meteorological Society, 61.11, 1401-1404.

nc2uf utilizes the ufsxz UF library. Both nc2uf and ufsxz are solely developed by Sean X. Zhang (2009) at Colorado State University.

```
Installation

=============

1) Dependencies - nc2uf requires the following software to be installed.

  Unidata NetCDF http://www.unidata.ucar.edu/software/netcdf/

  Universal Format provided by "ufsxz".


2) Configure "Makefile"

  PREFIX - Home directory of installation. Files will be placed relative to it.

    PREFIX/bin - binary files

    PREFIX/include - header files

    PREFIX/lib - library files

  ex: PREFIX=/usr/local/


  NCINCDIR - Directory where NetCDF header files reside.

  ex: NCINCDIR=/usr/local/include/


  NCLIBDIR - Directory where NetCDF library files reside.

  ex: NCLIBDIR=/usr/local/lib/


  UFINCDIR - Directory where UF header files reside.

  ex: UFINCDIR=/usr/local/include/


  UFLIBDIR - Directory where UF library files reside.

  ex: UFLIBDIR=/usr/local/lib/


3) Run "make" to compile.
```

4) Run "make install" to install. Use "sudo" as needed.

5) Run "make clean" to clean out intermediate setup files.

6) Optionally copy nc2uf.pl to execution directory.

Note: When running "make", configuration settings can be included, this will override the Makefile settings.
ex: make install PREFIX=/usr/local/

Usage

======

```
nc2uf [-h] [-f CFGPATH] [-o OUTPATH] NC1PATH NC2PATH ...
  -h: prints this message.
  -f CFGPATH: config file path (default: ./nc2uf.cfg).
  -o OUTPATH: output UF file path (default: ./output.uf).
  NC1PATH NC2PATH ... : input NetCDF file paths.

nc2uf.pl [CFGPATH] [OUTPATH] NC1PATH NC2PATH ...
  CFGPATH: config file path (default: ./nc2uf.cfg).
  OUTPATH: output UF file path (default: ./output.uf).
  NC1PATH NC2PATH ... : input NetCDF file paths.
```

The radial structure of UF files requires that the input NetCDFs to nc2uf must also be radial. This means the NetCDF must have a radial dimension and the variables to be converted must also be dimensionally radial.

nc2uf.pl is simply a PERL wrapper for nc2uf.

Configuration

==========

nc2uf execution is driven by a configuration file. Configuration parameters are set with the "=" sign and mostly correspond to UF ray header information such as "radarname", "raytime", etc... Since there are many rays in an UF file, these parameters must be sufficiently defined to dictate the values of all the ray headers of the final output UF. To achieve this, parameter values may be set to the following forms:

number - Constant integer or float value. Consequently, the corresponding ray header field will be constant across all rays of the output UF.

ex: lat=37.2

The "lat" (latitude) ray header field across all rays of the output UF will be set to 37.2 deg.

string - String enclosed by "". Constant across all rays.

ex: projectname="CASA"

dim.DIMNAME - NetCDF dimension. Since NetCDF dimensions have no value, this is mostly used to provide indexing references such as determining which order to read out a multi-dimensioned NetCDF variable.

ex: raydim=dim.Radial

var.VARNAME - NetCDF variable. The variable must be dimensionally radial. The radial dimension is defined by the "raydim" parameter. The corresponding ray header field across all rays of the output UF will then be uniquely taken from the specified variable. If the variable contains other dimensions, the first value of each ray will be taken.

ex: raytime=var.Time

"Time" is a single dimension NetCDF variable (radial). The "raytime" UF ray header field across all rays of the output UF will be set to the corresponding "Time" variable values.

var.VARNAME.VATTNAME - NetCDF variable attribute. Constant across all rays.

ex: units=var.velocity.units

global.GATTNAME - NetCDF global attribute. Constant across all rays.

ex: radarname=global.RadarName

In addition to these , parameter values may also contain special operator symbols:

' inversion operator - Inverts the value or variable that precedes it.

ex: prt=var.Prf"

The "prt" ray header field for each ray will be set as the reciprocal of the corresponding "Prf" NetCDF variable value.

* multiply operator - Mutiplies by value.

ex: binspacing=var.GateWidth*0.001

The "binspacing" ray header field of each ray will be set as the corresponding "GateWidth" NetCDF variable value factored by 1000.

The following is an explanation of the nc2uf configuration parameters.

General parameters

General parameters are delineated by the "[General]" tag.

raydim (NetCDF dimension)

Radial dimension.

bindim (NetCDF dimension)

Gate (bin) dimension.

volid (int)

Volume identifier.

swpid (int)

Sweep identifier.

radarname (str)

Radar name.

sitename (str)

Site name.

lat (double)

  Latitude of radar in degrees.

lon (double)

  Longitude of radar in degrees.

height (float)

  altitude of radar in km.

raytime (int)

  Ray time in Unix time (seconds since Jan 1 1970 UTC).

azimuth (double)

  Ray azimuth in degrees.

elevation (double

  Ray elevation in degrees.

swpmode (int)

  Sweep mode (see "ufsxz" manual).

swprate (float)

  Sweep rate in deg/sec.

ncbadval (double)

  NetCDF bad value.

ufbadval (short)

  UF bad value.

projectname (str)

  Project name.

blineazimuth (double)

  Baseline azimuth.

blineelevation (double)

  Baseline elevation.

tapename (str)

  Tape name.

binflag (int)

  Range gate (bin) status flag (see "ufsxz" manual).

Variable parameters

Variable parameters are delineated by [] brackets containing the variable name

uffieldname (str)

  UF field name mnemonic (2 ASCII characters).

scale (int)

  Scaling factor (actual value=file value/scale).

startrange (float)

  Start range in km.

binspacing (float)

  Gate (bin) spacing in meters.

nbin (int)

  Number of gates (bins).

pulsewidth (float)

  Pulse width in meters.

beamwidthH (double)

  Horizontal beam width in degrees.

beamwidthV (double)

  Vertical beam width in degrees.

rcvrbandwidth (float)

  Receiver bandwidth in MHz.

polarization (int)

  Polarization status (see "ufsxz" manual).

wavelength (float)

  Wavelength in cm.

nsample (int)

  Number of samples used in field estimate.

threshfield (str)

  Threshold field name mnemonic (2 ASCII characters).

threshval (float)

  Threshold value.

prt (float)

  Pulse repetition time in microseconds

radarconstant (float)

  Radar constant.

noisepower (float)

  Noise power in dBm.

rcvrgain (float)

  Receiver gain in dB.

peakpower (float)

  Peak power in dBm.

antgain (float)

  Antenna gain in dB.

pulsedur (float)

  Pulse duration in microseconds.

nyquistv (float)

  Nyquist velocity in m/s.

veflag (str)

  Velocity status flag (see "ufsxz" manual).

Appendix D

UFSXZ UF LIBRARY USER GUIDE

Manual for ufsxz

A library for the manipulation of Universal Format files

==========

Sean X. Zhang

Colorado State University

sxzazn@engr.colostate.edu

Last updated 2011 June 28

Introduction

==========

ufsxz is a library for manipulating Universal Format (UF) files. UF is a radar data format originally proposed and documented in:

Barnes, S.L., 1980: Report on a meeting to establish a common Doppler radar data exchange format. The Bulletin of the American Meteorological Society, 61.11, 1401-1404.

ufsxz is an entirely standalone library soley developed by Sean X. Zhang (2009) at Colorado State University.

```
Installation
=============
1) Configure Makefile
  PREFIX - Home directory of installation. Files will be placed relative to it.
    PREFIX/bin - binary files
    PREFIX/include - header files
    PREFIX/lib - library files
  ex: PREFIX=/usr/local/

2) Run "make" to compile.

3) Run "make install" to install. Use "sudo" as needed.

4) Run "make clean" to clean out intermediate setup files.

Note: When running "make", configuration settings can be included, this will
override the Makefile settings.
ex: make install PREFIX=/usr/local/


UF file structure
==================
UF files consist of records that correspond to rays (data acquired for a given
pointing direction). Large rays may be broken into several records. The multiple
records within a ray will be identical except differing data header, field
headers, and field values. Note that some programs such as IRIS converters do
not support multi-record rays, and must place a ray in a single record.
```

In the case of ufsxz, rays may consist of a maximum of 100 fields with a maximum of 50 fields per record, resulting in a maximum of 2 records per ray. Each field can have a maximum length of 2500 bins (gates). These values can be altered at compile time.

Field names must be of two ASCII character and can be arbitrary, but it is recommended to conform to the following SIGMET standard field mnemonics:

DZ - Reflectivity factor (dBZ)

CZ - Corrected reflectivity factor (dBZ)

DR - Differential reflectivity, ZDR (dB)

PH - Differential phase, PhiDP (deg)

KD - Specific phase, KDP (deg/km)

RH - Cross-polar correlation, RhoHV (0 to 1)

DM - Received power (dBm)

NC - Normalized coherent power (0 to 1)

VR - Raw velocity (m/s)

VT, VE - Velocity thresholded on NC (m/s)

VF - Velocity with good/bad flag on least significant bit (m/s)

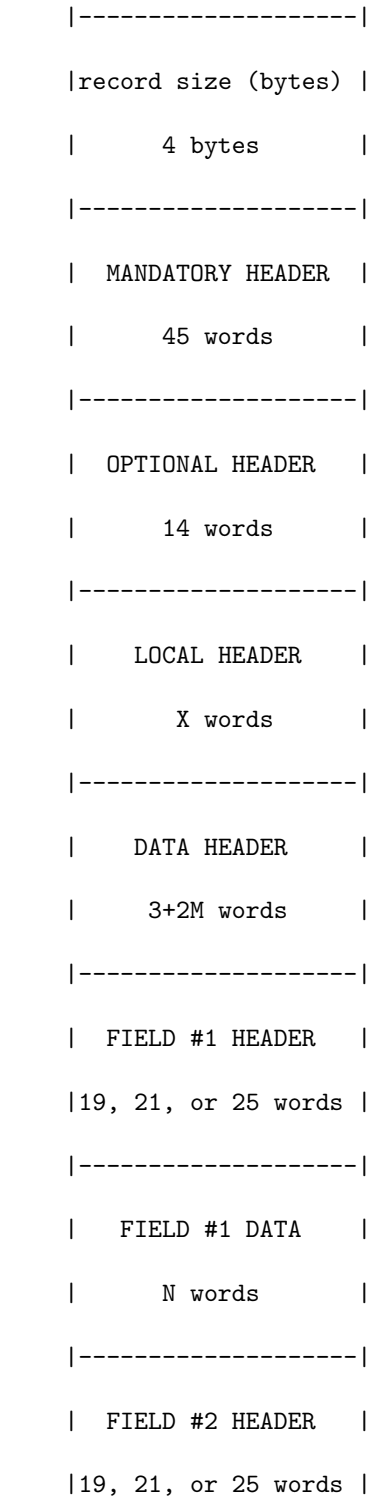
VP - Velocity thresholded on received power (m/s)

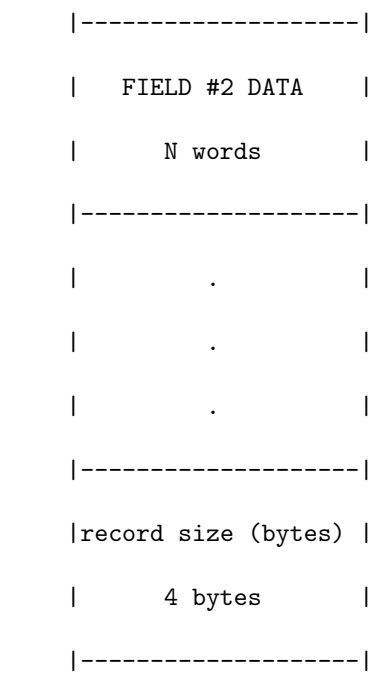
SW - Spectrum width (m/s)

SR - Raw spectrum width, no noise correlation (m/s)

Within each record, data is organized into two-byte aligned (16 bit word), byte-swapped (Big endian) format. The exception is that the record starts and ends with a 32 bit memory space indicating the number of bytes in the record, this value is still byte-swapped.

Record structure for a ray with M fields and N bins:





MANDATORY HEADER BLOCK

Word

1 "UF" (2 ASCII) *MUST NOT* have "UF" string anywhere else in UF file or read error will occur.

2 Record length (no. of 16-bit words)

3 Position of first word of optional header block (word position). If no optional header block exists, this points to the first existing header block following the mandatory header. In this way, word (3) always gives 1 + the length of the mandatory header (in words).

4 Position of first word of local header block (word position). If no local header block exists, this points to the start of the data header block.

5 Position of first word of data header block (word pos)

6 Record number within the file (starts from 1)

7 Volume number within the file (starts from 1)

```
    8   Ray number within the volume (starts from 1)

    9   Record number within the ray (starts from 1)

   10   Sweep number within the volume (starts from 1)

11-14   Radar name (8 ASCII)

15-18   Site name (8 ASCII)

   19   Antenna latitude - degrees (north is positive, south is negative)

   20   Antenna latitude - minutes

   21   Antenna latitude - seconds (s*64)

   22   Antenna longitude - degrees (east is positive, west is negative)

   23   Antenna longitude - minutes

   24   Antenna longitude - seconds (s*64)

   25   Antenna height above sea level (m)

   26   Ray acquisition time - year

   27   Ray acquisition time - month

   28   Ray acquisition time - day of month

   29   Ray acquisition time - hour

   30   Ray acquisition time - minute

   31   Ray acquisition time - second

   32   Ray acquisition time - time zone (2 ASCII - UT, CT, MT, etc...)

   33   Ray azimuth (deg*64)

   34   Ray elevation (deg*64)

   35   Sweep mode: 0 - Calibration

                    1 - PPI (constant elevation)

                    2 - Coplane

                    3 - RHI (constant azimuth)

                    4 - Vertical

                    5 - Target (stationary)
```

6 - Manual

7 - Idle

36 Fixed angle (deg*64, e.g. elevation of PPI, azimuth of RHI, etc.)

37 Sweep rate (deg/s*64)

38 UF generation date - year

39 UF generation date - month

40 UF generation date - day of month

41-44 UF generation program (8 ASCII)

45 Value stored for deleted, missing, or corrupted data

OPTIONAL HEADER BLOCK

Word

1-4 Project name (8 ASCII)

5 Baseline azimuth (deg*64)

6 Baseline elevation (deg*64)

7 Volume start time - hour

8 Volume start time - minutes

9 Volume start time - seconds

10-13 Tape name (8 ASCII)

14 Bin flag: number of range bins, minimum range, and range bin spacing are

0 - Same throughout volume

1 - Same only within each sweep

2 - Same only within each ray

LOCAL HEADER BLOCK

Any use, any contents

```
DATA HEADER BLOCK

  Word

    1   Total number of fields in the ray

    2   Total number of records in the ray

    3   Total number of fields in the record

    4   1st field name (2 ASCII)

    5   Position of first word of 1st field header

    6   2nd field name (2 ASCII)

    7   Position of first word of 2nd field header

   etc...


FIELD HEADER BLOCK

  Word

    1   Position of first data word of field

    2   Scale factor (meteorological value = file value / scale)

    3   Start range kilometers (km)

    4   Start range residue meters (m)

    5   Bin spacing (m)

    6   Number of bins

    7   Pulse width (meters)

    8   Horizontal beamwidth (deg*64)

    9   Vertical beamwidth (deg*64)

   10   Receiver bandwidth (MHz)

   11   Polarization: 0 - Horizontal

                      1 - Vertical

                      2 - Circular

                      3 - Elliptical
```

```
 12   Wavelength (cm*64)

 13   Number of samples used in field estimate

 14   Threshold field name (2 ASCII, SIGMET standard)

 15   Threshold value

 16   Scale

 17   Edit code (2 ASCII)

 18   Pulse repetition time (us)

 19   Bits per bin (must be 16)

20-?  Words for individual fields, as follows


for DM:

Word

 20   Radar constant (RC*scale such that dBZ = RC + DM + 20log(range km))

 21   Noise power (dBm*scale)

 22   Receiver gain (dB*scale)

 23   Peak power (dBm*scale)

 24   Antenna gain (dB*scale)

 25   Pulse duration (us*64)


for VR, VE, VT, VF, VP:

Word

 20   Nyquist velocity (m/s*scale)

 21   Velocity flag (2 ASCII): "FL" - Velocity flagged in least significant
                                      bit with good/bad flag (1=good,0=bad)
                               other - Velocity not flagged
```

ufsxz mechanics

==========

ufsxz manipulates UF files on a ray-by-ray basis corresponding to the radial structure of UF files. This means that ufsxz reads and writes UF files in ray chunks. ufsxz interfaces with UF rays via two important data structures and a data buffer. Reading and writing UF rays with ufsxz requires initializing these three components. See ufsxz.h for more details.

The first is the "UFHeaderInfo" data structure that contains general UF ray header info, such as ray identification, radar location, ray time, etc...

The second is the "UFFieldHeaderInfo" data structure that contains field-specific UF ray header info, such as UF field mnemonic, scaling factor, starting range, etc... Each field has its own "UFFieldHeaderInfo" data structure.

The third is the "raydata" data buffer that contains all the fields" data values of the current ray. "raydata" corresponds to UF data format by being big-Endian short integers, thus it can be directly read from or written to disk when manipulating UF rays.

UFHeaderInfo data structure

  int volnuminfile - volume number in file

  int raynuminvol - ray number in volume

  int swpnuminvol - sweep number in volume

  char radarname[8] - radar name

  char sitename[8] - site name

  double lat - latitude of ray source (antenna) in deg

```
  double lon - longitute of ray source (antenna) in deg
  float height - altitude of ray source (antenna) in meters
  int raytime - ray time in UNIX time
  double azimuth - ray azimuth in deg
  double elevation - ray elevation in deg
  int swpmode - sweep mode (see UF data structure)
  double fixangle - fixed angle (elevation for PPI, azimuth for RHI)
  float swprate - sweep rate in deg/s
  float badval - bad/missing value
  char projectname[8] - project name
  double blineazimuth - baseline azimuth in deg
  double blineelevation - baseline elevation in deg
  int voltime - volume time in UNIX time
  char tapename[8] - tape name
  int flag - range gate status flag (see UF data structure)
  int nfieldinray - number of fields in ray

UFFieldHeaderInfo
  char uffieldname[2] - UF field name (SIGMET standard)
  int scale - scale factor (actual value=file value/scale)
  float startrange - start range in km
  float binspacing - range gate spacing in km
  int nbin - number of gates
  float pulsewidth - pulse width in meters
  double beamwidthH - horizontal beamwidth in deg
  double beamwidthV - vertical beam width in deg
  float rcvrbandwidth - receiver bandwidth in MHz
```

```
  int polarization - polarization status id

  float wavelength - wavelength in cm

  int nsample - number of samples used in field estimate

  char threshfield[2] - threshold field name (SIGMET standard)

  float threshval - threshold value

  float prt - pulse repetition time in microseconds

  float radarconstant - radar constant

  float noisepower - noise power in dBm

  float rcvrgain - receiver gain in dB

  float peakpower - peak power in dBm

  float antgain - normalized antenna gain in dB

  float pulsedur - pulse duration in microseconds

  float nyquistv - nyquist velocity in m/s

  char veflag[2] - velocity status flag (see UF data structure)



Data mapping

=============

UF file (big endian)                    ufsxz structs (little endian)

-------------------------------------------------------------------------------

MANHEADER

uftag[2]="UF"                           -

reclen (no. of 16-bit words)            -

optheaderwpos (word pos)                -

locheaderwpos (word pos)                -

dataheaderwpos (word pos)               -

recnuminfile                            -
```

volnuminfile	UFHeaderInfo.volnuminfile
raynuminvol	UFHeaderInfo.raynuminvol
recnuminray	-
swpnuminvol	UFHeaderInfo.swpnuminvol
radarname[8]	UFHeaderInfo.radarname
sitename[8]	UFHeaderInfo.sitename
lat (deg,min,s*64)	UFHeaderInfo.lat (deg)
lon (deg,min,s*64)	UFHeaderInfo.lon (deg)
height (m)	UFHeaderInfo.height (m)
raytime (year,mon,mday,hr,min,s)	UFHeaderInfo.raytime (UNIX time)
azimuth (deg*64)	UFHeaderInfo.azimuth (deg)
elevation (deg*64)	UFHeaderInfo.elevation (deg)
swpmode	UFHeaderInfo.swpmode
fixangle (deg*64)	UFHeaderInfo.fixangle (deg)
swprate (deg/s*64)	UFHeaderInfo.swprate (deg/s)
gendate (yr,mon,mday)	-
progname[8]=ufsxz	-
badval	UFHeaderInfo.badval

OPTHEADER	
projectname[8]	UFHeaderInfo.projectname
blineazimuth (deg*64)	UFHeaderInfo.blineazimuth (deg)
blineelevation (deg*64)	UFHeaderInfo.blineelevation (deg)
voltime (hr,min,s)	UFHeaderInfo.voltime (UNIX time)
tapename[8]	UFHeaderInfo.tapename
binflag	UFHeaderInfo.binflag

DATAHEADER	
nfieldinray	UFHeaderInfo.nfieldinray
nrecinray	-
nfieldinrec	-
DATAHEADERFIELD	
uffieldname[2]	UFFieldHeaderInfo.uffieldname
fieldheaderwpos (word pos)	-
FIELDHEADER	
datawpos (word pos)	-
scale	UFFieldHeaderInfo.scale
startrangekm (km)	UFFieldHeaderInfo.startrange (km)
startrangem (m)	UFFieldHeaderInfo.startrange (km)
binspacing (m)	UFFieldHeaderInfo.binspacing (m)
nbin	UFFieldHeaderInfo.nbin
pulsewidth (m)	UFFieldHeaderInfo.pulsewidth (m)
beamwidthH (deg*64)	UFFieldHeaderInfo.beamwidthH (deg)
beamwidthV (deg*64)	UFFieldHeaderInfo.beamwidthV (deg)
rcvrbandwidth (MHz)	UFFieldHeaderInfo.rcvrbandwidth(MHz)
polarization (stat id)	UFFieldHeaderInfo.polarization (stat id)
wavelength (cm*64)	UFFieldHeaderInfo.wavelength (cm)
nsample	UFFieldHeaderInfo.nsample
threshfield[2]	UFFieldHeaderInfo.threshfield
threshval	UFFieldHeaderInfo.threshval
Scale	UFFieldHeaderInfo.scale
editcode[2]="na"	-

prt (us)	UFFieldHeaderInfo.prt (us)
nbitperbin=16	-
DMHEADER	
radarconstant (*scale)	UFFieldHeaderInfo.radarconstant
noisepower (dBm*scale)	UFFieldHeaderInfo.noisepower (dBm)
rcvrgain (dB*scale)	UFFieldHeaderInfo.rcvrgain (dB)
peakpower (dBm*scale)	UFFieldHeaderInfo.peakpower (dBm)
antgain (dB*scale)	UFFieldHeaderInfo.antgain (dB)
pulsedur (us*64)	UFFieldHeaderInfo.pulsedur (us)
VHEADER	
nyquistv (m/s*scale)	UFFieldHeaderInfo.nyquistv (m/s)
veflag[2]	UFFieldHeaderInfo.veflag

Global constants

==========

UF_MAX_NFIELDINRAY 100

  Maximum number of fields in each ray.

UF_MAX_NFIELDINREC 50

  Maximum number of fields in each record.

UF_MAX_NBIN 2500

  Maximum number of bins(gates) in each ray.

```
Global functions
================
long readray(int ufid, UFHeaderInfo* ufhptr, UFFieldHeaderInfo* uffhs,
 int uffhslen)
  Reads the next ray in the UF file into "UFHeaderInfo" and "UFFieldHeaderInfo"
  data structs and the "raydata" data buffer. On success, returns the size of
  the ray in number of bytes. Else returns -1, such as when end of UF file is
  reached.
  ufid - UF file id
  ufhptr - Pointer to "UFHeaderInfo" struct where general ray header info is
           stored.
  uffhs - Pointer to array of "UFFieldHeaderInfo" structs where field-specific
          ray header info is stored.
  uffhslen - length of the "UFFieldHeaderInfo" array buffer, i.e. max number of
             fields to read in.

long readprevray(int ufid, UFHeaderInfo* ufhptr, UFFieldHeaderInfo* uffhs,
 int uffhslen)
  Same as "readray", but reads the previous ray instead of the next.
  ufid - UF file id.
  ufhptr - Pointer to "UFHeaderInfo" struct where general ray header info is
           stored.
  uffhs - Pointer to array of "UFFieldHeaderInfo" structs where field-specific
          ray header info is stored.
  uffhslen - length of the "UFFieldHeaderInfo" array buffer, i.e. max number of
             fields to read in.
```

```
long readselectray(int ufid, int volnum, int raynum, UFHeaderInfo* ufhptr,
 UFFieldHeaderInfo* uffhs, int uffhslen)
  Same as "readray", but reads a selected ray instead of the next.
  ufid - UF file id.
  volnum - volume number of desired ray.
  raynum - ray number of desired ray.
  ufhptr - Pointer to "UFHeaderInfo" struct where general ray header info is
           stored.
  uffhs - Pointer to array of "UFFieldHeaderInfo" structs where field-specific
          ray header info is stored.
  uffhslen - length of the "UFFieldHeaderInfo" array buffer, i.e. max number of
             fields to read in.

int getfield(char* uffieldname, double* vec, int veclen, double badval)
  Reads values of selected field from "raydata" data buffer of current ray.
  uffieldname - name of field to read.
  vec - pointer to memory location where values will be stored.
  veclen - length of "vec" memory buffer, i.e. max number of values to read.
  badval - substitute value for bad/missing values.

void displayinfo(int ufid)
  Display info about current ray: ray start position, ray size, vol number, ray
  number.
  ufid - UF file id.

int pushfield(double* vec, int veclen, double badval, int scale)
```

```
  Writes field values into "raydata" data buffer of current ray.
  vec - data array to be written.
  veclen - length of "vec" array, i.e. max number of values to write.
  badval - substitute value for bad/missing values.
  scale - scaling factor (buffer value=actual value*scale).

int writeray(int ufid, UFHeaderInfo ufh, UFFieldHeaderInfo* uffhs, int nfield)
  Writes data structures and buffer of current ray to disk.
  ufid - UF file id.
  ufh - "UFHeaderInfo" struct.
  uffhs - "UFFieldHeaderInfo" structs for each field.
  nfield - number of fields to write.
```

ufchk utility

==========

ufchk is a program to inspect header information of UF files. It allows users to browse through each individual record (ray) of the UF file and display all the header values. Usage:

```
ufchk UFPATH
UFPATH: input UF file path.
```

Appendix E

PLOTNCGRID MANUAL

Manual for plotncgrid

A display for Cartesian-space NetCDF files

==========

Sean X. Zhang

Colorado State University

sxzazn@engr.colostate.edu

Last updated 2011 June 28

Introduction

==========

plotncgrid is a tool for displaying Cartesian-space NetCDF files. In addition to conventional colormaps, plotncgrid can also plot vector fields, designed for mapping wind products. Slices of 2D plots are produced along a 3rd dimension, providing a complete 3D visual representation of the data.

plotncgrid is solely developed by Sean X. Zhang (2009) at Colorado State University.

Installation

==========

```
1) Dependencies - plotncgrid requires the following software to be installed.
  Unidata NetCDF http://www.unidata.ucar.edu/software/netcdf/
  PNGwriter 0.5.4 http://pngwriter.sourceforge.net/

2) Configure "Makefile"
  PREFIX - Home directory of installation. Files will be placed relative to it.
    PREFIX/bin - binary files
    PREFIX/include - header files
    PREFIX/lib - library files
  ex: PREFIX=/usr/local/

  NCINCDIR - Directory where NetCDF header files reside.
  ex: NCINCDIR=-I/usr/local/include/

  NCLIBDIR - Directory where NetCDF library files reside.
  ex: NCLIBDIR=-L/usr/local/lib/

  PNGWINCDIR - Directory where PNGwriter header files reside.
  ex: PNGWINCDIR=-I/usr/include/

  PNGWLIBDIR - Directory where PNGwriter library files reside.
  ex: PNGWLIBDIR=-I/usr/include/

3) Run "make" to compile.

4) Run "make install" to install. Use "sudo" as needed.
```

5) Run "make clean" to clean out intermediate setup files.

6) Optionally copy run_plotncgrid_event.pl to execution directory.

Note: When running "make", configuration settings can be included, this will override the Makefile settings.
ex: make install PREFIX=/usr/local/

Usage
======

plotncgrid NCPATH CFGPATH OUTDIR

  NCPATH: input NetCDF file path.

  CFGPATH: config file path.

  OUTDIR: output directory.

run_plotncgrid_event.pl NCDIR CFGPATH OUTDIR

  NCDIR: input NetCDF file path.

  CFGPATH: config file path.

  OUTDIR: output directory.

As its name implies, plotncgrid should only be used to plot gridded NetCDFs, i.e. NetCDFs whose dimensions are Cartesian 3D. While the program can plot along any three dimensions, only dimensions corresponding to Cartesian 3D will yield the spatially-correct visual representation of the data.

In addition, plotncgrid requires the NetCDF to have at least three

"dimensional variables" corresponding to the axes values of the three dimensions to be plotted. For example, if a NetCDF is to be plotted against its Lat, Lon, and Alt dimensions, there needs to be three one-dimensional variables, each containing the axis values of Lat, Lon, and Alt, respectively. These three variables are known as the three "dimensional variables".

run_plotncgrid_event.pl is an optional PERL script that automates plotncgrid for entire data sets.

Configuration

==========

Configuration parameters are broken down into sections corresponding to each variable. Each variable section is delineated by [] brackets containing the variable name. Parameters are set with the "=" operator.

mode (fullframe|googlemap) default=fullframe

  Plotting modes:

  fullframe - Classic plot with margins, grids, colorbar, title, labels, etc...

  googlemap - Bezel-less plot designed to be superimposed on a map, NYI.

xvarname (str) default=x

  Dimensional variable containing X-axis values.

yvarname (str) default=y

  Dimensional variable containing Y-axis values.

```
slcvarname (str) default=z
  Dimensional variable containing Z-axis values, along which slices will be
  plotted.

slcvals ([min:spacing:max]|slc1,slc2,...) default=[0.5:0.5:2.5]
  Slice values to be plotted. Based on these specified value, the closest values
  along the slicing dimension will be plotted.

timeattname default=time
  Global attribute specifying the time of data acquisition (volume start time).

badvalattname default=badval
  Global attribute specifying the bad/missing value.

toground (0|1) default=0
  Plot Y-axis to 0, even if there is no defined value for those regions.
  Designed to give a relative-to-ground visual perspective when plotting RHI.

plotwind (0|1) default=0
  Plot vectors arrows, requires vector X-component and Y-component variables.
  Designed for plotting wind vectors, but can plot other vector value as well.

vxvarname (str) default=U
  Variable containing vector X-component.

vyvarname (str) default=V
  Variable containing vector Y-component.
```

veref (float) default=10.0

Reference velocity in m/s, the velocity magnitude represented by a vector arrow whose length equals the spacing between arrow origins.

arrowspacing (auto|int) default=auto

Arrow spacing, the pixel spacing between vector arrows. measured from arrow origins. "auto" sets arrow spacing to axis width in pixels divided by number of X-axis increments, with a minimum of 10.

cbarmin (float) default=0.0

Minimum value to be represented on the color bar. Data values smaller than "cbarmin" will be colorized as being equal to "cbarmin".

cbarmax (float) default=70.0

Maximum value to be represented on the color bar. Data values greater than "cbarmax" will be colorized as being equal to "cbarmax".

cbarinc (float) default=5.0

Colorbar increment that is represented by tick lines.

ncbarlvl (int) default=224

Colorbar palette count. If "ncbarlvl" is greater than the colormap palette count, it is set as equal to the colormap palette count. Otherwise, the colorbar palette will be a whole-range sampling of the colormap palette.

cmappath (str) default=./colormap/wdssii.colormap

```
  Color map path.

ncmaplvl (str) default=224
  Color map palette count. Nonarbitrary, must equal the actual palette count of
  the specified colormap file.

fontpath (str) default=./font/FreeSansBold.ttf
  Font file path.

fontsize (int) default=16
  Plot font size.

figwidth (int) default=1200
  Figure width in pixels.

figheight (auto|int) default=auto
  Figure height in pixels. "auto" sets height to maintain at least 1:1 increment
  ratio between X-axis and Y-axis values while also maitaining at least a 5:1
  width-to-height ratio visually (pixel count).

marginratio (float) default=0.1
  Percentage of figwidth dedicated to margins. e.g. marginratio=0.1 means all
  margin thicknesses are 10% of figwidth.

bgcolor (black|white) default=black
  Figure background color
```

```
arrowcolor (black|white) default=white
```

  Figure wind arrow color

References

[1] Armijo, L., 1969: A theory for the determination of wind and precipitation velocities with Doppler radars. *Journal of the Atmospheric Sciences*, **26**, 570-573.

[2] Barnes, S.L., 1980: Report on a meeting to establish a common Doppler radar data exchange format. *The Bulletin of the American Meteorological Society*, **61.11**, 1401-1404.

[3] Bringi, V.N., and V. Chandrasekar, 2001: *Polarimetric Doppler Weather Radar: Principles and Applications*. Cambridge University Press.

[4] Brotzge, J., 2006: Severe weather climatology of IP1. CASA Technical Report, Oklahoma University.

[5] Gal-Chen, T., 1982: Errors in fixed and moving frames of reference: applications for conventional and Doppler radar analysis. *Journal of the Atmospheric Sciences*, **39**, 2279-2300.

[6] Miller, L.J., and W. Anderson, 1991: *Multiple-Doppler Radar Wind Synthesis in CEDRIC*. National Center for Atmospheric Research, Mesoscale and Microscale Meterology Division.

[7] Miller, L.J., and S.M. Fredrick, 1998: *CEDRIC **C**ustom **E**diting and **D**isplay of **R**educed **I**nformation in **C**artesian Space - Manual*. National Center for Atmospheric Research, Mesoscale and Microscale Meterology Division.

[8] Oye, D., and M. Case, 1995: *REORDER: A Program for Gridding Radar Data - Installation and Use Manual for the UNIX Version*. National Center for Atmphospheric Research, Atmospheric Technology Division.

[9] Ray, P.S., et al., 1978: Triple-Doppler observations of a convective storm. *Journal of Applied Meteorology*, **17**, 1201-1212.

[10] Ray, P.S., et al., 1980: Single- and multiple-Doppler radar observations of tornadic storms. *Monthly Weather Review*, 108, 1607-1625

[11] Ruzanski, E., Y. Wang, and V. Chandrasekar, 2009: The CASA nowcasting system. *Extended Abstracts, World Meteorological Organization Symposium on Nowcasting*, Whistler, British Columbia, World Weather Research Programme, CD-ROM, 5.4.

[12] Wang, Yanting, V. Chandrasekar, and B. Dolan, 2008: Development of Scan Strategy for Dual Doppler Retrieval in a Networked Radar System. *IEEE Proceedings of International Geoscience and Remote Sensing Symposium 2008*.