MEASUREMENT OF SPIN STRUCTURE FUNCTION g_1 FOR THE PROTON AND DEUTERON IN THE RESONANCE REGION

Paul Edgar Raines

A DISSERTATION
for the Graduate Group in Physics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

1996

Supervisor of Dissertation

Graduate Group Chairperson

Acknowledgements

There are many people that deserve my thanks considering the numerous places I have spent my graduate career. First and foremost, I must thank my advisor, Keith Griffioen, for giving me the opportunity to go to SLAC and sticking through the bureacratic hassles and funding woes to keep me as his student.

I would like to thank all the members of the E143 collaboration and staff at SLAC. I am especially indebted to the other graduate students: Todd Averett, Johannes Bauer, Robin Erbacher, Jeff Fellbaum, Philippe Grenier, Masao Kuriki, T. J. Liu, and Philipp Steiner. It was a most pleasurable and educational experience working with them and I look forward to participating in future endeavors with them. I am grateful to Sebastian Kuhn, Linda Stuart, Frank Wesselmann and Charlie Young for their help in putting my analysis together. I would like to thank Ray Arnold, Peter Bosted, Emlyn Hughes and Steve Rock for their advise and guidance. I am grateful to Perry Anthony, Zen Szalata, and Joe White for allowing me to become intimately involved with ESA data acquisition projects.

I would like to thank David Balamuth for letting me get involved in the research at the Tandem Laboratory and to my fellow graduate students there, Judith Bush Newton and Ken Pohl, for their guidance and help. I am very grateful to David Armstrong, Mike Finn and Dirk Walecka at William and Mary for their help in solving my financial support issues.

A very heartfelt thanks to my parents for their support throughout my educational career for whatever goal I pursued. And a special thanks to Deborah for her support and understanding during the labor of putting this dissertation together.

ABSTRACT

MEASUREMENT OF SPIN STRUCTURE FUNCTION g_1 FOR THE PROTON AND DEUTERON IN THE RESONANCE REGION

Paul Edgar Raines

Keith Griffioen

Experiment E143 at SLAC measured the proton and deuteron spin structure functions g_1^p, g_2^p, g_1^d and g_2^d using polarized electrons at beam energies of 29, 16, and 9.7 GeV incident on polarized $^{15}NH_3$ and $^{15}ND_3$ targets. Two spectrometers detected the scattered electrons at the angles 4.5° and 7.0°, covering a kinematic range of $x > 0.03$ and $0.3 < Q^2 < 10$ GeV^2. Data taken at 9.7 GeV contain information on the nucleon resonances ($W^2 < 4$ GeV^2) at $Q^2 \approx 0.5$ and $Q^2 \approx 1.2$ GeV^2. This dissertation will describe the experiment and the analysis of the resonance data which yields results for g_1^p and g_1^d. These results are combined with previously published E143 deep-inelastic data at higher W^2 to extract the integrals $\Gamma_1^p(Q^2) \equiv \int_0^1 g_1^p(x,Q^2)dx$ and $\Gamma_1^d(Q^2) \equiv \int_0^1 g_1^d(x,Q^2)dx$. This provides the first accurate information on the low-Q^2 evolution of Γ_1 toward the Gerasimov-Drell-Hearn limit at $Q^2 = 0$.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A primary objective at the interface of nuclear and particle physics is to understand the inner structure of nucleons. One of the currently interesting questions is how the spin of a nucleon is distributed among its constituents. The simplest model for nucleon spin combines only the spin-$\frac{1}{2}$ valence quarks to form a state with total spin $\frac{1}{2}$. The nucleon's spin structure can be investigated by way of deep-inelastic polarized lepton-nucleon scattering from which one extracts the spin structure function g_1 from measured cross section asymmetries. Deep-inelastic scattering (DIS) corresponds to large squared four-momentum transfer Q^2. In this limit, electron scattering occurs essentially from a free quark and perturbative Quantum Chromodynamics (pQCD) works well to describe the interactions of the quarks and gluons.

Pioneering efforts were made at SLAC in the 70's to measure the nucleon's spin structure[1, 2], but within their large experiment error not much could be said to confirm or refute current theories. In 1988, results from a polarized muon-proton scattering experiment at CERN by the European Muon Collaboration (EMC)[3, 4] seemed to show that the valence quarks carried only a small fraction of the proton's spin. This so-called "spin crisis" led to an intense theoretical review and several additional polarized DIS experiments including those done by the Spin Muon Collaboration at CERN[5–8] and the E142 and E143 collaborations at SLAC[9–13]. These experiments confirmed the original CERN results with increasing precision and extended measurements to the deuteron and the neutron.

Although pQCD is on a firm footing at high Q^2, the nucleons that exist in our everyday world are far from this limit. How to bridge the gap between low-energy phenomenology where QCD is highly non-perturbative and the $Q^2 \to \infty$ limit is a matter of great current interest. For the polarized structure functions, two sum rules exist which predict values for the integral $\Gamma_1(Q^2) = \int_0^1 g_1(x, Q^2)dx$ at large and small limits of fixed Q^2. The Gerasimov-Drell-Hearn (GDH) sum rule[14, 15] relates Γ_1 at $Q^2 \approx 0$ to the nucleon's anomalous magnetic moment, while the Ellis-Jaffe sum rule[16, 17] relates Γ_1 at $Q^2 \to \infty$ to constants from nuclear beta decay.

As of yet, no theoretical model based on first principles exists to describe the evolution of the integral between the two limits. Precise experimental data are desired in this region to guide theoretical attempts to find a valid model. This is the goal of several experiments currently planned at TJNAF, MAMI and HERMES.

The E143 experiment, conducted at SLAC during the winter of 1993/94, measured the inclusive scattering of polarized electrons at energies of 9.7, 16.2, and 29 GeV from polarized protons and deuterons using $^{15}NH_3$ and $^{15}ND_3$ targets. The data taken at 9.7 GeV contain information on the region of the nucleon resonances ($W^2 < 4$ GeV^2) at $Q^2 \approx 0.5$ and $Q^2 \approx 1.2$ GeV^2. This dissertation describes the experiment and the analysis of the resonance region data which yields results for g_1^p and g_1^d. These results are combined with previously published E143 deep-inelastic structure functions at higher W^2 to extract the first integrals $\Gamma_1^p(Q^2)$ and $\Gamma_1^d(Q^2)$ for the region $0.5 < Q^2 < 1.5$ GeV^2.

Chapter 2

Theory

2.1 Electron-Nucleon Scattering

Electron-nucleon scattering provides a useful method for studying the structure of the nucleons. At low energy transfer, the electron will scatter elastically off the nucleon. As the energy transfer increases, the scattering becomes inelastic as the electron begins to excite various resonances in the nucleon. Increasing the energy transfer further into the deep-inelastic region, the nucleons themselves will break up losing their former identity in the debris.

To first order, electron-nucleon scattering proceeds by the one photon exchange process as shown in Figure 2.1. The incoming electron has energy E and momentum $\vec{k}$. The scattered electron has energy E' and momentum $\vec{k'}$. The angle between $\vec{k}$ and $\vec{k'}$ is θ. The exchanged virtual photon has four-momentum $q = k - k'$. The quantity P' represents any of the possible final states of the nucleon encompassing elastic and inelastic scattering. We will work in the lab frame where the target nucleon is at rest with mass M.

Several Lorentz invariants can be constructed from k, k' and P, including the four-momentum transfer squared Q^2, the energy transfer ν, the invariant mass

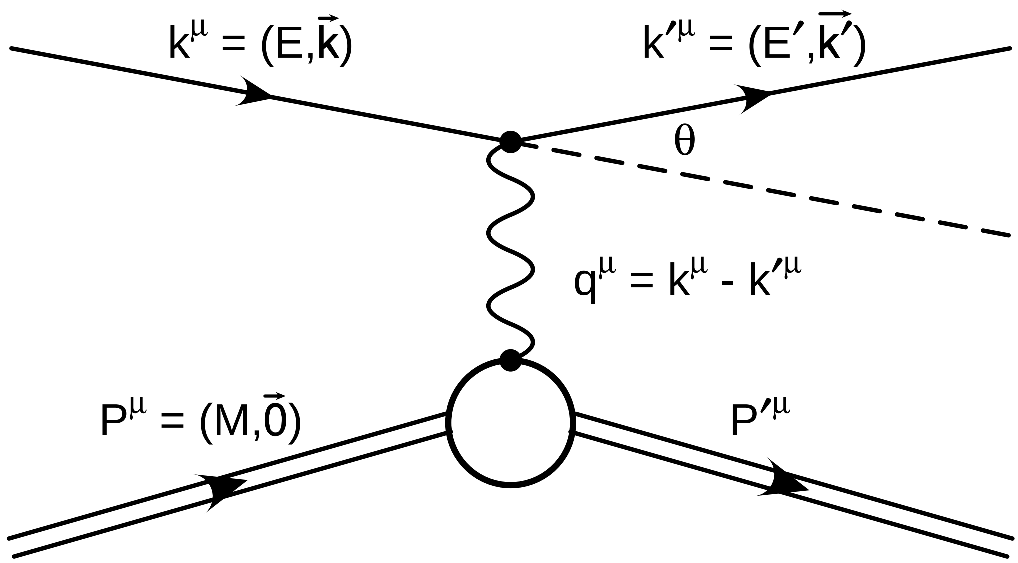

Figure 2.1: Feynman diagram of electron-nucleon scattering.

squared W^2, and the scaling variables x and y. These are defined as

$$
\begin{aligned}
Q^2 = -(q^\mu)^2 &= 2EE'(1-\cos(\theta)) \\
&= 4EE'\sin^2(\theta/2) \\
\nu = \frac{P^\mu q_\mu}{M} &= E - E' \\
W^2 = (P'^\mu)^2 &= M^2 - Q^2 + 2M\nu \\
x = \frac{-(q^\mu)^2}{2P^\mu q_\mu} &= \frac{Q^2}{2M\nu} \\
y = \frac{P^\mu q_\mu}{P^\mu k_\mu} &= \frac{\nu}{E}
\end{aligned}
$$

where the mass of the electron is neglected and the right-hand side values are for the lab frame. The variable x is also referred to as "Bjorken x". For simplicity in expressing the electron scattering formalism, it is convenient to define the following variables,

$$
\begin{aligned}
\epsilon &= 1/[1+2(1+\nu^2/Q^2)\tan^2(\theta/2)] \\
\gamma^2 &= Q^2/\nu^2 \\
\eta &= \epsilon\sqrt{Q^2}/(E-E'\epsilon) \\
\zeta &= \eta(1+\epsilon)/2\epsilon \\
\sigma_{\text{Mott}} &= \left[4\alpha^2(E')^2\cos^2(\frac{\theta}{2})\right]/Q^4
\end{aligned}
$$

where α is the fine structure constant.

For the one photon exchange process, the differential cross section for detecting the final electron in the solid angle $d\Omega$ is given by [18]

$$
\frac{d^2\sigma}{d\Omega dE'} = \frac{\alpha^2}{2MQ^4}\frac{E'}{E}L_{\mu\nu}W^{\mu\nu} \tag{2.1}
$$

The lepton current tensor, $L_{\mu\nu}$, can be written as a sum of symmetric (S) and antisymmetric (A) parts,

$$
\begin{aligned}
L_{\mu\nu} &= 2(L^{(S)}_{\mu\nu} + iL^{(A)}_{\mu\nu}) &(2.2)\\
L^{(S)}_{\mu\nu} &= k_\mu k'_\nu + k'_\mu k_\nu - g_{\mu\nu}(k\cdot k' - m^2) &(2.3)\\
L^{(A)}_{\mu\nu} &= m\varepsilon_{\mu\nu\alpha\beta}s^\alpha(k-k')^\beta &(2.4)
\end{aligned}
$$

where m is the electron mass, s the covariant four-vector spin of the incoming electron, $g_{\mu\nu} = (1,-1,-1,-1)$ the metric tensor, and $\varepsilon_{\mu\nu\alpha\beta}$ the completely antisymmetric tensor with $\varepsilon_{0123} = 1$. The final electron's spin is summed over since it is undetected.

The hadronic tensor $W_{\mu\nu}$ is similarly split into symmetric and antisymmetric parts,

$$W_{\mu\nu} = W^{(S)}_{\mu\nu} + iW^{(A)}_{\mu\nu} \tag{2.5}$$

$$\begin{aligned}\frac{1}{2M}W^{(S)}_{\mu\nu} &= \left(-g_{\mu\nu} + \frac{q_\mu q_\nu}{q^2}\right) W_1(\nu, Q^2) \\ &\quad + \left(P_\mu - \frac{P \cdot q}{q^2}q_\mu\right)\left(P_\nu - \frac{P \cdot q}{q^2}q_\nu\right)\frac{W_2(\nu, Q^2)}{M^2}\end{aligned} \tag{2.6}$$

$$\begin{aligned}\frac{1}{2M}W^{(A)}_{\mu\nu} &= \varepsilon_{\mu\nu\alpha\beta}q^\alpha \Big\{MS^\beta G_1(\nu, Q^2) \\ &\quad + \left[(P \cdot q)S^\beta - (S \cdot q)P^\beta\right]\frac{G_2(\nu, Q^2)}{M}\Big\}\end{aligned} \tag{2.7}$$

where S is the initial spin of the nucleon and $W_1(\nu, Q^2), W_2(\nu, Q^2), G_1(\nu, Q^2)$, and $G_2(\nu, Q^2)$ are four independent structure functions that contain information on the structure of the nucleon.

Averaging over initial electron and nucleon spins and summing over final spins (in the unpolarized case) picks out the symmetric terms in Eqs. 2.3 and 2.6 and 2.1 becomes

$$\begin{aligned}\frac{d^2\sigma^{\text{unp}}}{d\Omega dE'} &= \frac{\alpha^2}{MQ^4}\frac{E'}{E}L^{(S)}_{\mu\nu}W^{\mu\nu(S)} \\ &= \sigma_{Mott}\left[2W_1 \tan^2\frac{\theta}{2} + W_2\right]\end{aligned} \tag{2.8}$$

This is the unpolarized cross section which depends only on W_1 and W_2, the unpolarized structure functions. For brevity, we abbreviate this expression as simply σ.

For E143, the items of interest are the spin structure functions, G_1 and G_2. These can be extracted by measuring cross sections with known electron and nucleon polarizations. A difference between two measured cross sections with different polarization configurations is sensitive only to the antisymmetric terms in Eqs. 2.3 and 2.6, which depend only on the spin structure functions.

The simplest cases experimentally are those with the electron polarized parallel or anti-parallel to its momentum and the nucleon polarized parallel or perpendicular to the electron momentum. Forming the difference between the cross sections when the electron and nucleon polarizations are anti-parallel ($\downarrow\uparrow$) and parallel ($\uparrow\uparrow$)

produces the expression

$$
\begin{aligned}
\frac{d^2\sigma_{||}}{d\Omega dE'} &= \frac{d^2\sigma^{\downarrow\uparrow} - d^2\sigma^{\uparrow\uparrow}}{d\Omega dE'} = \\
&= 4\sigma_{Mott}\tan^2\frac{\theta}{2}\,\left[(E + E'\cos\theta)MG_1 - Q^2G_2\right] \qquad (2.9)
\end{aligned}
$$

Taking the configuration where the nucleon polarization is perpendicular to the incoming electron and aligned along the scattering plane, one gets the following expression

$$
\begin{aligned}
\frac{d^2\sigma_{\perp}}{d\Omega dE'} &= \frac{d^2\sigma^{\downarrow\leftarrow} - d^2\sigma^{\uparrow\leftarrow}}{d\Omega dE'} = \\
&= 4\sigma_{Mott}\tan^2\frac{\theta}{2}\,E'\sin\theta\,[MG_1 + 2EG_2] \qquad (2.10)
\end{aligned}
$$

For conciseness, we will use the notation of $\Delta\sigma_{||}$ and $\Delta\sigma_{\perp}$ to refer to the differential cross section differences. By measuring both quantities, one can obtain values for G_1 and G_2 individually. Solving for G_1 and G_2, one finds

$$
\begin{aligned}
G_1 &= \frac{1}{4\sigma_{\mathrm{Mott}}\tan^2(\theta/2)}\frac{1}{M(E+E')}\left[\Delta\sigma_{||} + \tan(\theta/2)\,\Delta\sigma_{\perp}\right] \qquad (2.11)\\
G_2 &= \frac{1}{4\sigma_{\mathrm{Mott}}\tan^2(\theta/2)}\frac{1}{2E(E+E')}\left[\left(\frac{E+E'\cos\theta}{E'\sin\theta}\right)\Delta\sigma_{\perp} - \Delta\sigma_{||}\right] \qquad (2.12)
\end{aligned}
$$

Note that at small values of the scattering angle θ, G_1 is primarily sensitive to $\Delta\sigma_{||}$ and G_2 to $\Delta\sigma_{\perp}$.

In many cases, it is more convenient to measure a cross section asymmetry defined as the ratio of the cross section difference to its sum. Corrections necessary for obtaining the absolute cross section, including spectrometer acceptance and detector efficiencies, cancel out in the ratio. The asymmetries are defined as

$$
\begin{aligned}
A_{||} &= \frac{\sigma^{\downarrow\uparrow} - \sigma^{\uparrow\uparrow}}{\sigma^{\downarrow\uparrow} + \sigma^{\uparrow\uparrow}} = \frac{\Delta\sigma_{||}}{2\sigma} = \frac{Q^2\left[(E+E'\cos\theta)MG_1 - Q^2G_2\right]}{2EE'\left[2W_1\sin^2(\theta/2) + W_2\cos^2(\theta/2)\right]} \qquad (2.13)\\
A_{\perp} &= \frac{\sigma^{\downarrow\leftarrow} - \sigma^{\uparrow\leftarrow}}{\sigma^{\downarrow\leftarrow} + \sigma^{\uparrow\leftarrow}} = \frac{\Delta\sigma_{\perp}}{2\sigma} = \frac{Q^2\sin\theta(MG_1 + 2EG_2)}{2E\left[2W_1\sin^2(\theta/2) + W_2\cos^2(\theta/2)\right]} \qquad (2.14)
\end{aligned}
$$

where the σ's represent the differential cross sections at the designated polarization configurations. The disadvantage of measuring asymmetries is that one must make a correction for counts due to unpolarized materials in the target. These counts will contribute to the denominator but not to the numerator when the measured asymmetry is formed. The measured asymmetry must be divided by a correction called the dilution factor, the ratio of total cross sections for polarizable and unpolarizable target materials, to remove the unwanted counts. Another difficulty with measuring asymmetries is that it requires knowledge of the unpolarized structure functions which are not always well known in the region of interest.

2.2 Deep Inelastic Scattering

Asymptotic freedom in QCD implies that as $Q^2 \to \infty$ the coupling between quarks goes to zero. In 1969, J. D. Bjorken[19] concluded that in the limit where both $\nu, Q^2 \to \infty$, quarks must behave like free, point-like charged leptons inside the nucleon. The scattering cross section in this limit, called the Bjorken limit or deep inelastic (DIS) regime, will be the incoherent sum of the elastic scattering from each quark. The nucleon structure functions no longer depend on ν and Q^2 independently but on a single variable $x = Q^2/2M\nu$.

This phenomena is called 'scaling'. The scaling variable x is most simply interpreted as the fraction of the total nucleon momentum carried by the struck quark. In practice, the scaling behavior breaks down logarithmically with Q^2 in perturbative QCD (pQCD). As Q^2 increases, the structure functions increase at low x and decrease at high x. This can be understood as the finer resolution of the exchanged photon at high Q^2 sees less of the virtual sea surrounding the struck particle which is less likely to carry a large fraction of the nucleon's momentum.

It is convenient to make a change of variables from ν to x and redefine the usual four structure functions as follows

$$F_1(x, Q^2) = MW_1(x, Q^2) \tag{2.15}$$

$$F_2(x, Q^2) = \nu W_2(x, Q^2) \tag{2.16}$$

$$g_1(x, Q^2) = M^2 \nu G_1(x, Q^2) \tag{2.17}$$

$$g_2(x, Q^2) = M\nu^2 G_2(x, Q^2) \tag{2.18}$$

Near the scaling limit, many references treat these functions as dependent on x only.

In leading order pQCD, the hadronic tensor $W_{\mu\nu}$ can be rewritten as a incoherent sum of its quark constituents each with its own leptonic tensor. Each quark is in motion with a fraction x of the nucleon's momentum. In this regime, the new structure functions can be simply expressed as the sums over accessible quark flavors of the quark distribution functions

$$F_1(x,Q^2) = \frac{1}{2}\sum_i e_i^2\left[q_i^{\uparrow}(x,Q^2)+q_i^{\downarrow}(x,Q^2)\right] \tag{2.19}$$

$$F_2(x,Q^2) = \sum_i xe_i^2\left[q_i^{\uparrow}(x,Q^2)+q_i^{\downarrow}(x,Q^2)\right] \tag{2.20}$$

$$\begin{aligned} g_1(x,Q^2) &= \frac{1}{2}\sum_i e_i^2\left[q_i^{\uparrow}(x,Q^2)-q_i^{\downarrow}(x,Q^2)\right] \\ &= \frac{1}{2}\sum_i e_i^2\Delta q_i(x,Q^2) \end{aligned} \tag{2.21}$$

where e_i is the fraction of an electron's charge carried by a quark of flavor i. The terms $q_i^{\uparrow}$ and $q_i^{\downarrow}$ are the quark distribution functions representing the probability, as a function of x and Q^2, of finding a quark or anti-quark of flavor i with spins aligned parallel ($\uparrow$) and anti-parallel ($\downarrow$) to the nucleon's spin. The term Δq_i is the difference between the quark distribution functions for the two spin alignments and thus represents the degree of polarization of quark flavor i in the nucleon. From Eqs. 2.19 and 2.20, one sees that in the scaling limit $F_2 = 2xF_1$. This is called the Callen-Gross relationship[20].

The dependence of the above structure functions on Q^2 can be predicted under pQCD. The Gribov-Lipatov-Altarelli-Parisi (GLAP) equations provide a method to evolve the structure functions from one Q^2 value to another in the perturbative region. These equations couple the quark distribution functions and the gluon distributions functions and account for the logarithmic variation with Q^2 of the structure functions.

In the Naive Quark-Parton Model, in which the quarks are completely independent and gluons are not considered, g_2 is predicted to be zero. Adding in the effects of intrinsic Fermi-motion, a finite but small value can be found for g_2 due to the transverse motions of the quark. However, due to its extreme sensitivity to the off-shell quark mass, this result is not a very reliable for predictions of g_2. Using the Operator Product Expansion (OPE)[21], Wandzura and Wilczek[22] found that in leading order pQCD, g_2 could be expressed in terms of g_1 as follows

$$g_2^{WW}(x,Q^2) = -g_1(x,Q^2) + \int_x^1 \frac{g_1(x',Q^2)}{x'}dx' \tag{2.22}$$

2.3 Virtual photon-nucleon asymmetries

A more direct understanding of the spin dependent nature of scattering in the resonance region can be obtained by considering the virtual photon-nucleon cross sections and asymmetries. From Fig. 2.1 we see that it is actually the virtual photon that probes the nucleon. Using the optical theorem, the virtual photon-nucleon cross sections are proportional to the imaginary part of the forward Compton helicity amplitudes[23]. These amplitudes are denoted by $M_{ab;cd}$ where a, b, c and d represent the helicities of the initial virtual photon and nucleon and final virtual photon and nucleon, respectively. Due to angular momentum conservation, parity and time reversal invariance, only four independent helicity amplitudes contribute. The standard choices are

$$\sigma^T_{\frac{1}{2}} = \frac{4\pi^2\alpha}{K} M_{1\frac{1}{2};1\frac{1}{2}} = \frac{4\pi^2\alpha}{MK}\left(F_1 + g_1 - \frac{2Mx}{\nu} g_2\right) \tag{2.23}$$

$$\sigma^T_{\frac{3}{2}} = \frac{4\pi^2\alpha}{K} M_{1-\frac{1}{2};1-\frac{1}{2}} = \frac{4\pi^2\alpha}{MK}\left(F_1 - g_1 + \frac{2Mx}{\nu} g_2\right) \tag{2.24}$$

$$\sigma^L_{\frac{1}{2}} = \frac{4\pi^2\alpha}{K} M_{0\frac{1}{2};0\frac{1}{2}} = \frac{4\pi^2\alpha}{MK}\left(\frac{F_2}{\nu}\left(1 + \frac{\nu^2}{Q^2}\right) - \frac{1}{M}F_1\right) \tag{2.25}$$

$$\sigma^{TL}_{\frac{1}{2}} = \frac{4\pi^2\alpha}{K} M_{0\frac{1}{2};-1-\frac{1}{2}} = \frac{4\pi^2\alpha}{MK}\frac{Q}{M\nu}(g_1 + g_2) \tag{2.26}$$

where $K = \nu - \frac{Q^2}{2M}$ is the virtual photon flux.

The terms $\sigma^T_{\frac{1}{2}}$ and $\sigma^T_{\frac{3}{2}}$ are the virtual photo-absorption cross sections for when the spin projection of the system along the virtual photon axis is $\frac{1}{2}$ and $\frac{3}{2}$, respectively. The term $\sigma^L_{\frac{1}{2}}$ is the cross section for longitudinally polarized incoming virtual photons while $\sigma^{TL}_{\frac{1}{2}}$ is the interference term between the transverse and longitudinal amplitudes. The ratio of longitudinal to transverse cross section is an important quantity which is defined as

$$R = \frac{2\sigma^L_{\frac{1}{2}}}{\sigma^T_{\frac{1}{2}} + \sigma^T_{\frac{3}{2}}} = \frac{\sigma^L_{\frac{1}{2}}}{\sigma_T} = \left(1 + \frac{\nu^2}{Q^2}\right)\frac{MF_2}{\nu F_1} - 1 \tag{2.27}$$

where

$$\sigma_T = \frac{1}{2}\left(\sigma^T_{\frac{1}{2}} + \sigma^T_{\frac{3}{2}}\right) \tag{2.28}$$

is the total transverse virtual cross section.

The virtual photon-nucleon asymmetries are defined as follows,

$$A_1 = \frac{\sigma^T_{\frac{1}{2}} - \sigma^T_{\frac{3}{2}}}{\sigma^T_{\frac{1}{2}} + \sigma^T_{\frac{3}{2}}} = \frac{1}{F_1}(g_1 - \gamma^2 g_2) \tag{2.29}$$

$$A_2 = \frac{\sigma^{TL}_{\frac{1}{2}}}{\sigma_T} = \frac{\gamma}{F_1}(g_1 + g_2) \tag{2.30}$$

Solving for g_1 and g_2 in terms of these new asymmetries, one finds

$$g_1 = \frac{F_1}{1+\gamma^2}(A_1 + \gamma A_2) \tag{2.31}$$

$$g_2 = \frac{F_1}{1+\gamma^2}(\frac{A_2}{\gamma} - A_1) \tag{2.32}$$

The magnitude of A_1 can obviously never exceed 1. It can be shown that

$$\left|\sigma^{TL}_{\frac{1}{2}}\right|^2 \leq \left|\sigma^L_{\frac{1}{2}}\right| \cdot \left|\sigma_T\right| \tag{2.33}$$

which leads by Eqs. 2.27 and 2.30 to the positivity limit,

$$|A_2| \leq \sqrt{R} \tag{2.34}$$

Since $R \to 0$ for infinite Q^2, A_2 must also approach zero for $Q^2 \to \infty$.

Using only basic quantum mechanics, it is instructive to get first order values for A_1 for both the elastic peak and Δ resonance for scattering on spin $\frac{1}{2}$ nucleons such as a proton. Since the photon is spin 1 and the proton is spin $\frac{1}{2}$, the particle in the final state can only have a total spin of $\frac{3}{2}$ or $\frac{1}{2}$. In the case of $\sigma^T_{\frac{3}{2}}$, the spin projection of the photon-proton system is $\frac{3}{2}$ so only the a final spin $\frac{3}{2}$ state is possible. For $\sigma^T_{\frac{1}{2}}$, both spin $\frac{1}{2}$ and $\frac{3}{2}$ states are possible. Figure 2.2 illustrates this in bra–ket notation along with appropriate Clebsch-Gordan coefficients.

In the case of elastic scattering, the final state is still a proton with total spin $\frac{1}{2}$. Therefore, only $\sigma^T_{\frac{1}{2}}$ contributes and

$$A_1^{elastic} = \frac{\frac{2}{3} - 0}{\frac{2}{3} + 0} = 1 \tag{2.35}$$

For the Δ resonance, the final state has a total spin of $\frac{3}{2}$. Both spin projections are

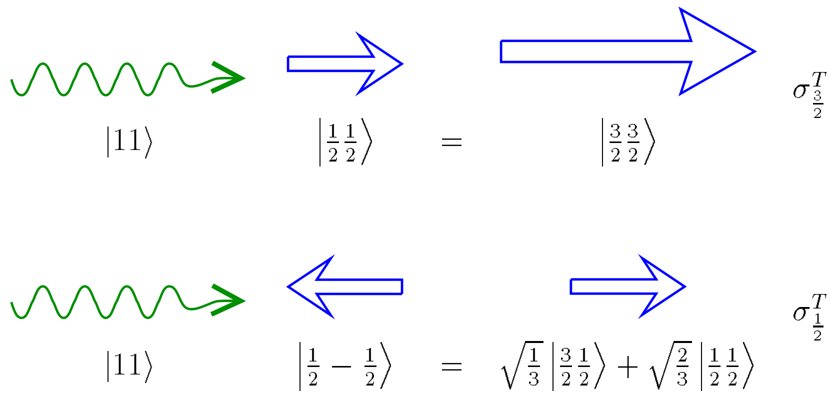

Figure 2.2: Transverse photon-proton scattering cross sections.

possible and therefore

$$A_1^{\Delta} = \frac{\frac{1}{3}-1}{\frac{1}{3}+1} = -\frac{1}{2} \tag{2.36}$$

The virtual photon-nucleon asymmetries are related to the electron-nucleon asymmetries as follows,

$$A_{||} = D(A_1 + \eta A_2) \tag{2.37}$$

$$A_{\perp} = d(A_2 - \zeta A_1) \tag{2.38}$$

where D (the depolarization factor) and d are kinematic coefficients defined as follows

$$D = \frac{1-\frac{E'\epsilon}{E}}{1+\epsilon R} \qquad d = D\sqrt{\frac{2\epsilon}{1+\epsilon}}$$

which account for that fact that the momentum vector of the virtual photon exchanged is not aligned with the polarization axis of the electron and nucleon.

2.4 Electron-deuteron scattering

The above formalism applies to scattering from individual nucleons. In the case of the deuteron, a first order estimate is to consider it as a simple combination of one proton and one neutron. When concerned with polarization states, one must take into account that the deuteron can have non-zero orbital angular momentum. The deuteron has total angular momentum of $\mathbf{J} = \mathbf{L} + \mathbf{S} = 1$, where $\mathbf{L}$ is the orbital angular momentum and $\mathbf{S}$ the total spin. Polarized electron-nucleon scattering is only sensitive to the z projection of the total spin, S_z.

When the deuteron is in its ground S-state with $\mathbf{L} = 0$, then $\mathbf{S} = 1$ and a measured polarization of 100% corresponds to $S_z = 1$. However, the deuteron ground state has a small admixture of the D-state with $\mathbf{L} = 2$ and $\mathbf{S} = 1$. On evaluating the Clebsch-Gordan coefficients, one finds at 100% target polarization, the deuteron is 10% likely to be in a $S_z = +1$ state, 30% in a $S_z = 0$ state, and 60% in a $S_z = -1$ state. If A_S is the measured asymmetry of the S-state, then the measured asymmetry of the D-state is related by

$$A_D = 0.1A_S - 0.6A_S = -0.5A_S \tag{2.39}$$

Taking w_D as the probability of the deuteron to be in the D-state, the total measured asymmetry would be

$$A = (1 - w_D)A_S - w_D A_D = (1 - 1.5w_D)A_S \tag{2.40}$$

A numerical value for w_D cannot be measured directly. However, several phenomenological models exist that predict it[24–26]. For this analysis, a value of $w_D = 0.05 \pm 0.01$ was used to cover the range of predictions.

The asymmetry of the S-state is expected to be the average of the proton and neutron, weighted by the cross-sections. Considering the Callen-Gross relation, it is equivalent in the scaling limit to weighting only by the unpolarized structure function F_1. Adopting the convention that quantities for the deuteron represent the average nucleon (i.e. are on a per nucleon basis),

$$F_1^d = \frac{1}{2}(F_1^p + F_1^d), \tag{2.41}$$

the deuteron's asymmetry is expressed by

$$A^d = (1 - 1.5w_D)\left[\frac{F_1^p}{2F_1^d}A^p + \frac{F_1^n}{2F_1^d}A^n\right] \tag{2.42}$$

where A may represent $A_{||}$, $A_{\perp}$, A_1 or A_2. From Eq. 2.31, one finds

$$g_1^d = \frac{1 - 1.5w_D}{2}(g_1^p + g_1^n) \tag{2.43}$$

At high x, the effects of Fermi motion are important and must be corrected for. The motion of the proton and neutron relative to the center of mass of the nucleon will modify the actual incident energy E and thus "smear" the data. At lower x and outside the resonance region, the EMC effect and nuclear shadowing become important[27]. However, for the deuteron, these effects are small.

2.5 Sum Rules

Several sum rules exist that give predictions for the integrals of spin structure functions (and combinations) over the range of x from 0 to 1. The experimental validity of these sum rules test how well the spin structure functions are understood in QCD. DIS experiments attempt to cover as large a range in x and Q^2 as possible given the constraints of beam energy and detector acceptances. In E143, the range of x covered at 9.7 GeV was 0.03 to 1. This is referred to as the data region. Between 0 and 0.03, an extrapolation must be made based on theoretical predictions. The behavior of the integrals in this region is an area of much debate and a major contributor to systematic error. In discussing the following sum rules, we will use the common shorthand for the integral of

$$\Gamma_1(Q^2) = \int_0^1 g_1(x, Q^2)dx \tag{2.44}$$

2.5.1 Bjorken sum rule

In 1966, Bjorken derived a sum rule[28] based purely on light cone current algebra with the assumption of isospin symmetry between the proton and neutron quark distribution functions. This rule relates the ratio of the axial-vector and vector coupling constants from neutron beta decay, $\left(\frac{g_A}{g_V}\right)_n$, to the difference between the integrals for $g_1^p(x)$ and $g_1^n(x)$.

The ratio of the axial-vector and vector coupling constants can be expressed in terms of the quark distributions by

$$\left(\frac{g_A}{g_V}\right)_n = \frac{1}{2}(u_p^\uparrow - u_p^\downarrow) - \frac{1}{2}(d_p^\uparrow - d_p^\downarrow) - \left[\frac{1}{2}(u_n^\uparrow - u_n^\downarrow) - \frac{1}{2}(d_n^\uparrow - d_n^\downarrow)\right] \tag{2.45}$$

where $u_{p(n)}$ is the total (over all x) quark distribution function representing the probability of finding an up quark or anti-quark in the proton (neutron) with its spin aligned ($\uparrow$) or anti-aligned ($\downarrow$) relative to the spin of the proton (neutron). The quantity $d_{p(n)}$ is the same quantity for down flavor. Using isospin symmetry, $u_p \equiv d_n$ and $d_p \equiv u_n$, Eq. 2.45 simplifies to

$$\begin{aligned}\left(\frac{g_A}{g_V}\right)_n &= (u_p^\uparrow - u_p^\downarrow) - (d_p^\uparrow - d_p^\downarrow) \\ &= \Delta u - \Delta d\end{aligned} \tag{2.46}$$

where we have used a shorthand of $\Delta q = q_p^\uparrow - q_p^\downarrow$ which is referred to as the expectation value of the polarization of quark flavor q in the proton.

Assuming contributions from only the up, down and strange quarks, we can expand Eq. 2.21 for both the proton and neutron

$$2g_1^p(x) = \frac{4}{9}\Delta u(x) + \frac{1}{9}\Delta d(x) + \frac{1}{9}\Delta s(x) \tag{2.47}$$

$$2g_1^n(x) = \frac{4}{9}\Delta d(x) + \frac{1}{9}\Delta u(x) + \frac{1}{9}\Delta s(x) \tag{2.48}$$

where we have applied isospin asymmetry so that all the quark distributions refer to those in the proton. Subtracting g_1^n from g_1^p and integrating over x yields

$$\Gamma_1^p - \Gamma_1^n = \frac{1}{6}\left[\Delta u - \Delta d\right] \tag{2.49}$$

Combining this result with Eq. 2.47, we obtain the Bjorken sum rule,

$$\Gamma_1^p - \Gamma_1^n = \frac{1}{6}\left(\frac{g_A}{g_V}\right)_n \tag{2.50}$$

This expression is only valid at the scaling limit, $Q^2 \to \infty$, so at finite Q^2 QCD corrections must be made. These corrections are separated into two types: corrections in orders of the strong interaction coupling constant α_s and corrections in orders of $1/\sqrt{Q^2}$. The former are called "leading twist" or "twist-2" while the latter are referred to as "higher twist" (twist-3 $\sim 1/\sqrt{Q^2}$, twist-4 $\sim 1/Q^2$, etc.). To third order in leading twist[29], the Bjorken sum rule is written as follows

$$\Gamma_1^p - \Gamma_1^n \quad = \quad \frac{1}{6}\left(\frac{g_A}{g_V}\right)_n \left(1 - \frac{\alpha_s}{\pi} - 3.583\frac{\alpha_s^2}{\pi^2} - 20.215\frac{\alpha_s^3}{\pi^3}\right) \tag{2.51}$$

The Bjorken sum rule is considered to be fundamental because the only assumption made is the well accepted isospin symmetry between the neutron and proton. Feynman himself claims that 'Its verification, or failure, would have a most decisive effect on the direction of future high energy theoretical physics'[30]. Fortunately, experiments so far, have found that the sum rule is on solid ground. The E143 deep-inelastic analysis found a value of $0.154 \pm 0.010(stat) \pm 0.016(sys)$[31] at $Q^2 = 3$ GeV^2. Using $\alpha_s = 0.35 \pm 0.05$[32, 33] and $\frac{g_A}{g_V} = 1.2573 \pm 0.0028$[34], Eq. 2.51 predicts a value of 0.171 ± 0.009.

2.5.2 Ellis-Jaffe sum rules

The Ellis-Jaffe sum rules give predictions for the individual integrals Γ_1^p and Γ_1^n[16, 17]. Their derivations require assumptions of SU(3) flavor symmetry and an unpolarized strange quark contribution ($\Delta s = 0$). Therefore, these rules are considered less fundamental that the Bjorken sum rule.

The ratio of the axial vector coupling constant to the vector coupling constant in Σ^- decay can be expressed in terms of the quark distributions in the neutron using V-spin symmetry[1] according to

$$\left(\frac{g_A}{g_V}\right)_{\Sigma^-} = -(u_n^\uparrow - u_n^\downarrow) - (s_n^\uparrow - s_n^\downarrow) \tag{2.52}$$

Then using isospin symmetry as in Eq. 2.46, this can be written in terms of the proton quark distribution difference as

$$\left(\frac{g_A}{g_V}\right)_{\Sigma^-} = \Delta d - \Delta s \tag{2.53}$$

where the strange content of both neutron and proton are assumed equal.

Combining Eqs. 2.47 and 2.48 separately with Eqs. 2.47 and 2.53, the integrals

[1] V-spin symmetry is related to isospin symmetry but relates the u and s quarks rather than the u and d quarks. U-spin symmetry between d and s quarks complete the SU(3) flavor symmetry picture.

over x of g_1^p and g_1^n are given by

$$\Gamma_1^{p(n)} = \pm\frac{1}{12}\left(\frac{g_A}{g_V}\right)_n + \frac{5}{36}\left[\left(\frac{g_A}{g_V}\right)_n + 2\left(\frac{g_A}{g_V}\right)_{\Sigma^-}\right] + \frac{1}{3}\Delta s \tag{2.54}$$

where the + sign is for the proton and the − is for the neutron. Using two alternate constants from baryon decay, F and D, where

$$F + D = \left(\frac{g_A}{g_V}\right)_n = \Delta u - \Delta d \tag{2.55}$$

$$F - D = \left(\frac{g_A}{g_V}\right)_{\Sigma^-} = \Delta d - \Delta s \tag{2.56}$$

Eq. 2.54 can be written as

$$\Gamma_1^p = \frac{1}{18}\left[9F - D + 6\Delta s\right] \tag{2.57}$$

$$\Gamma_1^n = \frac{1}{18}\left[6F - 4D + 6\Delta s\right] \tag{2.58}$$

If one assumes that strange quark sea is not polarized in the nucleon ($\Delta s = 0$), the integrals are expressed purely in terms of the well measured coupling constant ratios. These relations are the Ellis-Jaffe sum rules.

As with the Bjorken sum rule, at finite Q^2, QCD corrections are needed. To third order in leading twist[35], the Ellis-Jaffe sum rule can be written as

$$\begin{aligned}\Gamma_1^{p(n)} = &\left[1 - \frac{\alpha_s}{\pi} - 3.5833(\frac{\alpha_s}{p})^2 - 20.2153(\frac{\alpha_s}{\pi})^3\right] \\ &\times\left(\pm\frac{1}{12}(F + D) + \frac{1}{36}(3F - D)\right) \\ &+\left[1 - 0.3333(\frac{\alpha_s}{\pi}) - 0.5495(\frac{\alpha_s}{\pi})^2\right]\frac{1}{9}(3F - D)\end{aligned} \tag{2.59}$$

The predictions of the Ellis-Jaffe sum rule for the proton, deuteron, and neutron at $Q^2 = 3$ GeV^2 using $F - D = -0.340 \pm 0.017$[34] are shown in Table 2.1 along with the results of the E143 deep-inelastic analysis[31].

Disagreement between prediction and measurement is significant (over three standard deviations for the proton and deuteron). Other experiments are consistent with the E143 results and also differ significantly from theory[6, 8, 9]. Recalling that the Ellis-Jaffe sum rules assumes that the strange quark sea is unpolarized, one can

	Prediction	E143 DIS results
Γ_1^p	0.160±0.007	0.127±0.003±0.008
Γ_1^d	0.069±0.004	0.046±0.003±0.004
Γ_1^n	-0.011±0.005	-0.027±0.008±0.010

Table 2.1: Comparison of predictions for Ellis-Jaffe sum rules with results of E143 deep-inelastic analysis at $Q^2 = 3$ GeV^2.

interpret the difference to be a measurement of Δs. For the E143 deep-inelastic results, this corresponds to $\Delta s = -0.10 \pm 0.03$ for the proton and $\Delta s = -0.08 \pm 0.02$ for the deuteron. Using Eqs. 2.55 and 2.56, one can also extract the net contribution of the quarks to the nucleon polarization, $\Delta u + \Delta d + \Delta s$. For the proton, E143 determined this sum to be 0.27 ± 0.10[10] which implies that the quarks are responsible for only about 30% of the nucleon's spin.

One should also note that another assumption of the Ellis-Jaffe sum rule is SU(3) symmetry which is known to be broken. How much this symmetry breaking should effect the sum rules is currently a matter of discussion (see for example Ref. [36]).

A sum rule also exists for the integral of the g_2 spin structure functions called the Burkhardt-Cottingham sum rule[37], which predicts at $Q^2 \to \infty$

$$\Gamma_2^{p,n} = \int_0^1 g_2^{p,n} dx = 0 \tag{2.60}$$

The BC sum rule is derived by considering the asymptotic behavior of virtual Compton helicty amplitudes. It also assumes that g_2 is well-behaved at low x which need not be the case. Also, a non-zero value for this rule could occur when one takes into account the quark transverse spin distribution and higher twist corrections.

2.5.3 Gerasimov-Drell-Hearn sum rule

At the other end of the Q^2 spectrum, the Gerasimov-Drell-Hearn (GDH) sum rule[14, 15] provides a method to predict the value of Γ_1 at $Q^2 = 0$. Note that at $Q^2 = 0$ one deals with real photons. The GDH sum rule relates the integral of the virtual photon-nucleon cross section difference to the anomalous nucleon magnetic moment

κ according to

$$\frac{2\pi^2\alpha}{M^2}\kappa_{p,n}^2 = \int_{\nu_{thr}}^{\infty} \left[\sigma^T_{\frac{3}{2}}(\nu) - \sigma^T_{\frac{1}{2}}(\nu)\right]_{p,n} \frac{d\nu}{\nu} \tag{2.61}$$

where $\nu_{thr} = Q^2/2M$ and the contribution of the elastic peak is excluded. We can relate this to Γ_1 using Eqs. 2.23 and 2.24 to obtain

$$\Gamma_1 = \int_0^1 \frac{\nu - \frac{Q^2}{2M}}{8\pi^2\alpha/M} \left(\sigma^T_{\frac{3}{2}}(\nu) - \sigma^T_{\frac{1}{2}}(\nu)\right) dx \tag{2.62}$$

Changing the variable of integration from x to ν leads to

$$\Gamma_1 = \frac{Q^2}{16\pi^2\alpha} \int_{\nu_{thr}}^{\infty} (1-x) \left(\sigma^T_{\frac{3}{2}}(\nu) - \sigma^T_{\frac{1}{2}}(\nu)\right) \frac{d\nu}{\nu} \tag{2.63}$$

Assuming the elastic contribution to Γ_1 is negligible or has been explicitly subtracted, Eq. 2.63 can be related to the GDH sum rule by dividing by Q^2 and taking the limit as $Q^2 \to 0$,

$$\lim_{Q^2\to 0} \frac{\Gamma_1}{Q^2} = \frac{1}{16\pi^2\alpha} \int_{\nu_{thr}}^{\infty} \left(\sigma^T_{\frac{3}{2}}(\nu) - \sigma^T_{\frac{1}{2}}(\nu)\right) \frac{d\nu}{\nu} = -\frac{\kappa^2}{8M^2} \tag{2.64}$$

In other words, when Γ_1 is plotted as a function of Q^2, it will approach the origin with a slope equal to $-\kappa^2/8M^2$. Using the experimental values $\kappa_p = 1.79$ and $\kappa_n = -1.91$[34], this slope is equal to -0.45 GeV^{-2} for the proton and -0.52 GeV^{-2} for the neutron. Note that this implies that near $Q^2 = 0$, Γ_1 is a negative quantity. For the proton, the Ellis-Jaffe sum rule predicts a positive value for the Γ_1 at $Q^2 \to \infty$. Therefore, the integral must cross through zero at least once somewhere in between. Previous experiments at $Q^2 \geq 3$ GeV^2 have all measured values for Γ_1 that are distinctly positive.

Various theoretical models have been proposed that interpolate Γ_1 between the GDH and scaling limit[38–40]. One of the major goals of this analysis is to produce accurate values for Γ_1 at $Q^2 < 3$ GeV^2 for comparison to these predictions. The models and results of this analysis will be discussed in detail in the conclusions.

Chapter 3

Experimental Setup

The Stanford Linear Accelerator Center (SLAC) is a two mile long electron LINAC capable of producing polarized electron beams pulses at 120 Hz of up to 50 GeV in energy and >80% polarization. The beam can be delivered to a number of experimental facilities on site. The E143 Collaboration utilized the End Station A (ESA) facility. In this large warehouse-sized building, apparatus was assembled to measure the incoming beam charge, profile, and polarization. A polarized target was placed in the beam path and two spectrometers were built at angles 4.5° and 7.0° to the beam line to detect the scattered electrons. Behind ESA, a long chamber called Beam Dump East absorbed unscattered beam. Cables were run from the spectrometers into the counting house where the signal processing electronics and computers handled the tasks of data acquisition and experiment control.

Data were taken in units called runs. During a run, every effort was made to keep beam, target, detector, and electronics conditions constant. When a change occurred (i.e target switch, beam energy change, electronic adjustments), a new run would be started. A run would be stopped and a new run started after about 70 minutes even if no conditions were changed in order to maintain a reasonable data set size per run. There were over 3000 runs during E143.

3.1 Polarized Electron Source

Pulses of longitudinally polarized electrons were supplied to the accelerator through photoemission from a p-doped, strained-lattice GaAs cathode[41]. The GaAs crystal was illuminated by circularly polarized photons generated by a flash-lamp-pumped Ti-sapphire laser operated at 845 nm. A layout of the source apparatus is shown in

Figure 3.1.

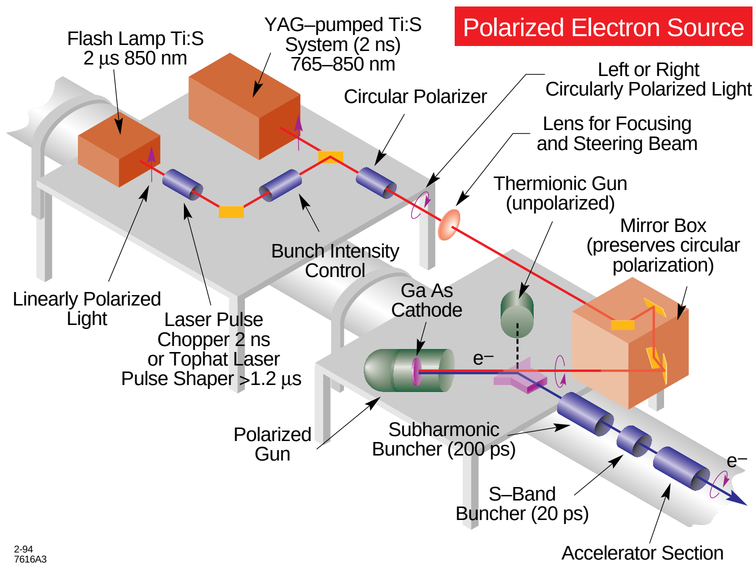

Figure 3.1: Layout for the SLAC polarized electron source.

Sources based on traditional unstrained GaAs are limited to a maximum of 50% polarization. Due to the spin-orbit interaction, the top level of the valence band is split by 0.34 eV into a fourfold degenerate $P_{\frac{3}{2}}$ level and a twofold degenerate $P_{\frac{1}{2}}$ level as shown in Figure 3.2. Transitions to the lowest level of the conduction band, a $S_{\frac{1}{2}}$ state, from the $P_{\frac{3}{2}}$ state are induced by laser light at the band gap energy 1.52 eV. If the light is circularly polarized with helicity ±1 then due to angular momentum conservation, only transitions of $\Delta m_j = \pm1$ are allowed. For positive helicity, the possible transitions are $m_j = -\frac{3}{2} \rightarrow m_j = -\frac{1}{2}$ and $m_j = -\frac{1}{2} \rightarrow m_j = \frac{1}{2}$. Due to Clebsh-Gordan coefficients, these transitions have a relative probability of 3 and 1, respectively, which leads to a maximum theoretical polarization of $(3-1)/(3+1)$ $= 50\%$.

A strained-lattice GaAs-GaAs$_{1-x}$P$_x$ cathode is created by placing GaAs of proper thickness on top of a GaAs$_{1-x}$P$_x$ sublayer. The resulting bi-axial compressive strain lifts the degeneracy between the $m_j = \pm\frac{3}{2}$ and $m_j = \pm\frac{1}{2}$ states of the $P_{\frac{3}{2}}$ level. A precisely tuned laser can then drive only one of the two transitions mentioned above and achieve 100% polarization under ideal conditions. To increase the quantum efficiency, defined as the ratio of photoemitted electrons to the number of incident photons, a thin layer of Cesium oxide was periodically applied to the cathode. The

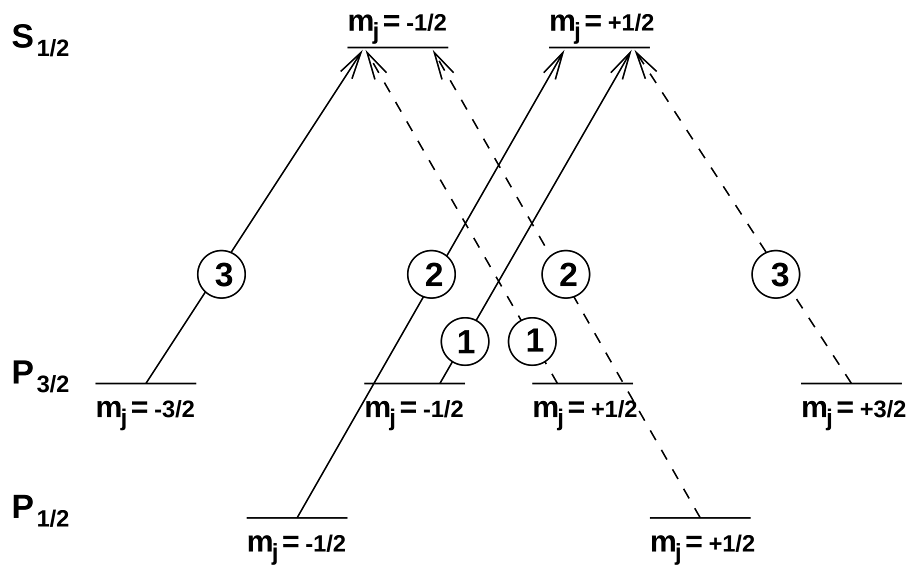

Figure 3.2: Energy level diagram for unstrained GaAs. Solid lines represent right handed transitions and the dashed lines represent left handed transitions. The circled numbers denote the relative probability of the transition.

presence of the Cesium lowers the bulk conduction band below the vacuum level making it easier to for electrons in the conduction band to escape. For the E143 experiment, the actual polarization observed was between 83% and 86%.

The direction of polarization of ejected electrons can easily be reversed by switching from positive helicity light to negative helicity light. The laser provides a uniform 2.3μs pulse which is then circularly polarized by a Pockels cell. The Pockels cell is a birefringent crystal in which the sign of the circular polarization is controlled by the sign of the high voltage applied. During E143, the sign was controlled pulse to pulse by a pseudo-random number generator in order to reduce systematic effects.

The SLAC source is capable of currents greater than $1 \times 10^{11} e^-$/pulse. However, for E143, the beam intensity was limited to below $4 \times 10^9 e^-$/pulse due to the limitations of the polarized target and the counting rate ability of the spectrometers.

3.2 Beam

Electrons from the source are injected into the main accelerator at an energy of 120 keV and then accelerated for roughly two miles to the desired energy before being switched into ESA. The acceleration was achieved by a series of approximately 250

Klystrons which deliver high power RF near 4 GHz to a series of copper cavities that create a standing wave electric field. The exact frequency of the RF and size of the cavities is chosen to match the speed of the electron bunches so that they constantly see a field that accelerates them from one cavity to the next.

3.2.1 Energy Measurement

After acceleration, the electrons were directed into ESA by eight magnets known collectively as the A-bend. These eight magnets were connected to the same power supply in series with a ninth reference magnet, identical in design to the others, that was located off the beam line in the Main Control Center (MCC). A flip-coil located in this reference magnet measured the field and thus, once properly calibrated, served as the method to measure the momentum of the electrons in the beam. An NMR probe was also used to provide an additional check. A series of adjustable slits located between the fourth and fifth magnet were used to control the energy spread of each pulse. The spread was maintained at less than 1% for E143.

The beam was bent by $24.5°$ along the A-bend which causes the spin of the electrons to precess. Due to the electron's anomalous magnetic moment, the net precession is different than the beam angle so by careful selection of the beam parameters a net change of zero in the spin alignment is possible. The relationship of the magnetic field, B, and beam energy, E, at fixed angle, θ, for electrons traveling very close to the speed of light is

$$eBct = \theta E \tag{3.1}$$

where t is the duration of the deflection. In a uniform magnetic field, the angle between the direction of polarization and momentum, ϕ, changes in time according to

$$\frac{d\phi}{dt} = \frac{eB}{2mc}(g-2). \tag{3.2}$$

For the $\theta = 24.5° = 0.1361\pi$ deflection angle of the A-bend, one finds the change in ϕ is

$$\delta\phi = \frac{d\phi}{dt}t = \frac{eB}{2mc}(g-2)t = \frac{0.1361\pi E(g-2)}{2mc^2}. \tag{3.3}$$

To retain the longitudinal polarization of the electrons, $\delta\phi$ must be an integer multiple of π which then implies a condition on the beam energy that

$$E = \frac{2Nmc^2}{0.1361(g-2)} = 3.237N\ GeV. \tag{3.4}$$

For E143, the values of N were 3, 5, and 9 corresponding to beam energies 9.711, 16.185, and 29.133 GeV, respectively. Since the polarization of the beam entering ESA is a maximum only at these values, measurement of the beam polarization while sweeping a small range in energies around the above values provides an excellent method of calibrating the A-bend. The Møller polarimeter described below in section 3.3 was used in this manner to show that the A-bend measurement was off by about 0.05%.

3.2.2 Rastering

The $^{15}NH_3$ and $^{15}ND_3$ targets used in E143 (see 3.4 below) require temperatures be below $\sim$ 1 K in order to obtain maximum polarization. If the beam were to be maintained at one spot on the target, the localized beam heating would make this impossible even with the available high-power cooling refrigerator. Accumulated radiation damage is also a problem for maintaining ideal polarization. For these reasons, in order to make best use of all the available material, the beam was rastered over the target.

To reduce beam heating further, a 2.3 μs beam pulse was used. Typical beam pulses used at SLAC are on the order of a few nanoseconds. This presented a problem with the use of several beam monitoring devices along the linac calibrated for the rapid current gradient of smaller pulses. In order to use these monitors, one pulse out of the 120 pulses/sec called the witness pulse was created a few nanoseconds in length at high intensity and not steered into the A-bend.

The rastering was accomplished by a pair of Helmholtz coils located just outside ESA about 60 meters upstream of the target. For each pulse, the coils were supplied with a programmed current that stepped the beam spot through a prescribed pattern on the target as shown in Figure 3.3. The raster pattern is a grid of 256 positions separated by 1 mm at the face of the target. Each successive spill skips either two positions in x and/or two positions in y. It therefore takes four passes over the target to cover each grid point.

Power supply problems would sometimes keep a position from changing or cause it to be skipped. This was not considered a significant problem for the target because estimates showed that at least ten successive spills on the same spot would be required to have a noticeable effect on the polarization. Also, at 120 Hz, two beam pulses occur in the normal 60 Hz power cycle. A dependence between the two timeslots was observed in the output of the power supply causing a slight shift in the amount of beam deflection by the raster magnets. The shift in beam position at the target that resulted was small enough not to affect the target adversely.

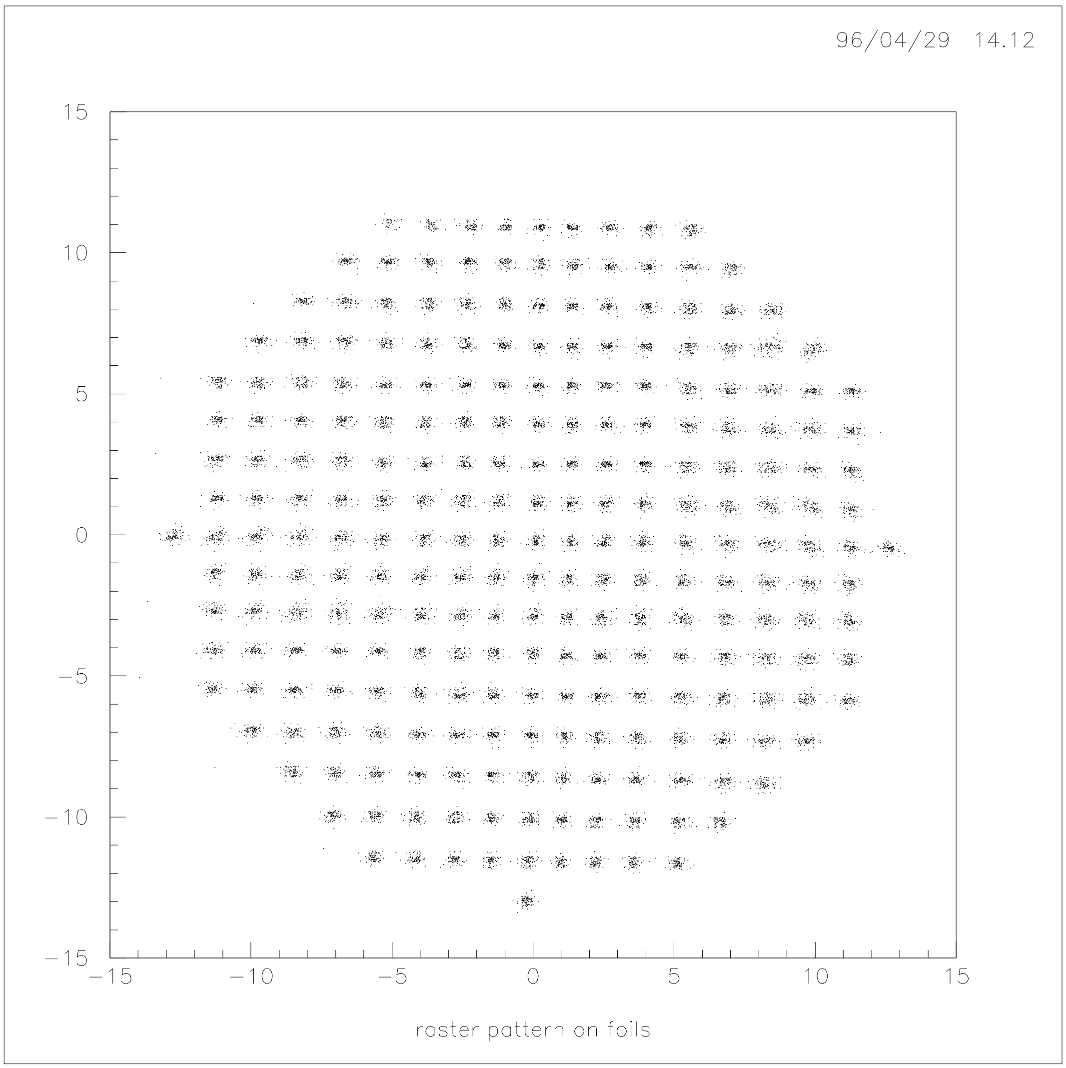

Figure 3.3: Raster pattern of beam on the foil array.

3.2.3 Monitoring

To align the beam on target, two roller screens were installed in ESA. One was located near the ESA beam entrance and the other was located just before the target. Each roller screen consisted of a sheet of fluorescent material mounted on rollers and placed in the beamline. Along one section of the screen a large hole allowed the beam to pass undisturbed. The fluorescent screen can be rolled into the beam by controls in the counting house. A video camera that viewed the screen allowed visual inspection of the beam's location and profile. A set of cross hairs was placed in front of the camera to give a reference point for beam alignment.

A "foil array" was placed a 11 meters downstream of the target to measure the beam position and profile of the unscattered beam. This device operates as a secondary emission monitor. A set of 48 thin vertical foils placed edge-on to the beam together with a flat collector plate measured the x direction (horizontal axis perpendicular to the beam direction) while another set of 48 horizontal foils with collector plate measured the y direction (vertical axis perpendicular to the beam direction). Each foil was made of brass (70% Cu, 30% Zn) and was 1 mil thick and 1 cm wide. The foils were separated by 1 mm.

Each foil was held at a high potential relative to the collector plates. When an electron entered the foil, it knocked out a large number of low energy electrons. The ejected electrons were absorbed by the collector plate thereby producing a current spike that was integrated by an ADC in the counting house. To reduce noise and grounding problems, the signal was passed through a transformer and a high pass filter before entering the ADC. Foils were used instead of wires because the edge-on foil configuration provided the larger secondary emission required to monitor a low beam intensity. Typical values for the beam position and profile on the foils are shown in Figures 3.4 and 3.5.

A slab ($\sim$ 0.5 meter) of scintillator attached to a phototube was placed near the beam entrance to ESA a few feet from the beam pipe. The signal was sent to an oscilloscope in the counting house as well as to an ADC which integrated the signal over the entire spill. This device was called the bad spill monitor since any significant signal meant that the beam was badly steered or focused and as a consequence scattered off the beam pipe or other object near the entrance. A similar device was placed several feet below and downstream of the target. This was called the good spill monitor; a large signal in this scintillator means that the beam was scattering off the target. This signal was also sent to an ADC and to

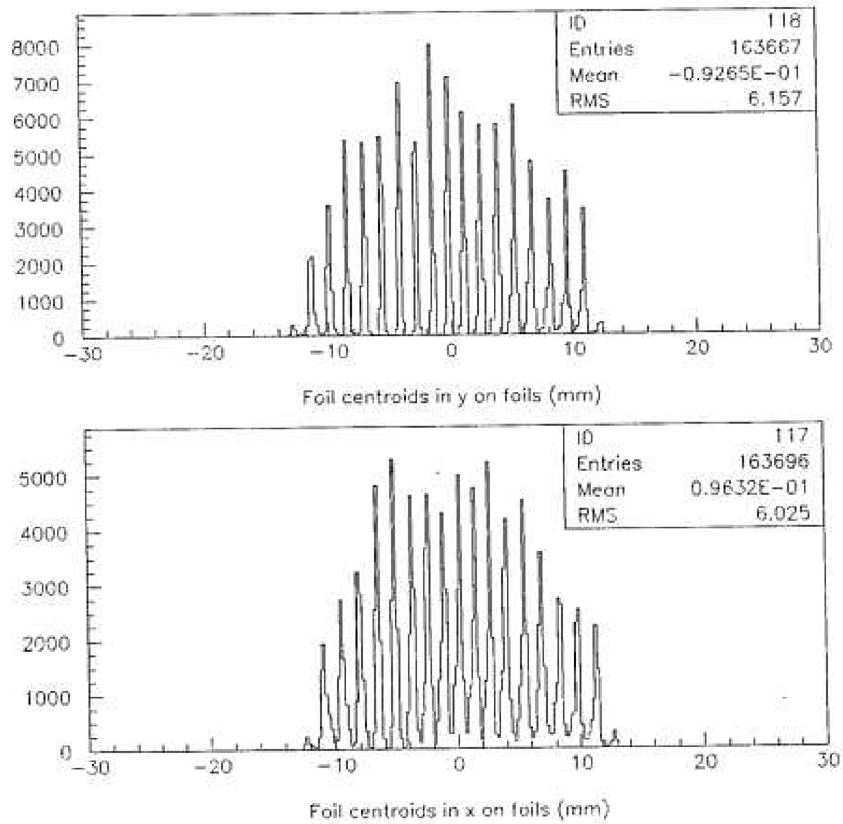

Figure 3.4: Typical beam positions in x and y on the foil array.

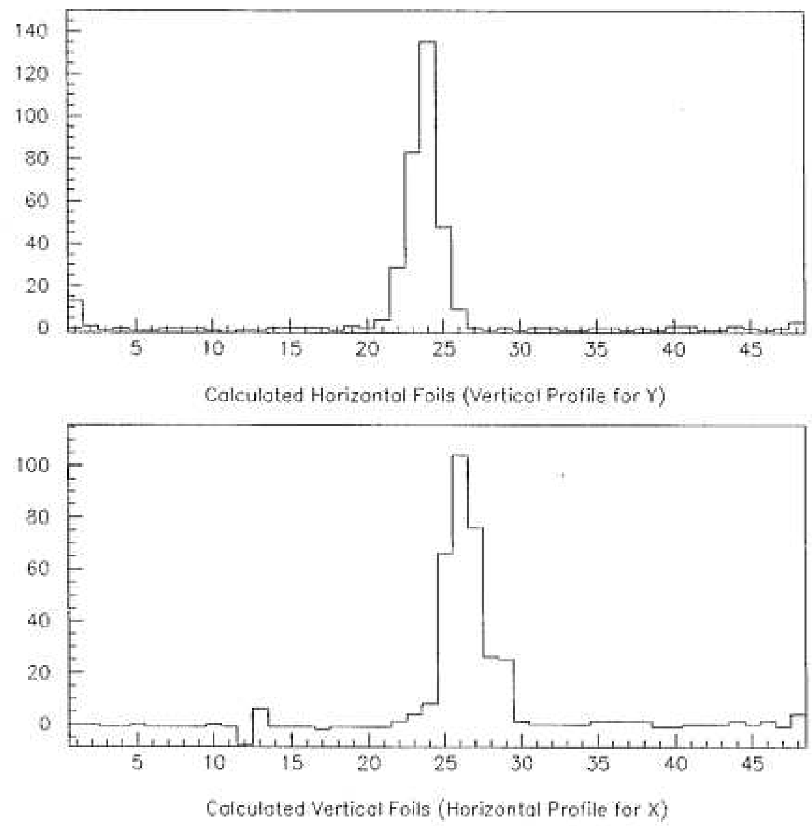

Figure 3.5: Typical single spill beam profile measurement in x and y on the foil array.

the same oscilloscope in the counting house. The oscilloscope display served as the main visual indicator of the beam quality.

Two toroid devices were used to measure the charge of the incident beam. For historical reasons, these were labeled toroid 2 and toroid 3 and were located 9.1 m and 5.6 m upstream of the target, respectively. Each toroid consisted of an iron ring with a coil of wire wound around it. The coil was connected to an RC circuit and preamplifier which produced a damped oscillating voltage with amplitude proportional to the amount of beam charge in a single beam pulse. This signal was transmitted to the counting house into an amplifier with switchable gain and then read into an ADC. The gain setting for E143 was 100 which is appropriate for spills with 1 - 5 $\times 10^9$ electrons.

The ADC value was converted into giga-electrons/pulse using a constant obtained from calibration. The calibration procedure consisted of sending a precisely determined amount of charge through a wire passing through the toroid monitors. Calibration runs were conducted periodically during the experiment. An analysis of the calibration data for different test currents found a slight current dependence with the measurement becoming more accurate at higher currents. The systematic error assigned to the measured charge ranges from 0.4% to 1.0% in the range of currents used for E143.

3.2.4 Chicane System

When transverse alignment of the target polarization was need, the target was rotated by 90° about the vertical axis. In this configuration, the magnetic field used to polarize the target is perpendicular to the beam axis. It is strong enough to deflect the beam away from the target and also prevent the beam from reaching the beam dump. In addition, the magnetic field rotates the spin of the incoming electrons so that they are no longer longitudinally aligned with the beam. To compensate for these effects, four magnets were set up in a chicane system. Three magnets were used in front of the target to bend the beam down and back up again. The target's magnetic field deflected the beam downward again. A final magnet after the target bent the beam back into the alignment with the beam dump. However, the outgoing beam was shifted down relative to the beam spot on the target and thus the reading of the foil array had to be corrected. The layout of the chicane system is shown in Figure 3.6.

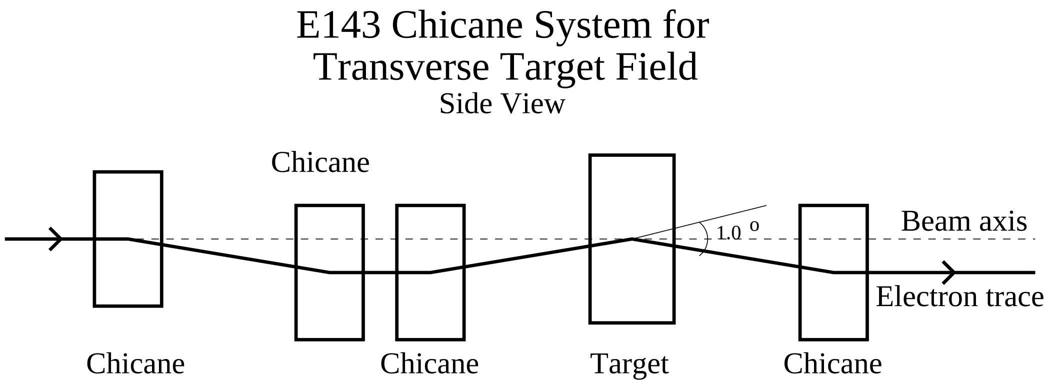

Figure 3.6: Schematic view of the chicane system. The system corrected for the effects of the target magnetic field when the target was in perpendicular polarization mode.

3.3 Møller Polarimetry

The beam polarization was measured on a daily basis by two independent Møller polarimeters. Møller scattering is electron-electron elastic scattering. When both electrons are polarized and relativistic, the cross section is [42]

$$\begin{aligned} \frac{d\sigma}{d\Omega} &= \frac{\alpha^2}{s}\frac{(3+\cos^2\theta)^2}{\sin^4\theta}\Big[1 - P_z^B P_z^T A_z(\theta) \\ &\qquad - P_t^B P_t^T A_t(\theta)\cos(2\phi - \phi_B - \phi_T)\Big] \end{aligned} \tag{3.5}$$

where s is the center-of-mass (cm) energy; θ is the cm-frame scattering angle; ϕ is the azimuth of the scattered electron; P_z^B, P_z^T are the longitudinal polarizations of the beam and target, respectively; P_t^B, P_t^T are the transverse polarizations; ϕ_B, ϕ_T are the azimuths of the transverse polarization vectors; and $A_z(\theta)$ and $A_t(\theta)$ are the longitudinal and transverse asymmetry functions

$$\begin{aligned} A_z(\theta) &= \frac{(7+\cos^2\theta)\sin^2\theta)}{(3+\cos^2\theta)^2} \\ A_t(\theta) &= \frac{\sin^4\theta}{(3+\cos^2\theta)^2}. \end{aligned} \tag{3.6}$$

For E143, the beam and Møller target were always longitudinally polarized so that Eq. 3.5 reduces to

$$\frac{d\sigma}{d\Omega} = \frac{\alpha^2}{s} \frac{(3 + \cos^2\theta)^2}{\sin^4\theta} \left[1 - P_z^B P_z^T A_z(\theta)\right] \tag{3.7}$$

By counting the scattered electrons as a function of θ for the case of equal but oppositely signed P_z^B, one can construct the a measured asymmetry defined as

$$A_{meas} = \frac{\sigma^{\uparrow\downarrow} - \sigma^{\uparrow\uparrow}}{\sigma^{\uparrow\downarrow} + \sigma^{\uparrow\uparrow}} = P_z^B P_z^T A_z(\theta) \tag{3.8}$$

where $\sigma^{\uparrow\uparrow}$ ($\sigma^{\uparrow\downarrow}$) is the cross section for spins aligned (anti-aligned). This asymmetry is a maximum when $\theta = 90°$.

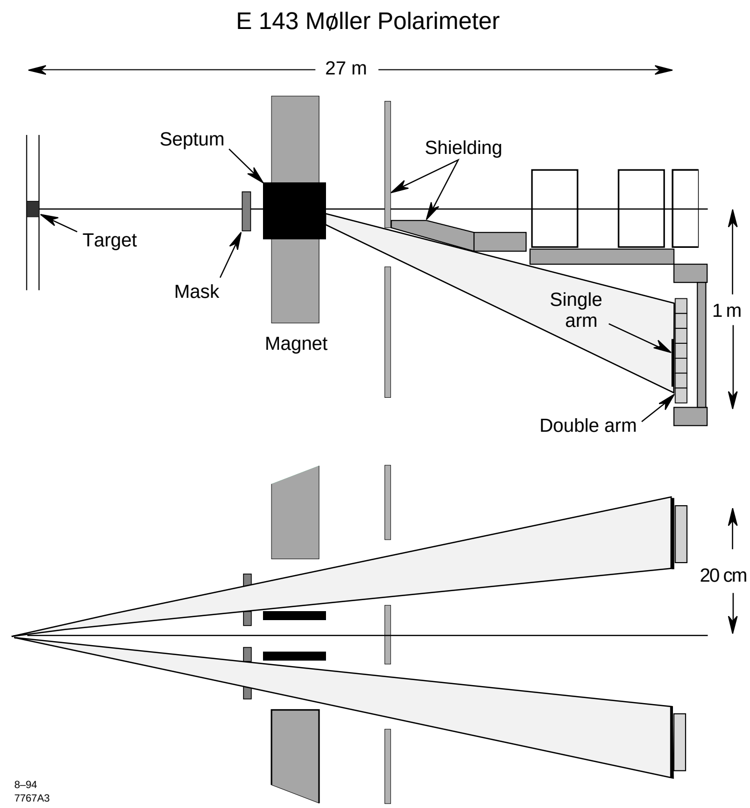

Figure 3.7: E143 Møller Polarimeter view from above (top) and side (bottom).

The E143 Møller polarimeter, as shown in Figure 3.7, consisted of a polarized electron target, a mask to select scattered electrons in the vertical plane, an analyzing magnet, and both single and double arm detectors. The targets used were six thin ferromagnetic foils (49% Fe, 49% Co, 2% Va) of varying thickness. Polarization is easiest to produced along the plane of the foil. A 100 Gauss magnetizing field was applied along the beam direction to magnetically saturate the foil. At saturation, an average of two excess electron spins per atom are aligned with the external field which results in a target polarization of roughly $\frac{2}{26}$.

The target foil polarizations were 0.0816 for the 20 μm foils and 0.0827 for the 30, 40, and 154 μm foils[43]. These polarizations were determined by measuring an integrated induced voltage in a pick up coil of 500 turns when the the magnetic field was swept from -100 to +100 Gauss. Two measurements are made: one with the foil inside the coils and one without the foil. The difference in integrated induced voltages is directly proportional to the bulk magnetization of the foil which in turn is directly proportional to the foil polarization at saturation. These measurements have a relative systematic error of 1.7%.

To make beam steering and target thickness calculations easier, the foil was placed at a 20° angle with respect to the beam axis. This reduced the effective target polarization by a factor of $\cos 20°$.

A collimator downstream of the target (see Figure 3.8) served as a mask to select scattering angles transverse to the bend plane of the dipole magnet which followed. The collimator had two wedge shaped apertures of constant ϕ acceptance (0.2 radian top, 0.22 radian bottom) and good acceptance over the entire double arm detector acceptance of $\theta = 70° - 110°$ in the cm-frame. The electrons were momentum analyzed by the dipole magnet with a $\int B\, dl$ of 21 kG-m. A large 3.2 inch thick septum reduced the field seen by the unscattered and forward scattered beam inside the beampipe, thereby minimizing backgrounds in the detectors. Since Møller scattering is elastic, the angle and momentum of the scattered electrons were correlated, resulting in a spatial x-y correlation after the magnet. An additional shielding wall with appropriate aperture for good Møller scattered electrons was placed after the magnet for additional background reduction.

Figure 3.9 shows the position of the single and double arm detectors relative to the incident scattered electrons. The single arm detector consisted of independent top and bottom parts each containing 4 silicon pad detectors 12 channels wide. These detectors operated in parallel behind 3 radiation lengths of lead which absorbed the soft photon background. Each pad detector accepted a specific momentum range and its channels measured the spread away from the selected scattering plane. The pads were large enough so that the Møller electrons were mostly con-

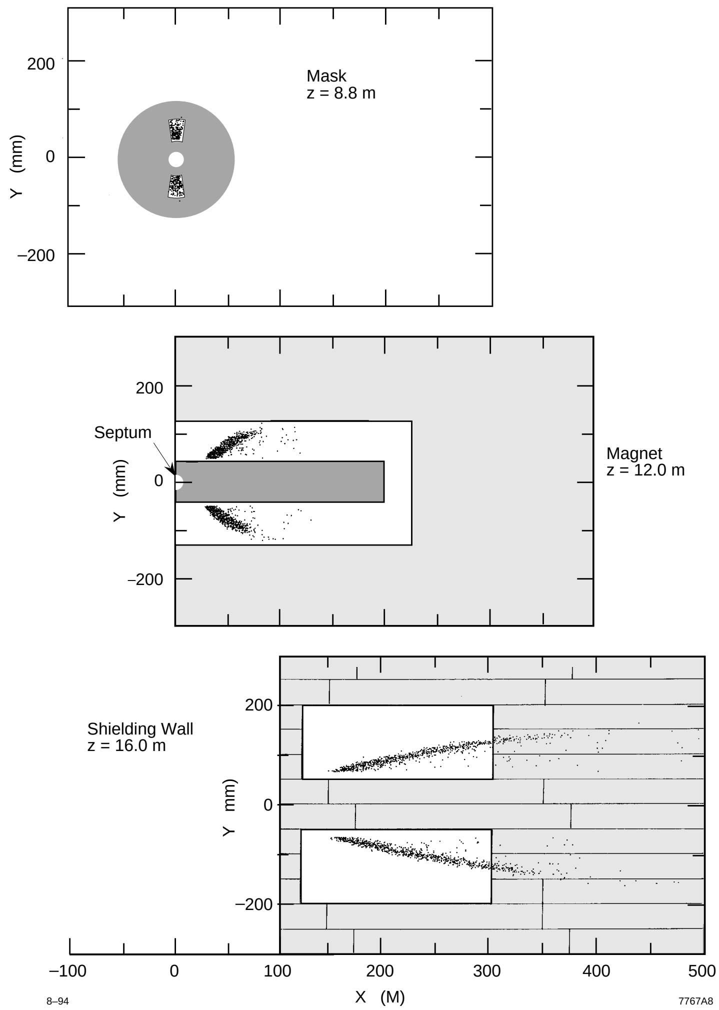

Figure 3.8: Spatial distribution of Møller scattered electrons at the mask (top), magnet exit (middle) and downstream momentum slit (bottom).

tained in only two channels. The signal in each channel was integrated over the entire beam pulse and recorded along with the sign of the beam polarization. At the end of the run, the average signal was calculated separately for the two beam helicities and an unpolarized signal and an asymmetry computed separately for each pad. A background was fit to the wings of the distribution and subtracted from the region of the Møller peak. The beam polarization was then calculated from the observed asymmetry using Equation 3.8 at the θ corresponding to the pad detector.

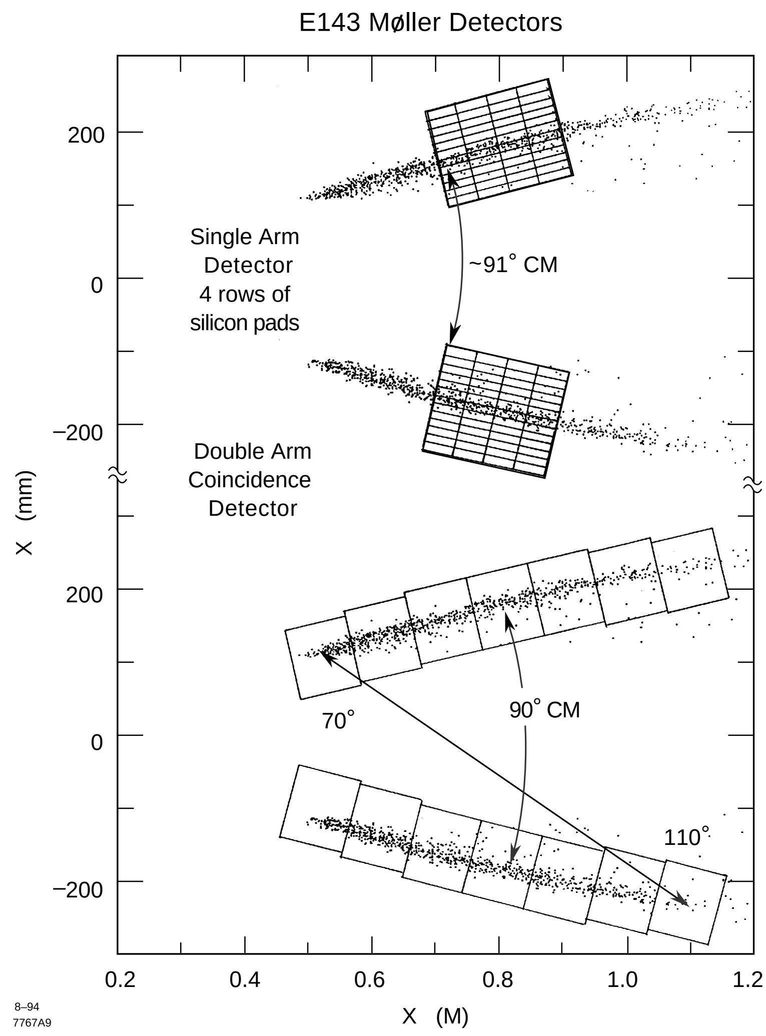

Figure 3.9: Single and double arm detector positions relative to the Møller scattered electrons.

The double arm detector consisted of an array of seven lead-glass 4×4 inch blocks located behind each single arm detector. Both electrons from the Møller scattering event were detected by looking for a time coincidence between appropriate pairs of top and bottom blocks. The time resolution was close to 1 nsec yielding sharp coincidence peaks with negligible backgrounds. The observed asymmetry only needed small corrections for deadtime and acceptance. The results of the single and double arm measurements were in good agreement. Since the systematic error in the double arm measurement was considerably smaller than that of the single arm measurement, the results of the double arm detector were used for analysis.

It was recently noted by Levchuk [44] that the intrinsic momenta of the target electrons can have a significant effect on the measured asymmetry. Inner shell electrons have momenta around 100 keV/c which is small compared the beam energy of several GeV but not small compared to the electron rest mass and can therefore alter the scattering angle by up to 10%. For the E143 foils, the polarized electrons are all from the 3d (M) shell which have a mean momenta around 2-10 keV/c. For the single arm detector, corrections for target momenta effects were on the order of 3%. For the double arm detector, which is had less spatial resolution and therefore was less sensitive to momenta smearing, the corrections were $\leq 1\%$.

The beam polarization was found to vary with the quantum efficiency (QE) of the SLAC polarized source (see Section 3.1). These measurements were fitted to an arbitrary functional form so that the beam polarization could be interpolated between Møller measurements from the more frequent measurements of QE as shown in Figure 3.10. Unfortunately, the spread in daily measurements about the fitted function was somewhat larger than could be explained by statistics, either due to systematic errors in the Møller measurement or to nonreproducibility in the P_z^B versus QE behavior of the beam. Including all these effects and the uncertainty in the foil polarization, the overall systematic error on the beam polarization measurements was calculated to be 2%[45].

During the 9.7 GeV running, no double arm Møller measurements were made and the single arm measurements have yet to be analyzed. It is the opinion of the Møller group that use of the P_z^B versus QE function fit is sufficient for analysis of the 9.7 GeV data and that the systematic error should be increased to 3%.

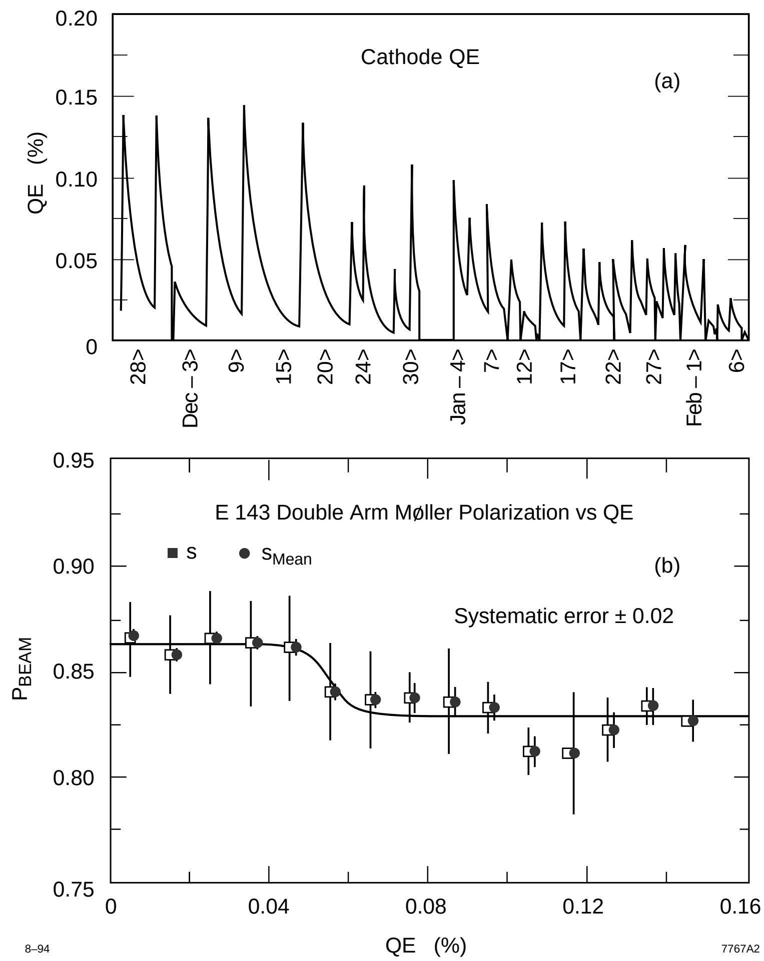

Figure 3.10: (a) Quantum efficiency of the source photocathode during E143. (b) Beam polarization measured by the Møller polarimeter double arm detectors versus quantum efficiency.

3.4 Target

In order to obtain high statistics, a target capable of withstanding high incident beam intensity while maintaining a high and stable nucleon polarization is needed. A target system using solid $^{15}NH_3$ for proton data and solid $^{15}ND_3$ for deuteron data was chosen. Ammonia has demonstrated a high radiation resistance and high polarizability [46, 47]. The method of Dynamic Nuclear Polarization was used to polarize the targets. This target system was constructed, tested and operated by a group from the University of Virginia[48].

3.4.1 Target Polarization

Polarization of a target using Dynamic Nuclear Polarization (DNP) requires a small number of paramagnetic centers in the target which provide a mechanism to transfer spin to the nucleons. Depending on the material, these centers can be produced by doping through chemical or irradiation methods. When a magnetic field B is applied, the electron-nucleus system will have the following Hamiltonian:

$$H = -\vec{\mu_e} \cdot \vec{B} - \vec{\mu_N} \cdot \vec{B} + H_{int} \tag{3.9}$$

where μ_N is the magnetic moment of the nucleus and H_{int} is the spin interaction term.

For a spin-1/2 nucleus, which is the case for a single proton in hydrogen, the energy levels will be split into four possible states as shown in Figure 3.11. Without the H_{int} term, only the W_1 and W_4 transitions are allowed due to dipole selection rules but when the term is included, W_2 and W_3 transitions are possible but suppressed relative to W_1 by roughly 10^{-3} [49]. At thermal equilibrium, only the two lowest states are significantly populated, but due to the small energy difference, these two states are populated about equally resulting in a small nucleon polarization. This polarization, due to the Maxwell-Boltzmann distribution, is

$$P_{te} = \tanh(\mu B/(kT)) \tag{3.10}$$

where k is the Boltzmann constant and T is the material temperature. In a field of 5 Tesla and at a temperature of 1 K, this corresponds to a thermal polarization of 99.8% for electrons and only 0.511% for protons.

Microwaves can be tuned to a frequency that will drive the W_2 or W_3 transitions

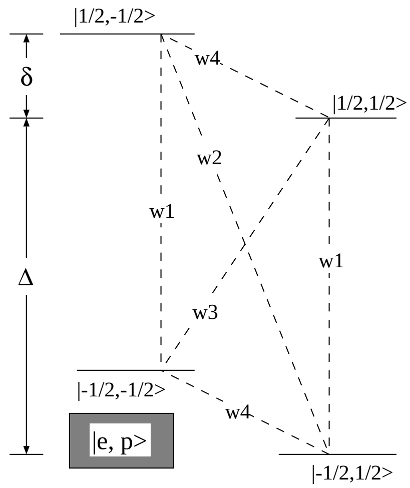

Figure 3.11: Energy level diagram for electron-proton system in magnetic field B.

and flip the spins of both the electron and proton. The key to dynamic nuclear polarization is that the high energy state decays predominantly by the W_1 transition. This is due to the fact that the spin-lattice coupling for nucleons is much weaker than that for electrons and therefore the nucleon spin relaxation time is much longer. The end result is that one of the lower energy states becomes more populated than the other.

The procedure only applies to nucleons near the paramagnetic centers. The rest of the material slowly becomes polarized by spin diffusion through the dipole-dipole interaction between neighboring nuclei. The outlying nucleons will tend to remain polarized longer since the paramagnetic centers also serve as the primary mechanism for nuclear spin relaxation. However, it is important to remember that the material will serve as a target in a beam of high energy electrons. The beam will produce new paramagnetic centers in the target material at 1 K and increase the available modes of nucleon spin relaxation. Eventually the number of centers will become too high, causing the polarization to fall below a usable level. Fortunately, these new paramagnetic centers can be removed by annealing. For this process, the beam is turned off and the target slowly heated to around 85 K for about 5 minutes. This allows most of the paramagnetic centers created at 1 K to recombine. The target can then be cooled back down to 1 K and polarization equal to or even larger than before can be obtained.

For the deuteron, the picture is slightly different but the principal is the same.

Since the deuteron is a spin 1 particle there are three deuteron substates. The magnetic moment of the deuteron is about 1/3 that of the proton and therefore interacts more weakly with the paramagnetic centers which results in a slower spin diffusion rate. Also, the deuteron has a non-zero quadrupole moment resulting in an additional energy splitting. This effect is very small and has no effect on the polarization mechanism.

3.4.2 Target Material

For E143, frozen $^{15}NH_3$ and $^{15}ND_3$ were used as target materials. When making the ammonia targets, ^{15}N was used instead of the more abundant ^{14}N because it has no polarizable neutron and the nuclear physics is easier to model. A correction for the unpaired, polarizable proton must be made to the data during analysis. Ammonia gas with ^{15}N purity better than 98% was slowly frozen inside a container immersed in liquid nitrogen [50]. The resulting solid piece of ammonia was crushed to 1-2 mm granules by pushing it through a wire mesh. These granules were measured at 77 K to have a density of 0.917 g/cm^3 for $^{15}NH_3$ and 1.056 g/cm^3 for $^{15}ND_3$, with a relative error of 1%.

To create the required paramagnetic centers, the method of pre-irradiation gave the best results. Pre-irradiation of different target samples was done at the MIT-Bates Laboratory (350 MeV), at HEPL on the Stanford campus (35 MeV), at the Saskatoon Accelerator (250 MeV), and at the Naval Postgraduate School in Monterey California (65 MeV). The pre-irradation process creates $^{15}NH_2^-$ ions by knocking out a proton. The extra electron remains localized in the lattice as long as the material is kept cold serving as a paramagnetic center.

The ammonia was then placed into a cell of a target holder and lowered into a cryostat of liquid ^{4}He. The nitrogen, helium and other target holder materials were in the acceptance of the spectrometers and thus served to dilute the measured asymmetry. To correct for this, one must know the thickness and density of each material species present to the beam. These quantities for all materials except the ammonia were easy to obtain from well documented values and are listed in Table 3.1.

The thickness (or volume packing fraction) of the ammonia target was measured in three different ways. The first method was to weigh the granules while suspended over a liquid nitrogen dewar. For the second method, the event rates for data taken while the target was in the beam were analyzed. Since the unpolarized cross section

Material	Thickness (cm)	Density (g/cm^3)
^{4}He	0.37	0.145
Aluminium endcaps	0.00762	2.70
Copper in NMR coil	0.00673	8.96
Nickel in NMR coil	0.00289	8.75
Titanium windows in tail	0.00712	4.54
Aluminum win. in LN_2 shield	0.00508	2.70
Aluminum entrance win. in cryostat	0.00762	2.70
Aluminum exit win. in cryostat	0.01016	2.70
Aluminum beampipe exit win.	0.00762	2.70
Aluminum win. in downstream gas bag	0.00508	2.70

Table 3.1: Thickness and density for unpolarizable materials in acceptance of E143 spectrometers.

of ammonia is comparatively well known, the amount of target material could be extracted. In the third method, a used target was placed in the helium cryostat with the same structural design as the actual target system. The attenuation of a collimated 60 keV X-ray beam passing through the target was then measured. From known absorption coefficients, the target thickness could be calculated. The event rate method was found to have the lowest error and was used for the final analysis.

3.4.3 Target Apparatus

The target system consisted of a ^{4}He evaporation refrigerator, a large pumping system, a strong magnetic field and a high power microwave tube as shown in Figure 3.12. The magnetic field was produced by a pair of superconduction Helmholtz coils which provided a cylindrically symmetric field with a maximum central field of 5.1 Tesla. The microwave tube used for polarization was limited to a narrow band around 136.5 GHz and thus the magnets were set to 4.87 Tesla to match. The uniformity of the field over the central 3 cm region containing the target was better than 10^{-4} Tesla. The pumping system maintained a target temperature of 1.06 K at a heat load of 1.3 Watts from the beam and microwaves combined. The entire target assembly could be rotated 90° to allow for running with the target polarization either longitudinal or transverse to the incident beam direction. In either configuration, the sign of the field could be switched to point the target spins in the opposite direction. This was done periodically in order to minimize systematic error.

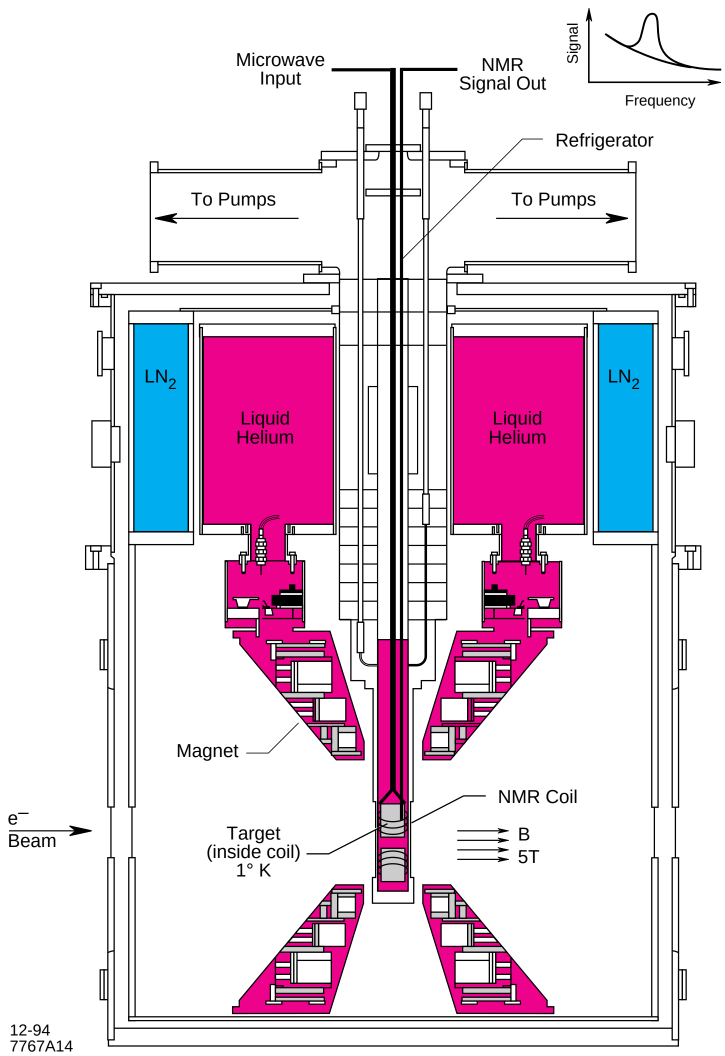

Figure 3.12: E143 polarized target cross-section view.

To avoid entry into the End Station, a target insert of roughly 1 meter in length was employed containing cells for each type of target. A total of three such inserts were made. Each insert had three cylindrical cells 2.5 cm in diameter and 3.0 cm in length machined from Torlon. Torlon has a high resistance to radiation and a low free proton density. One cell contained the $^{15}ND_3$, another the $^{15}NH_3$ and the third was left empty. Below these cells was an additional cell of carbon or aluminum chosen to be roughly equal in radiation length with the ammonia target. This extra cell was used for spectrometer calibration. Each cell was fitted with removable endcaps made from 0.015 inch thick aluminum. The $^{15}ND_3$ was placed in the top cell close to the horn of the microwave waveguide in order to receive the maximum power.

Two NMR pickup coils made of 70/30 copper/nickel wire of diameter 0.5 mm were placed in each of the top three cells. The first coil was a short straight vertical piece traversing the entire diameter of the cell and was used to measure the proton polarization in $^{15}NH_3$ and the residual proton polarization in $^{15}ND_3$. The second coil was a four turn helical coil of diameter near 2 cm and was used to measure the deuteron polarization in $^{15}ND_3$ and the ^{15}N polarization in both materials. NMR signals were carried to the target cells by a beryllium/copper cable of diameter 0.085 inch.

3.4.4 Target Measurement

The target material and embedded coil served as the inductor in a tuned LRC circuit which made up the NMR system used to measure the target polarization. The frequency of the circuit was swept back and forth through the resonance of the target nuclei by roughly ± 300 kHz. At resonance, the power absorbed by the material is proportional to its polarization. Typical signals from the system for both target types are shown in Figure 3.13. The area under the central bump is proportional to the polarization. Notice that the deuteron signal shows two peaks due to the quadrupole splitting.

To extract the polarization from the NMR signal taken during DNP, the signal must be calibrated to that of a known polarization. The obvious reference to use is a signal measured at thermal equilibrium (TE) at which point the target has a known polarization from the Maxwell-Boltzmann distribution. For protons, this polarization was given above by Equation 3.10. For deuterons, which are spin-1 and thus have three possible spin states, the polarization is given by

$$P_{TE} = \frac{4\tanh(\frac{\mu B}{2kT})}{3+\tanh^2(\frac{\mu B}{2kT})}. \tag{3.11}$$

At a magnetic field of 4.87 Tesla and temperature of 1.5 K, the TE polarizations are 0.33% for protons and 0.068% for deuterons. To reduce the error due to the low signal to noise ratio for such small polarizations, several measurements were made. A typical proton calibration consisted of 25 measurements of 500 sweeps each. A typical deuteron calibration consisted of 25 measurements of 1000 sweeps each. The

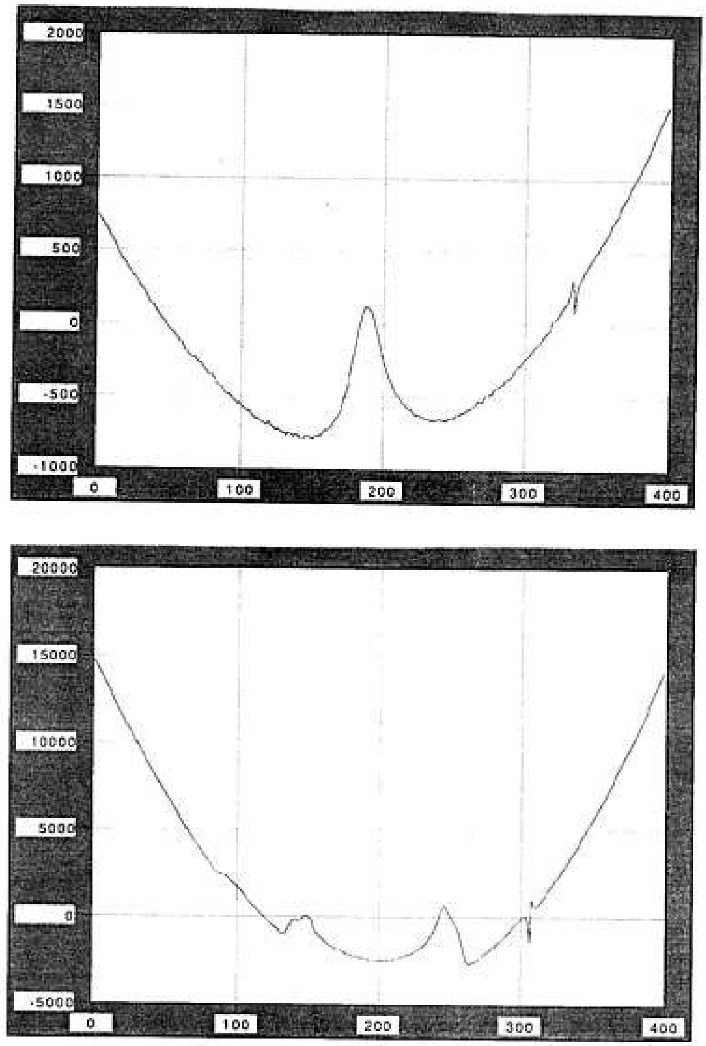

Figure 3.13: Power absorbed by $^{15}NH_3$ (above) and $^{15}ND_3$ (below) targets as a function of frequency during a typical NMR sweep.

peak area, A_{TE} was obtained by subtraction of the baseline signal. A baseline curve was obtained from a measurement taken with the magnet slightly shifted (1%) off the nominal field so the sweep did not pass through resonance.

The polarization of the target during DNP could then be calculated using

$$P_{DNP} = \frac{P_{TE}}{A_{TE}} A_{DNP} \tag{3.12}$$

where A_{DNP} is the peak area measured during DNP. A TE calibration was done approximately 5 time during each target's lifetime typically just after a target was load and after each anneal. The statistical error in the calibration constants were 0.2% and 3.0% for the protons and deuterons respectively. Typical plots showing the performance of the target polarization over time are shown in Figure 3.14.

3.4.5 Beam Heating Effect

When the beam passes through the target, its raises the temperature of the ammonia beads it hits causing the polarization at that point to drop. Rastering the beam greatly reduces this depolarization effect by allowing the affected spot some time to cool back down. However, the raster time is not long enough to allow a return to the original temperature and an equilibrium polarization slightly lower than that achieved without incident beam results in the rastered region. This rastered region does not cover the entire target so that the polarization of the target outside the beam path remains higher than the area inside the rastered region. When an NMR measurement of the target's polarization is taken, it is the average polarization of the entire target that is measured, which is slightly lower than that really seen by the beam.

A model of the target depolarization was developed by the target group [51] using the thermal properties of the target material, the size of the rastered region, and location of the NMR coils. The actual polarization seen by the beam, P_t, was found to be related to the measured polarization, P_{meas}, and beam current, I_b, according to the formula

$$P_t = P_{meas}(1 - C_{heat}\frac{I_b}{4 \times 10^9 \mathrm{e}^-/\mathrm{pulse}})$$

where C_{heat} is 0.0081 for $^{15}NH_3$ and 0.0197 for $^{15}ND_3$. For the 9.71 GeV runs, this correction reduced the measured polarization values by approximately 2% relative

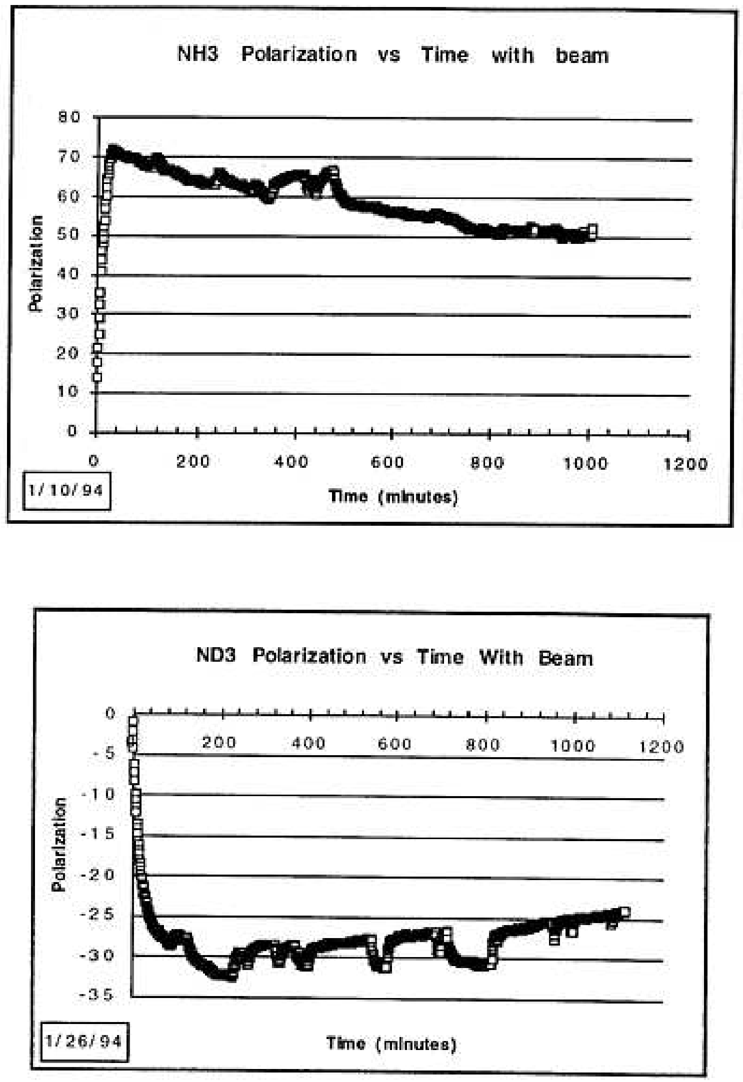

Figure 3.14: Plots of $^{15}NH_3$ (top) and $^{15}ND_3$ (bottom) target polarization performance as a function of time.

for the proton target and 5% for the deuteron target.

3.5 Spectrometers

The scattered electrons were measured by two independent spectrometer systems centered on the scattering angles 4.5° (SP4) and 7.0° (SP7), as shown in Figure 3.15. These spectrometers were initially setup for the E142 experiment and slightly modified for E143. Although the arrangement was not ideal for the kinematics of E143, the setup was considered sufficient for the experimental goals and not worth the complications needed to reassemble the spectrometers at different angles.

The cross sections measured were very small, on the order of 10^{-32} cm^2/sr/GeV. The asymmetries were also small, on the order of 10^{-3}. The kinematic range each spectrometer was set to cover depended on the beam energy. The E' (or x) range of each spectrometer was specifically designed to overlap in a significant part of their coverage. Therefore, data from the 2 spectrometers provided some information on the Q^2 dependence of the structure functions. To compare our results with most predictions in the scaling region, we must evolve to a common Q^2.

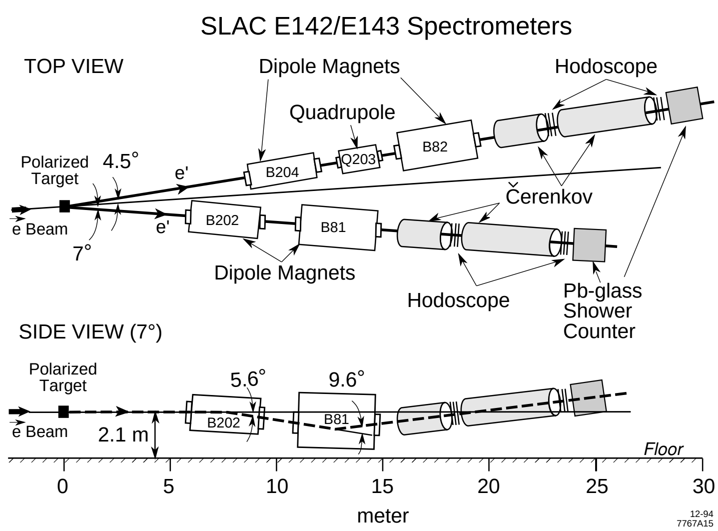

Figure 3.15: E143 spectrometers centered on the scattering angles 4.5° and 7.0°.

3.5.1 Magnets and Collimators

Both spectrometers contained two dipole magnets which were configured to bend the scattered electrons downward and then upward again. This "two bounce" system has two main advantages over designs with a single bend direction. First, the solid angle of the detector remains relatively constant over the accepted momentum range. Second, the large photon background due to bremsstrahlung radiation, radiative Møller scattering and the decay of π mesons is almost completely rejected. The probability of a photon scattering twice inside the magnet walls and reaching the detector package is very small. The main disadvantage of the "two bounce" design is that the total dispersion in the bend plane is small, resulting in a loss of momentum resolution. For the main goal of E143 to measure asymmetries in the deep-inelastic region, this was not a large problem.

With identical acceptance, the ratio of incident electron rates between the two spectrometers is roughly the ratio of the Mott scattering cross sections at their respective angles, which goes roughly as the inverse ratio of the angles to the fourth power, $(7.5/4.0)^4 \approx 12$. It would be inefficient to run SP4 at its maximum allowed rate and SP7 at only $\frac{1}{12}$th of that. It was therefore decided to reduce the solid angle acceptance of SP4. The settings used were ± 14 mr in the bend plane for both spectrometers and ± 5 mr in the non-bend plane for SP4 and ± 10 mr for SP7. Because of the smaller acceptance in SP4, a quadrapole was placed between its dipole magnets to defocus the scattered particles in the non-bend plane onto a larger detector area. This quadrupole magnets was essentially the only difference between the two spectrometers. Typical ray trace plots through the magnets for both sets of magnets are shown in Figures 3.16 and 3.17.

3.5.2 Čerenkov Detectors

Each spectrometer had two threshold-type Čerenkov detectors to identify electrons within a large background of pions. Čerenkov light is emitted when a charged particle passes through a medium at a higher velocity than the speed of light in that medium. This light is emitted within a cone of angle θ defined by

$$\cos\theta = \frac{1}{n\beta} \tag{3.13}$$

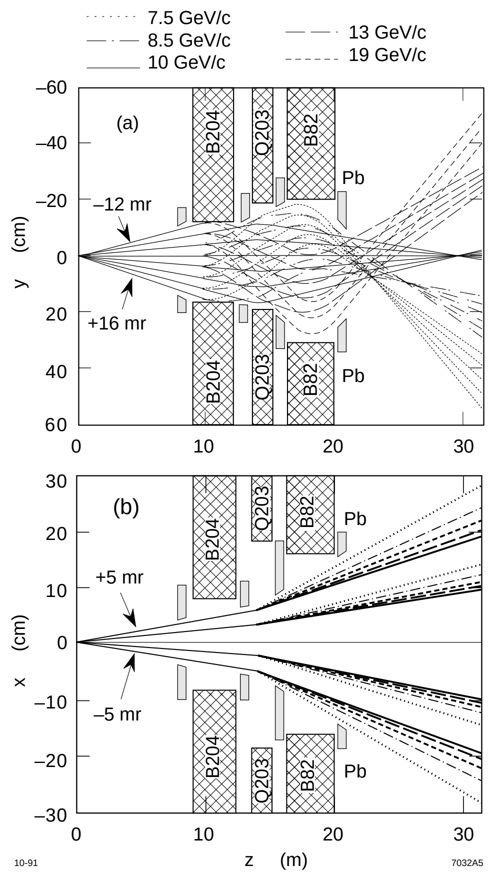

Figure 3.16: Trajectories of electrons with different momenta and scattering angles through magnets in SP4. Lines are drawn relative to a false geometry in which the central ray 10 GeV is a straight line.

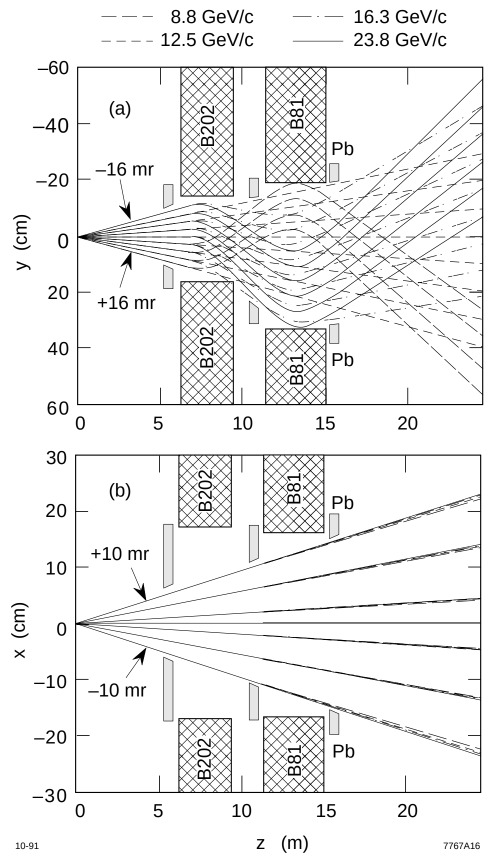

Figure 3.17: Trajectories of electrons with different momenta and scattering angles through magnets in SP7. Lines are drawn relative to a false geometry in which the central ray 12.5 GeV is a straight line.

where n is the refractive index of the medium and $\beta = v/c$ is the velocity of the particle with respect to the velocity of light. If $n\beta$ is less than 1, then no Čerenkov light is emitted. The key to selecting electrons while rejecting pions is to find a medium that produces light only for electrons. This is possible since the more massive pion will always have a slightly smaller β value than the electron in the same energy range. For example, at 13 GeV, a pion will have $\beta = 0.9999424$ while an electron will have a β of essentially unity. A Čerenkov detector containing a gas with $n = 1.000058$ will not produce a signal for any pion less than 13 GeV but will produce a signal for any electron in the GeV range.

To achieve indices of refractions with $n - 1 \approx 10^{-5}$, one must use a gas medium. The first Čerenkov detector (C1) in each spectrometer was a two meter tank filled with nitrogen gas at 6.3 PSI which corresponded to a pion threshold of 9 GeV. The second Čerenkov detector (C2) was a four meter tank filled with nitrogen at 3.0 PSI corresponding to a pion threshold of 13 GeV. Nitrogen was used because of its ability to transmit light down to 150 nm allowing for a larger number of detectable photons. Two mirrors were used in C1 and three were used in C2 to focus the emitted light on the phototube (Hamamatsu R1584-01) which was mounted internally on the side of each tank. The mirrors were made from 3 mm thick glass coated with 80 nm of Al followed by a protective coating of 30 nm MgF_2. Reflectivity ranged was around 80% at 160 nm. The phototubes were coated with a wavelength shifter (2430 nm thickness of para-terphenyl) to improve its efficiency for UV light. Each detector was made of an aluminum cylinder with 1.27 cm thick side walls and 1 mm thin end windows. The thin windows were desired in order to minimize δ-ray production and multiple scattering.

Higher pion thresholds, though desirable, were not achievable since the high voltage base of the phototube was prone to arc at pressures below 3 PSI. Having two detectors with different thresholds allowed us to better understand the pion background and to calibrate the shower counter.

3.5.3 Hodoscopes

Seven planes of hodoscopes were included in each spectrometer for particle tracking. From a particle's track, one can calculate its momentum and scattering angle using the proper reverse matrix elements corresponding to the magnet settings. The hodoscopes labeled in order of their location along the particle path were H1U, H2X, H3Y, H4Y, H5X, H6Y, and H7U as shown in Figure 3.18. The letters X, Y, and U indicate the the hodoscope measured the position of the track along the x, y and diagonal axes.

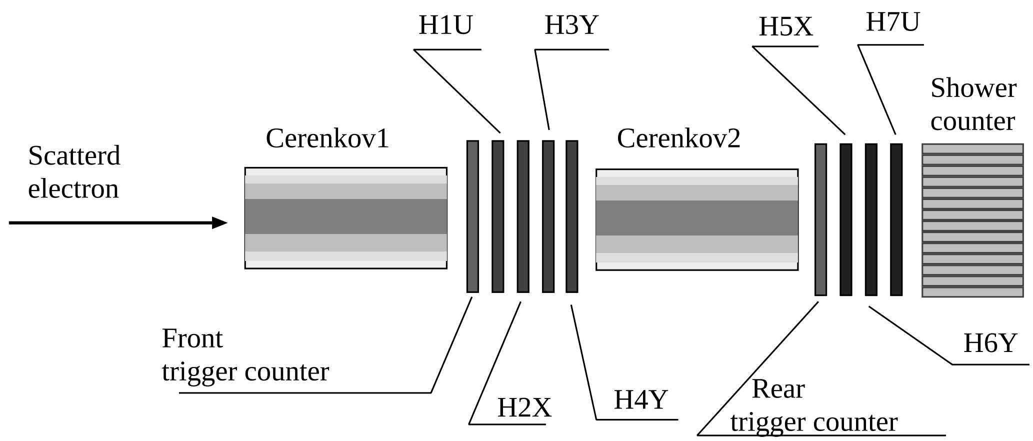

Figure 3.18: Location of the hodoscope and trigger planes relative to the Čerenkov tanks and shower counter.

A hodoscope plane consisted of long, flat scintillation counters called fingers. Several fingers were placed parallel to each other transverse to the axis the hodoscope was intended to measure. Every finger had a 2/3 overlap in its width with its neighbors as shown in Figure 3.19. This two layer configuration improved the spatial resolution of the hodoscope through use of time coincidence between overlapping fingers. A schematic of the front layer of H1U is shown in Figure 3.20.

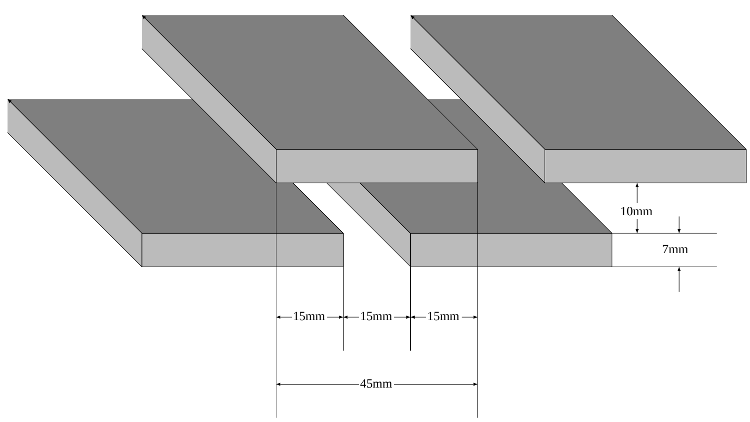

Figure 3.19: Hodoscope finger overlap (dimensions given are for H1U).

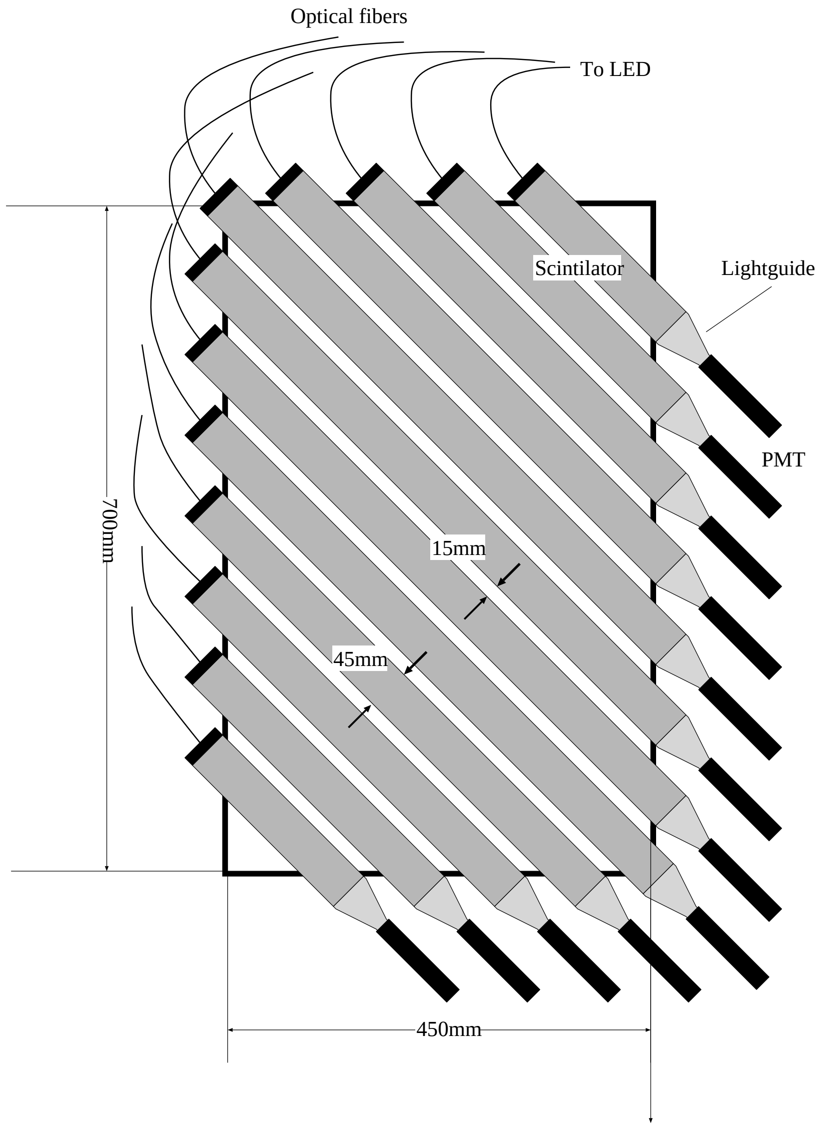

Figure 3.20: Schematic of finger layout for front half of H1U.

Tables 3.2, 3.3, and 3.4 show the exact finger dimensions for each hodoscope. Because of their higher spatial resolution, planes H2X, H3Y, H5X and H6Y dominated the tracking while the other three planes mainly served to reject backgrounds. Each hodoscope was connected by a lightguide to a phototube at only one end. A optical fiber was connected to the other end which led to an LED used for verifying that a finger was alive.

Plane	width(mm)	length(mm)	thickness(mm)	# of fingers
H1U	45	Various	6.2	25
H2X	20	590	6.2	34
H3Y	30	430	6.2	31
H4Y	47.6	355.6	6.2	20
H5X	30	1070	6.2	27
H6Y	30	510	6.2	55
H7U	75	Various	10	21
Total finger number				213

Table 3.2: Hodoscope dimensions for the 4.5° spectrometer.

Plane	width(mm)	length(mm)	thickness(mm)	# of fingers
H1U	45	Various	6.2	25
H2X	30	690	6.2	23
H3Y	30	430	6.2	36
H4Y	47.6	482.6	6.2	20
H5X	30	1070	6.2	27
H6Y	30	510	6.2	55
H7U	75	Various	10	21
Total finger number				207

Table 3.3: Hodoscope dimensions for the 7° spectrometer

In addition to the seven hodoscope planes, two additional large scintillator trigger counters were placed in each spectrometer for use in forming on-line triggers. The front trigger counter, S1, was located immediately before H1U. It covered an area of 450×700 mm. Two phototubes were connected to the scintillator both on top

H1U			H7U	
length(mm)	# of fingers		length(mm)	number
200	2		200	2
260	2		300	2
320	2		400	2
380	2		500	2
440	2		600	2
500	2		700	2
560	2		760	2
620	2		820	7
680	2			
720	1			
740	6			
Total	25		Total	21

Table 3.4: U-hodoscope dimensions.

and bottom. The rear trigger counter, S2, was located immediately before H5X. It covered an area of 550×1100 mm and had only one phototube on the top and bottom. The active area for both trigger counters was larger than the spectrometer acceptance.

3.5.4 Shower counters

The last element in each spectrometer was a lead glass shower counter made from two hundred 6.2×6.2×75 cm blocks stacked 10 wide and 20 high as shown in Figure 3.21. The radiation length was 3.17 cm and the refractive index was 1.58. Incident high energy electrons produce an electromagnetic shower by bremsstrahlung and e^+/e^- pair creation. The electrons and positrons thus produced emit Čerenkov radiation in an amount proportional to the incident energy. Collection of this light through phototubes at the back of the block allows one to reconstruct the particles energy. An incident electron deposits on average more than 99.9% of its energy within a cluster of 9 adjacent blocks. Hadronic particles such as pions lose most of their energy through ionization and thus produce a smaller signal than electrons at the same energy. This provides a powerful method to reject hadronic particles. The entry position of an electron could be roughly determined by energy weighting the position of contributing lead glass blocks. This position provided a way to match a hodoscope track with a cluster.

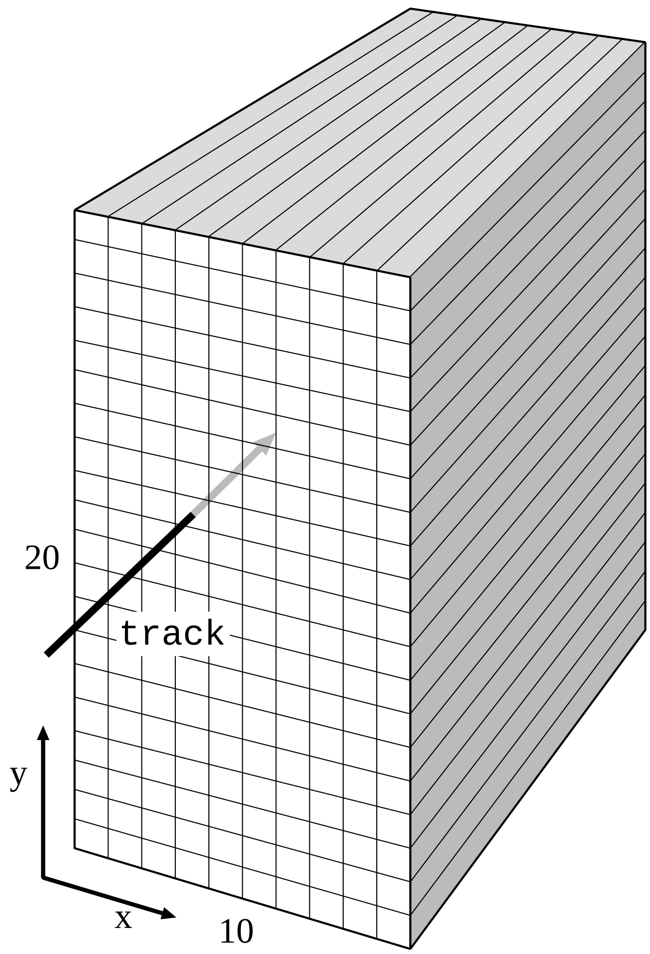

Figure 3.21: Schematic of segmented lead glass shower counter.

The shower counter was calibrated by comparing the momentum, P, of the matching track to the energy deposited, E for events with strict requirements for electron purity. For extremely relativistic electrons, the ratio $E/P = 1$. The measured E/P values were accumulated over a series of runs and their distribution fit to a Gaussian. A correction factor was determined for each block to shift the value to unity and the data re-analyzed. This procedure was repeated until the mean of the E/P distribution converged to 1. Appropriate adjustments were also made to calibrate edge and corner blocks to account for the undetected energy fraction. Separate calibration constants were determined for each target and field orientation.

3.6 Electronics and Triggers

Reconstructing particle energy and tracks and doing electron selection required recording a large amount of timing and pulse height data from each spectrometer's TDCs and ADCs. Several different triggers were established to pick out events containing useful information. Due to the constraints on available electronics and data acquisition speed, the ADCs only recorded for up to the first four triggers of a spill and the TDCs were limited to up to 16 triggers per spill. We used 2277 LRS pipeline TDCs which had 32 channels per module. Each TDC was reset at the start of each spill on the leading edge of the BEAMGATE signal, **BG**. This signal was formed by a logical AND of the **A2N** signal that was sent to ESA by the Main Control Center (MCC) at the beginning of each spill and a computer ready/run signal. Each **A2N** arrived 2.5 μs before the actual electron spill and lasted for 6 μs. Since each spill was around 2.3 μs, this covered the entire spill and could be used as input to gate logic for all electronics and to interrupt the computer for data acquisition. Logic was also present to mimic a BEAMGATE signal for calibration and pedestal runs.

The analog output from each of the two hundred shower counter phototubes was split by a linear fan out into five equal signals. Four of these signals were connected individually to an ADC channel in one of four identical ADC crates. Each crate was separately gated by one of the allowed first four triggers. Long cables were used to produce a 200 ns delay on the signal to allow time for the trigger. The fifth signals from each block were summed together through fan-ins to form an output pulse proportional to the total energy deposited in the shower counter. This signal was then fanned out into the input of five discriminators set to different threshold levels. These were labeled **SH(VL)**, **SH(L)**, **SH(M)**, **SH(H)** and **SH(VH)** corresponding to very low, low, medium, high, and very high settings of the discriminator threshold. The output from these discriminators were used in the producing various triggers as will be described below and connected to counters for online monitoring of rates.

The output from each Čerenkov ADC was similarly split and sent to an ADC channel in each of the four crates and to four discriminators with different thresholds. For the first Čerenkov , the discriminator outputs were labeled **C1(L)**, **C1(M)**, **C1(H)** and **C1(VH)** corresponding to low, medium, high and very high threshold settings. Similarly, the second Čerenkov produced **C2(L)**, **C2(M)**, **C2(H)** and **C2(VH)** signals. Each discriminator output was split into several signals with one serving as input to a dedicated TDC and the others used as inputs to trigger logic and online rate counters.

The outputs of each phototube in the front trigger counter were added together and then sent to a discriminator. The back trigger counter was similarly connected to a discriminator. The output from the fingers of the H4Y hodoscope plane were added together and passed to a discriminator. The output from these three discriminators were then input into a AND-logic coincidence unit, the output of which was labeled **S**, the scintillator trigger. The **S** signal was the "loosest" trigger, occurring when almost any particle passed through the spectrometer. In order to prevent this trigger from overwhelming the electronics, it was fanned out into four scalers, **N1**, **N2**, **N3** and **N4**. This prescalers were set to various scale factors throughout the experiment depending on the condition of the particular run as shown in Table 3.5. For the entire experiment, **N1**, **N2** and **N3** were always set to identical values and **N4** was always set to a lower or equal value. The output signals of the prescalers were labeled **S/N1**, **S/N2**, **S/N3** and **S/N4** and used as inputs to trigger logic. They were also attached to counters to track their rates and determine if they needed adjustment. The **S** signal was also fanned out to two counters where one was gated on positive helicity spills and the other on negative helicity spills. These counters provided a very rough online indication of the raw asymmetry.

For every hodoscope plane, the signal for each finger was processed through a discriminator and input to a TDC. Each channel of the TDC was able to record up to 16 values with a resolution of 1 ns and a total dynamic range of 65 μs. Since the TDC was read and cleared at the end of each 2-3 μs spill, this range was more than enough. Each TDC was started by the BEAMGATE signal and gated by a signal called the HODOGATE signal, **HG**, which was a logical OR of $\mathbf{S/N4} \cdot \mathbf{C1(M)}$, $\mathbf{S/N4} \cdot \mathbf{C2(M)}$, $\mathbf{C1(L)} \cdot \mathbf{C2(L)}$, **S/N1**, **S/N2**, and **S/N3** followed by a logical AND with the BEAMGATE.

The TDCs were configured such that if more than 16 pulses entered a channel only the times of the last 16 were stored. Also, the TDCs from each crate were read into a single large compressed buffers such that only channels with non-zero values were recorded. This presented a problem since only the channel number and timing values were recorded with no specific indication of the module number. The module position could be inferred from the position of the data in the buffer, but when a module has few non-zero values, ambiguities arise. Therefore, channel 32 for each module was reserved for a marker pulse that always input a unique and definite time

	4° Spectrometer		7° Spectrometer	
Runs	N1=N2=N3	N4	N1=N2=N3	N4
774-818	8	1	8	1
819-887	16	1	8	1
890-965	24	4	8	1
971-1064	8	1	8	1
1065-1650	32	4	8	1
1659-1729	128	4	16	1
1738-1883	32	4	8	1
1884-1942	16	1	8	1
1946-2221	32	4	8	1
2222-2235	32	4	2	1
2236-2425	16	2	2	1
2427-2445	4	1	2	1
2455-2472	16	2	1	1
2473-2674	16	2	2	1
2575-2906	32	4	2	1
2907-2911	8	1	2	1
2916-2942	32	4	2	1
2943-2949	8	1	2	1
2953-3004	32	4	2	1
3005-3009	8	1	2	1
3010-3137	32	4	2	1
3148-3378	1	1	1	1

Table 3.5: Summary of scintillator trigger prescaler settings.

that could be used to identify the TDC module.

Another trigger called the PION-OR was formed by a logical OR of **S/N1**, $\mathbf{S/N2} \cdot \mathbf{SH}(\mathbf{VL})$ and $\mathbf{S/N3} \cdot \mathbf{SH}(\mathbf{L})$. However, since the **N1**, **N2** and **N3** prescalers were always equal, this signal was logically the same as **S/N1** and thus served as a trigger on any particle. However, for most kinematic regions measured in E143, pions dominated the rate so the label was still appropriate.

Four other triggers were formed by logical AND combinations of the shower, Čerenkov, and scintillator signals. These were collectively referred to as the efficiency triggers since they could be used to investigate detector efficiencies. A logical AND of **S/N4**, **C2(M)** and **SH(M)** formed the C1 efficiency trigger. A logical AND of **S/N4**, **C1(M)** and **SH(M)** formed the C2 efficiency trigger. A logical AND of **S/N4**, **C1(M)** and **C2(M)** formed the shower efficiency trigger. A logical AND of **SH(L)**, **C1(L)** and **C2(L)** formed the MAIN efficiency trigger. These four triggers along with the PION-OR trigger were combined in a logical OR to produce the MAIN-OR trigger (see Figure 3.22).

The MAIN-OR signal, followed by a logical AND with the BEAMGATE, served to gate the shower and Čerenkov ADCs in the four ADC crates. This was done through a custom-made module called the Saclay Trigger Divider (STD) which properly gated the correct crate on the first through fourth trigger and also sent signals to counters for tracking the number of each trigger and trigger overflows. The STD was reset by the leading edge of the BEAMGATE.

A CAMAC crate called the beam crate contained all the ADCs and counters for the beam monitoring devices discussed in Section 3.2.3 above. These device were gated solely by the BEAMGATE signal. The crate also contained a TDC for timing the beamgate and registers for obtaining the timeslot and the beam polarization. To keep the signals from the badspill and goodspill monitors within the range of the ADC, attenuators were inserted in the input signal appropriate to the current and target for the run. Another CAMAC crate called the Møller crate held all the modules for the Møller single and double arm detectors.

All electronics for the spectrometers except for the hodoscope discriminators were located in the counting house, making it easy to adjust thresholds and fix problems while the beam was present. Both the target insert and the Møller foils were mounted on stepper motors with controls in the counting house for switching between target cells and foils.

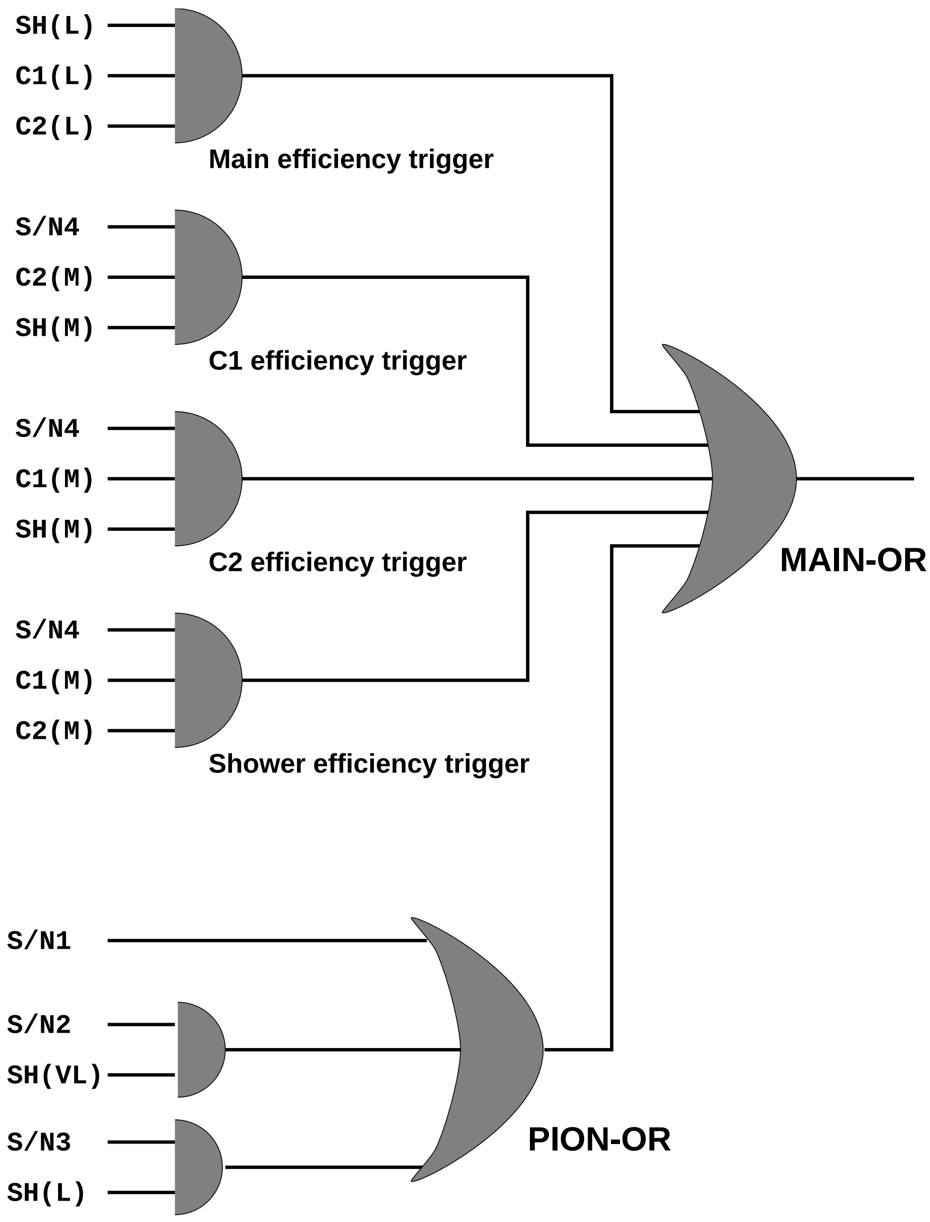

Figure 3.22: Logic for PION-OR, efficiency triggers and MAIN-OR.

3.7 Data Acquisition

Data were read from the CAMAC modules at the end of each spill. The central data acquisition (DAQ) computer was a VAX 4000/200 running VAXeln, a realtime operating system. This computer had a QBUS with three Jorway interfaces. One interface was connected to the SP4 CAMAC rack, another to the SP7 CAMAC rack, and the third to the beam and Møller CAMAC racks. This computer ran a program called DatServ which collected the CAMAC data from the Jorway interfaces and shipped them over a dedicated network to a DEC 4000/60 running the TapeServ program. Up to three EXABYTE 8mm tape drives were connected to this computer by its SCSI bus. The data were written to one tape at a time by the TapeServ program. The main reason for having multiple drives is that while one tape was being written, a tape in the other drive could be ejected and a new tape inserted and mounted. Also, it was convenient to leave the tape used for recording Møller data in its own drive between Møller run series. Due to DECNET limitations, DatServ could only transfer a maximum of 4096 bytes in the time between spills to TapeServ. If rates were high enough to produce more than 4096 bytes of data, the data record was truncated. Since the VAX 4000/200 had no monitor, a third computer, a VAX 3100, served as a terminal to run remote low-level control, debug and monitoring programs.

This dedicated network was connected across a bridge to the standard Ethernet subnet in ESA. The bridge served to shield the data and tape service computers from all traffic not specifically addressed to it. A VAX 4000/300 on the ESA subnet side of the bridge ran the DAQCNTRL program which was used by the experimenters on shift to control and monitor DAQ at the user level. This program allowed for starting and stopping runs, setting the run type, mounting and ejecting tapes, controlling the target insert and Møller foil position motors, etc. The VAX 4000/300 also had one Jorway interface attached to its QBUS that connected to CAMAC racks holding the magnets controls, high and low voltages supplies, certain scalars and a number of target monitoring modules. Programs for controlling and monitoring these devices were run on dedicated terminals in the counting house.

Four additional workstations were used for online analysis (OLA). The OLA programs running on each of the workstation obtained data from the TapeServ program. SP4 and SP7 analyses were each run on a separate VAX 4000/90 while beam and Møller analyses were each run on a separate VAX 4000/60. The spectrometer analyses were very time intensive and were generally only able to process one or two spills per second thus sampling less than 3% of the spills. This was sufficient for all online aspects of the spectrometer monitoring. The beam analysis was able to process around 50% of the spills which was important to investigate beam rastering and stability.

Most target control and monitoring was done on a Macintosh Quadra system that was connected by cables to the target apparatus in ESA through a GPIB interface. The program LabView was used to control the target insert position, refrigerator and field magnets, monitor the polarization and temperature, and send information to the DAQCNTRL program. The later was done through a FIFO connection to a control module in a CAMAC crate.

Chapter 4

Data Analysis

During the data taking period of E143, over 300 8mm tapes with 2 gigabyte capacity were written containing raw ADC and TDC information. These tapes were processed to convert the raw data for each run into beam charge, particle tracks, shower clusters, and other physics quantities which were written to a new set of tapes called Data Summary Tapes (DST). From the processed data, electrons were selected and counted using a variety of cuts to define a good electron. Cross sections and asymmetries were then calculated from data binned in the appropriate kinematic variables. The spin structure functions were extracted from the cross sections.

The first few weeks of running starting in late October 1993 were dedicated to checkout and debugging of the detectors and electronics. Data taking started in earnest on November 22. On December 1, 1993, running was stopped for a full day to make adjustments and fix several problems. The most critical problems were a misalignment of a mirror in the 4.5° 4 meter Čerenkov tank and a failure to record the witness pulse (see Sec. 3.2.2) which, despite having no spectrometer data, was need for verifying a properly record beam helicity. Data taking ended February 6, 1994.

The majority of E143 running was done at 29 GeV for both longitudinal and transverse spin orientations, since the main goal of the experiment was to obtain a precise measurement of the spin structure functions g_1 and g_2 in the deep inelastic region. Longitudinal data were also taken at 16 GeV for nine days and at 9.7 GeV for 5 days to study Q^2 dependence and to improve the models for radiative corrections. The 9.7 GeV data also contained significant information about the resonance region, which is the focus of this thesis. Data for 9.7 GeV were taken at the very end of the

E143 experiment corresponding to runs 3142-3378.

In addition to data on the $^{15}NH_3$ and $^{15}ND_3$ targets, data were taken on empty, aluminum and carbon targets for calibration and testing purposes. A fraction of beam time was spent on special runs for determining ADC pedestals, calibrating the toroids, determining the positron background, and measuring the beam polarization using the Møller Polarimeter.

4.1 DST Production

The most computationally intensive part of the data analysis was the production of DSTs. The purpose of the DST analysis was to convert the raw ADC and TDC readings from the beam monitors and spectrometers into physics quantities. The DST provided all of the information needed to separate electron events from pion events and other background in the next stage of the analysis. Also, the total volume of raw data was reduced by a factor of four.

Processing of the beam information for the DSTs was straightforward. Using pedestal and calibration information, actual beam pulse charge in gigaelectrons was determined for both toroids. Beam centroid and spot size (defined as one standard deviation) on the foil array was calculated. A correction was also made to the centroid due to the fringe fields of the spectrometer magnets. These magnets were close enough to the beam line exiting from the target to affect its path. The correction adjusted the foil array coordinates to appear as if there was no effect.

All other beam values were simply copied over with possible pedestal subtraction. For goodspill, badspill, and rastering magnet values, only a relative value on a per run basis was desired. Similarly, ADC values from the Čerenkov detectors were simply pedestal subtracted and copied to tape. Processing of the shower counter and hodoscope information was more involved and is explained in detail below.

DST production code was run IBM RS/6000 workstations running AIX 3.2. Production was automated using the LSBATCH system which handled job submission to a farm of twenty RS/6000s for which interactive login was disabled. Typically, one run took close to an hour to process.

A first set of data summary tapes (DST1) was made a few month after the end of the experiment. A year later, after significant improvements and bug fixes had accrued, a second set (DST2) was created. DST1 data were written on Exabyte 8 mm tapes. Later, a tape silo system became available at SLAC and these tapes were copied into it. The silo system is an automated tape library service capable of administering hundreds of tapes using robotic arms to transfer tapes between shelf storage and tape drives. It is easily controlled with a small set of UNIX commands. When DST2 was produced, it was written directly to tapes in the silo system. DST2 was used for the present analysis.

4.1.1 Shower Cluster Finding

Most incident particles deposited their energy in more than one block of the shower counter. It was therefore necessary to determine which blocks registering an energy deposit belong to the same particle event. These blocks were grouped together into a cluster. When there was no overlap of clusters in the counter, determination of clusters was straightforward. However, when overlaps were present, a more sophisticated method of identification was required. Resolving of the clusters was done by a "cellular automaton" algorithm, an iterative process where at each step the value of a cell is determined from transition rules by the value of its neighbors[52, 53].

The blocks of the shower counter correspond to cells and therefore each cell has up to 8 neighbors. Blocks with no energy deposited are excluded. The energy deposited is taken as the cell's initial value. The state of the system then evolves according to the following three transition rules:

1. A cell is a "virus" if its value is higher than the value of each of its neighbors.
2. At each iteration, a cell will take the value of its highest energy neighbor.
3. Once a cell is labeled a virus or becomes "contaminated" by one (takes the value of a virus according to rule 2), it becomes impervious to change.

The process is repeated until all cells are unchangeable. Cells with the same final value are collected into the same cluster. Figure 4.1 shows three iterations of a sample cell array.

For E143 analysis, the "cellular automaton" algorithm was run on the shower counter ADC data from each MAIN-OR trigger. Up to four triggers were possible per spill. When the blocks with maximum energy from two different clusters were located next to each other, the algorithm failed. It was also possible for two triggers to be close enough together in time that the energy deposited in a block from an event would be partially read by both spills. Similarly, an event that did not cause a trigger but occurred shortly before one would also cause a partial reading which

			0.2		
			3.0	1.0	0.2
0.3	0.2	0.4	7.0	2.0	0.2
2.0	8.0	1.0	0.4		
0.2	0.6	0.3	0.2		

$\Rightarrow$

			3.0		
			7.0	7.0	2.0
8.0	8.0	8.0	7.0	7.0	2.0
8.0	8.0	8.0	7.0		
8.0	8.0	8.0	1.0		

$\Rightarrow$

			7.0		
			7.0	7.0	7.0
8.0	8.0	8.0	7.0	7.0	7.0
8.0	8.0	8.0	7.0		
8.0	8.0	8.0	8.0		

Figure 4.1: Three iterations of the cellular automaton algorithm on a region of cells.

were labeled “ghost clusters”. It was determined in E143 that such overlaps in space and time were sufficiently rare to not merit a change in the algorithm although this was a concern in the efficiency study as will be discussed below.

Once found, a cluster’s total energy was then determined by the sum of the true energy deposited in all member blocks. The centroid of the cluster was calculated as the energy weighted average of the centers of the involved blocks. This centroid was later used as an additional point in tracking.

4.1.2 Shower Cluster Identification

The shower counter was capable of discriminating between pions and electrons using a specially trained neural network[53, 54]. The characteristics of each cluster were feed into the neural network which returned a value between -1 and $+1$. The closer the value was to $+1$, the more likely the cluster was produced by an electron. The closer the value was to -1, the more likely it was from a pion.

The neural network used was a multi-layered feed-forward network which consists of layers of neurons with one input layer, any number of hidden (intermediate) layers and an output layer. The output of each neuron on one level was directly connected to each neuron on the next level. The output of a neuron i was obtained by applying a transition function f on the weighted sum of its inputs using the formula

$$O_i = f\left(k_i \sum_{j=1}^{N} W_{ij} O_j\right) \tag{4.1}$$

where O_j is the output of neuron j in the previous level and W_{ij} is a weighting coefficient, called the synaptic strength, between neuron i and neuron j and k_i is an overall adjustment coefficient for neuron i. For E143, the transition function was a sigmoid defined by

$$f(x) = \frac{e^x - 1}{e^x + 1} \tag{4.2}$$

The coefficients W_{ij} and k_i were determined in a supervised learning stage in which well-classified patterns were input into the network. An error function was minimized by updating the coefficients at each presentation using an error back propagation algorithm with the expected output. A positive weighting coefficient corresponds to an excitation of the neuron while a negative one corresponds to an inhibition.

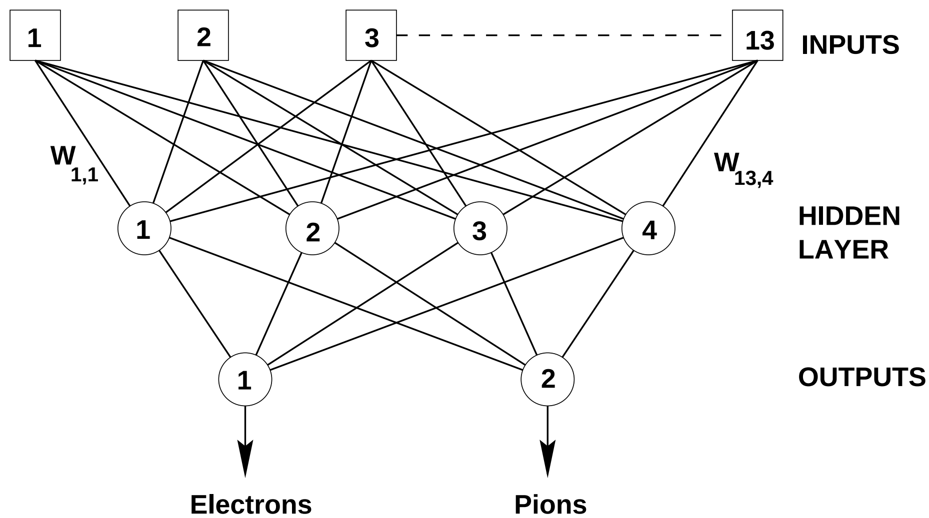

Figure 4.2: Scheme for the multi-layered neural network.

The network used for E143 consisted of three layers as shown in Figure 4.2. The input layer had 13 input neurons, the hidden layer had 4 neurons, and the output layer had 2 neurons, one to signify an electron and the other a pion. The inputs for each cluster were:

E_{tot}:	Total energy in the central 9 blocks of cluster.
E_5/E_{tot}:	Ratio of energy in the central block to E_{tot}.
E_{16}:	Energy in ring of blocks around the central 9 blocks.
E_1 to E_9:	Energy in each of the central 9 blocks.

N_{blocks}: Number of hit blocks in cluster.

The learning step used a simulated set of 3000 electron and 3000 pion events produced by the Geant program corresponding to a beam energy of 29 GeV. Initial weights were chosen randomly in the range -0.01 to 0.01. The training was completed after 100 iterations. The neural network code was added to the DST production program and run on each cluster. Only the result of the electron output neuron was written to DST. A plot of this value from a sample run is shown in Figure 4.3.

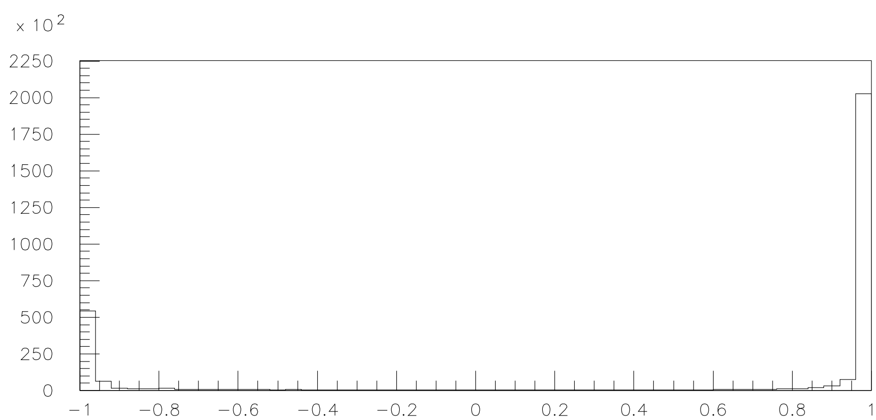

Figure 4.3: Output from neural network for a typical run.

4.1.3 Tracking

The most computationally intensive part of DST production was reconstructing the tracks of the scattered particles using the spatial and timing information from the Čerenkov detectors, hodoscopes and shower counter. A reading on a hodoscope finger's TDC signified a hit on that finger at a given time. The finger's spatial resolution was taken as its width divided by $\sqrt{12}$ which is the root mean square distance from the center of the finger. A reading on the TDC connected to the low threshold discriminator for a Čerenkov detector signified a hit on the Čerenkov at the given time. For spatial fitting, the Čerenkov was considered to be a finger with 100 m (i.e. infinite) spatial uncertainty.

A shower cluster was treated like two hodoscope finger hits, one in x and another in y; however, the tracking code always required that both or neither be present in a track. The spatial resolution for a cluster was taken to be 10 mm which was chosen as a reasonable uncertainty on locating the shower centroid. A more precise value was not needed for reliable tracking and would only add unnecessary computation time to DST production. A cluster's time was that of its corresponding MAIN-OR trigger. Since there were a maximum of four MAIN-OR triggers per spill, there were only four possible unique cluster times while each hodoscope finger and Čerenkov had independent time measurements limited only in number by the 16 channels of their TDC. The time resolution used for all TDC values was 1 ns.

Track	required # of hits		
Class	Čerenkov	Hodoscope	Shower
1	2	4	2
2	2	4	0
3	0	4	2
4	0	6	0

Table 4.1: Track classification according to the minimum number of hits on each detector.

Tracks were divided into four classes according to the number of hits from each detector type. The definitions for these classes are shown in Table 4.1. The track-finding algorithm looped over the track classes from 1 to 4, finding tracks that fit the class. Once a hit was marked as part of a track, it was removed from the pool of candidates for finding the next track both in subsequent searches for the same class and for searches in later classes. During each search pass, a group of candidate hits meeting the current classification were found using a rough timing cut. The times for these hits were adjusted to a common z to account for the propagation time of the particle through the spectrometer. These adjusted values were then fit to get a median time and the hit furthest from this time found. If the hit was more than 4σ away, it was dropped from the group and from future searches and the remaining hits refitted. This fit–drop process was repeated until no hit was off by more than $4\ \sigma$ or there were not enough hits left to qualify for the class. In the later case, the hits were discarded and another search started. If the hit group passed this timing cut, the fit–drop process was repeated using a combination space–time fit.

Once a group of hits passed the fitting procedure, all hits still available were looped over and checked to see if they fit the track and if so were added to the group. The hits were then refit to get the final path of the track. The track's path was then required to pass through the first and last x hodoscope planes in the x direction and through the first and last y hodoscope planes in the y direction. Note that this does not mean that any hits were required in these planes. At this point, the track was declared official and written to the DST. A check was made to see if the track qualified as a track under a previous class and if so it was relabeled to that class. The momentum, angles and position of the track at the target were reconstructed and also written to tape. The search loop was then continued until no initial pool of candidates could be found. At that point, all hits were reinstated for search at the next class level except for those that had been assigned to an official track. A search loop for the next class was then begun.

Track classes were used mainly as a diagnostic tool in studying the tracking algorithm. A track's classification was not used in the final analysis to extract spin structure functions.

4.2 Particle Identification and Counting

The next step of analysis was to use the information written to DST to extract and count electron events needed to form asymmetries and cross sections. Every cluster and track represents a possible electron which must be considered. Cuts must be made on the quantities available to decide whether an event is an electron or not. The stricter the cuts the purer the sample but also the lower the efficiency. One must try to find a set of cuts, that minimize contamination while maximizing overall efficiency. Another concern in defining cuts is that a cut must not systematically favor one beam helicity over the other or a false asymmetry will result.

The first step to finding proper cuts was to histogram the available quantities to find areas of separation between pion and electron events, as well as identify good and bad data. Bad data could be the result of a detector failure, an electronic problem or a bug in the DST production code. If no method of correction could be found, entire affected runs were removed from the analysis. A initial set of cuts to define a good electron were then made, the data reprocessed and the results observed. Adjustments and additions were made to try to optimize the tradeoff between purity and efficiency and the process repeated until a suitable electron definition was found.

Although individual cuts were independent of the target type[1], they are not in general independent of the kinematics. The data taken at 29 GeV were analyzed first to obtain the spin structure functions g_1 and g_2 at the highest possible Q^2. The 16 and 9.7 GeV data were analyzed later to investigate the Q^2 dependence of g_1 The cuts created for 29 GeV data were used as starting cuts for 9.7 GeV and were subsequently adjusted. The resulting set of cuts for the 9.7 GeV data in the resonance region are discussed below.

Counting was done on a run-by-run basis. The cuts fell into two categories for beam and detector. If a spill passed beam cuts, then the charge measured for that spill was added to the run's sum. Separate sums were kept for each beam helicity (left or right). Detector cuts were then applied to each event in the spill and those that passed were summed into an array indexed by the helicity, spectrometer angle ($4.0°$ or $7.5°$), and kinematics.

For the resonance region analysis, two separate kinematic binnings were used: one in x and Q^2 and another in W^2 and Q^2. The former allowed for comparison to the deep inelastic analysis which had bins only in x and Q^2. The additional binning in W^2 and Q^2 was done in the present analysis, since W^2 is the most convenient variable for identifying the resonance peaks. For x, 38 logarithmically scaled bins were used between 0.01 and 0.90. For W^2, 80 equal sized bins were used between 0.0 and 10.0 GeV^2. For Q^2, 10 equal sized bins were used between 0.0 and 1.0 GeV^2 for SP4 and between 0.0 and 2.0 GeV^2 for SP7.

A spectrometer provides two independent measures of an electron's momentum. One comes from tracking which directly measures the momentum, p of the particle. The second comes from the shower cluster energy, E. The method that gives the most accurate result depends on the energy of the electron. For the shower counter, resolution goes as roughly $1/\sqrt{E}$ and therefore improves at higher electron energy. For tracking, resolution improves the more a particle is bent and therefore decreases with higher energy. Figure 4.4 shows a plot of the resolution of the two methods[2]. For the analysis, the momentum from tracking was used since it is more accurate in

[1]This is not necessarily true of the cut's efficiency

[2]The resolution functions for tracking in this figure were found using a Monte Carlo calculation prior to the experiment

the energy range observed.

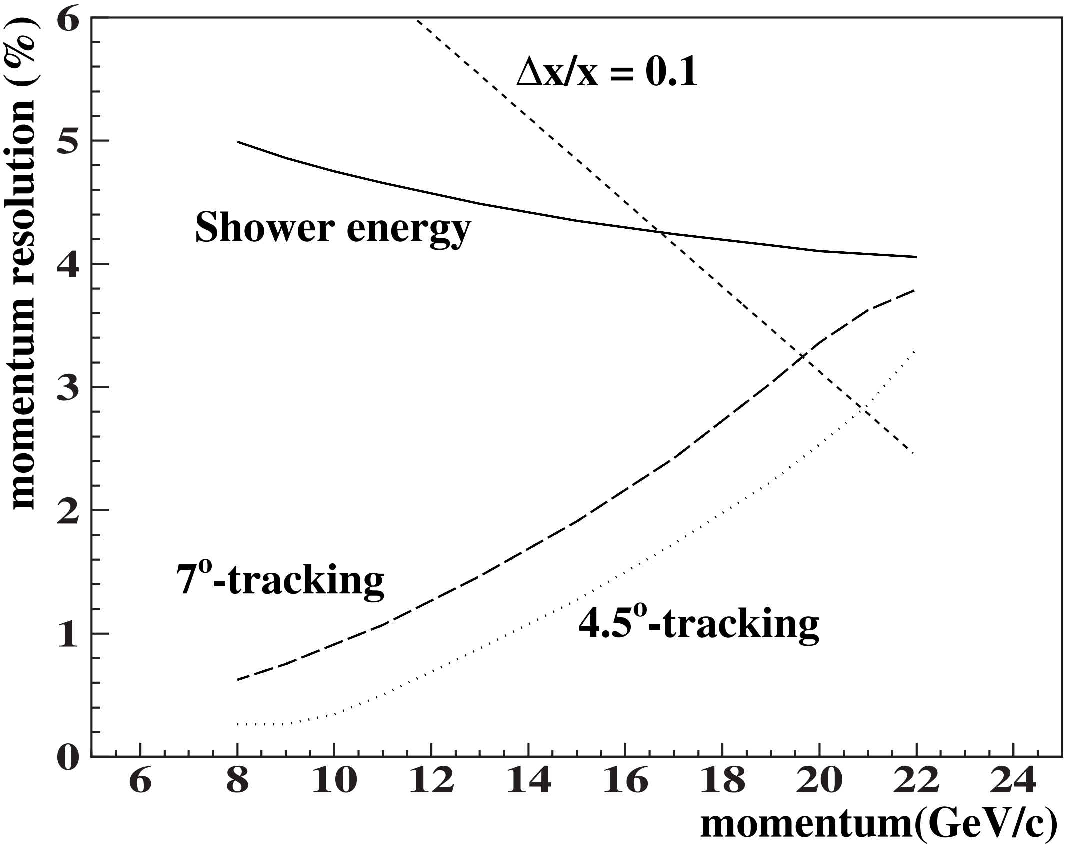

Figure 4.4: Comparison of momentum resolution from tracking and shower counter.

At the end of a run, the charge sums and arrays were written out to a file referred to as a "sumfile". As will be discussed later, additional information was also accumulated during this counting phase and written to this sumfile for use in dead-time and efficiency calculations. Each run typically took up to an hour to process on the LSBATCH system and produced a sumfile of 230 kilobytes which was written to hard disk.

4.2.1 Beam Cuts

The first requirement for a good event was that the beam quality be acceptable. Due to the rastering, beam stability was sometimes difficult to maintain. It is especially important to exclude beam spills that collided with non-target materials or that missed the target entirely. These events would dilute the polarized scattering or overcount the incident charge. Scattering from certain objects, such as the target cell windows or NMR coil, was unavoidable and required special correction. One problem with determining good cuts for beam quantities was that some of them, such as current, changed by rather large amounts from run to run or even within a run. To accommodate this, it was decided to sacrifice the statistics from the first few hundred spills in a run to calculate averages for some of the quantities. This was a very negligible loss since most runs had on the order of 300,000-500,000 spills. Every 5000 spills, new averages would be calculated for the next 236 spills. During this time, no data would be lost, since the averages from the previous calculation would still be used.

The primary method of excluding non-target hits and target misses was by cutting on the goodspill and badspill ADC values. A significant reading on the badspill monitor signified that the beam was hitting something upstream of the target. A cut that the badspill ADC value be less than 700 was used. A low reading on the goodspill monitor represents a miss of the target while an abnormally high reading meant that something in addition to the target was being hit. Therefore, both a maximum and minimum cut were used. Since the goodspill reading was extremely dependent on current, cuts were made using a running average. The upper limit was set to 1.75 times the average and the lower limit set to half the average. Figure 4.5 shows typical ADC spectra for the goodspill and badspill monitors.

For measuring the beam current, toroid 2 was found to give the most accurate reading and was therefore used for beam cuts and for integrating the incident charge. The current was found to drift by a fair amount over the time of a run. The absolute value of the current was not a major concern at this point. However, gross deviations from the average current signaled a more severe stability problem. A cut was specified to accept only spills whose charge was within ± 0.75 gigaelectrons of the dynamic average. Also, to avoid spills with very low charge or no charge at all, a lower limit of 0.5 gigaelectrons was also applied.

An abnormally small or large beam spot size was another indicator of beam instability. The beam was therefore required to have a spot radius between 0.5 and 10 mm on the foil array. The beam position on the foil was not allowed beyond a 12 mm radius from the foil center. This is actually greater than the size of the target, but the beam had a sizable divergence between the target and foil array. Also, the center of the raster pattern was not always aligned on the foil center. An additional

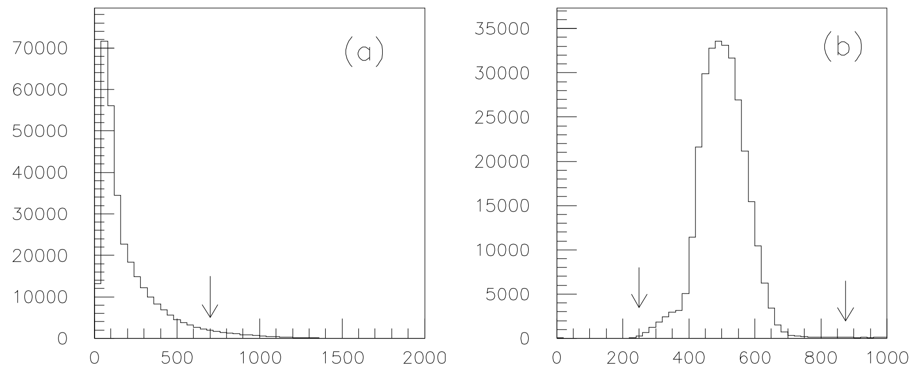

Figure 4.5: Spectrum of (a) badspill and (b) goodspill detector ADC values. Arrows show location of cuts.

requirement was made that the beam position be within 16 mm of the center of the raster pattern. The raster pattern center was calculated dynamically as the mean position of all points on the raster pattern. Histograms from a typical run of the distance of the beam position on the foil array from (a) the foil array center and from (b) the raster pattern center are shown in Figure 4.6. The difference between the two spectra show that in this example the raster pattern center is slightly off the foil array center.

The last beam cut eliminated events with inaccurately recorded beam helicity. The helicity was chosen pseudo-randomly at the source and the value chosen sent as a binary value 2 miles down the linac to ESA. It was found for unknown reasons that this value was sometime corrupted. To guard against this error, the beam helicity of the spill was checked against a prediction using the same pseudo-random algorithm used at the source.

Using 0 for left-handed helicity and 1 for right-handed helicity, the algorithm determines the helicity of the current spill by doing a logical XOR of the helicity from 19 spills before with the helicity from 33 spills before. Analysis therefore required an initial seed of 33 spills to be read in before a prediction could be made. The next 33 spills were then checked to see if they were consistent with the first 33.

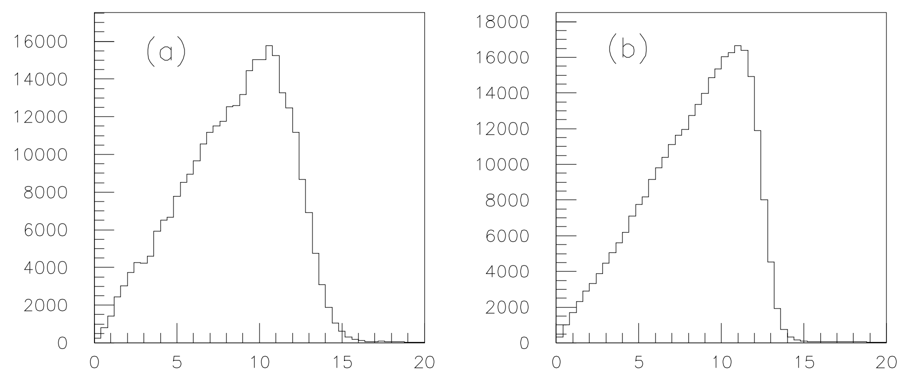

Figure 4.6: Spectra of distance of beam position to (a) foil array center and (b) raster pattern center.

If so, then analysis continued to check each spill's helicity with the prediction and processed the spill if it passed. If a failure to match was found at any point, a new seed was taken and the process repeated. The loss of statistics from losing the first 66 spills was negligible.

4.2.2 Detector Cuts

Once a spill passes the beam cuts, the next step is to count the electrons scattered into the spectrometers. The two main sources of background in the spectrometers were from pions produced in the target and electrons generated by pair production. Pions are removed by using cuts on Čerenkov, tracking and shower quantities defined below. Electrons from pair production were corrected for by measuring positrons in specials runs with the field of the spectrometer magnets reversed. The runs were then processed in the same way as normal runs to remove pions and get the positron rate. From this rate, a correction factor was determined and applied in the asymmetry and cross section calculations as discussed in Section 4.3.

An electron event should produce a signal in both Čerenkov detectors and the shower counter. The MAIN efficiency trigger, which required a signal from the low threshold discriminators from both Čerenkov detectors and the shower counter, was an obvious choice for a first level cut. However, during offline analysis, it was discovered that some shower blocks often failed to feed their signal in to the shower sum signal used by the discriminators during the 9.7 GeV running. A trigger cut was therefore not made. Since additional cuts are made individually on Čerenkov and shower values, a trigger cut serves only as a way to speed up the analysis program by skipping events with no chance of containing an electron. It is not a crucial cut for obtaining electron purity.

Electron candidates in each spill were found by looping over the clusters found. This effectively made a cut requiring that a shower cluster existed. The energy reading from a shower cluster centered on an edge block could not be trusted since a significant amount of radiation would leak out the side. Therefore, all clusters centered on an edge block were rejected.

A cut was then made on the Čerenkov ADC signals for the trigger in which the cluster occurred. Typical ADC spectra are shown for the Čerenkov tanks in Figure 4.7. The wide peak centered above channel 100 is the electron signal. Some high energy pions can produce Čerenkov radiation in the detectors. However, these pions produce fewer photoelectrons and show up only the in the lower channels of the ADC spectrum. A cut requiring the signal to be greater than 50 was used to remove the largest portion of this contamination.

For each cluster, the tracks present in the trigger were looped over and the one that best matched the shower cluster was found using a chi-squared fit. This effectively made a cut requiring that a track existed. The chi-squared was calculated using

$$\chi^2 = \left(\frac{\Delta x}{\sigma_x}\right)^2 + \left(\frac{\Delta y}{\sigma_y}\right)^2 + \left(\frac{\Delta t}{\sigma_t}\right)^2 \tag{4.3}$$

where Δx and Δy are the distances in the x and y direction, respectively, between the cluster centroid and track's position extrapolated to the z position of the shower counter. Δt is the difference between the time of the track and the cluster's trigger time. The σ's are the errors in the distances and time and were taken to be 5 mm and 1 ns.

Note that only the best matching track was found. Several additional requirements were needed to verify that the track truly belongs to the same electron that produced the cluster. Individual cuts were made on Δx, Δy, and Δt. Both $|\Delta x|$ and

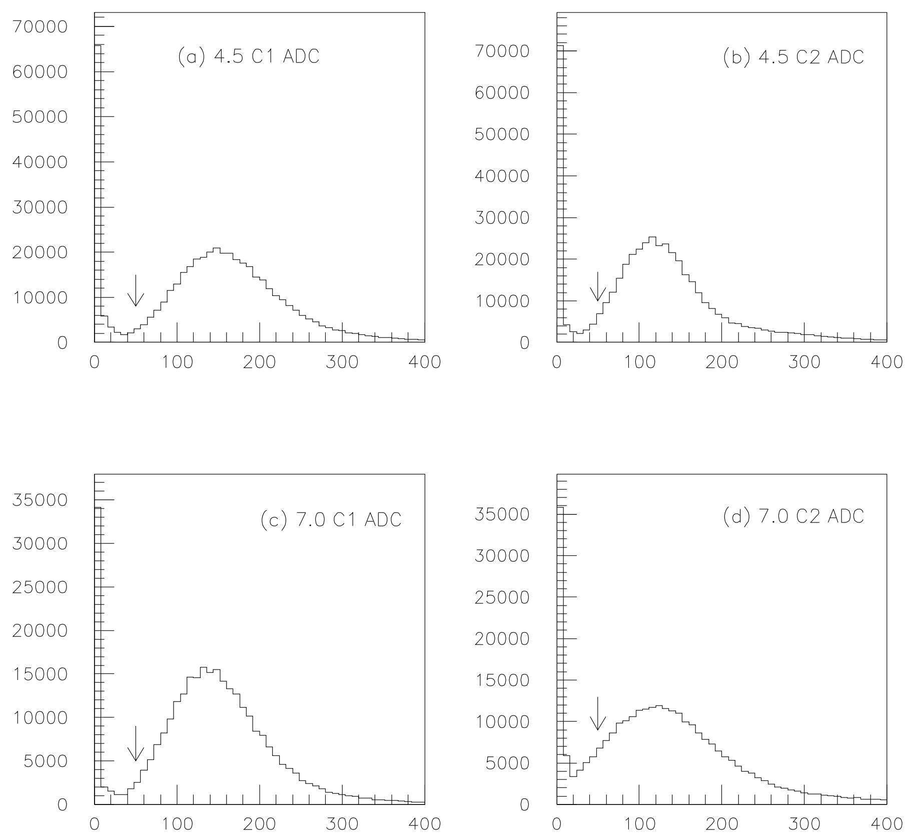

Figure 4.7: ADC spectra from the Čerenkov tanks for a typical run. Arrows show location of lower limit cut.

$|\Delta y|$ were required to be < 40 mm and $|\Delta t|$ was required to < 6 ns. An additional requirement was made that the extrapolated position of the track at the target be within 15 mm of the measured beam position. Plots of these distances for both spectrometers are show in Figures 4.8 and 4.9.

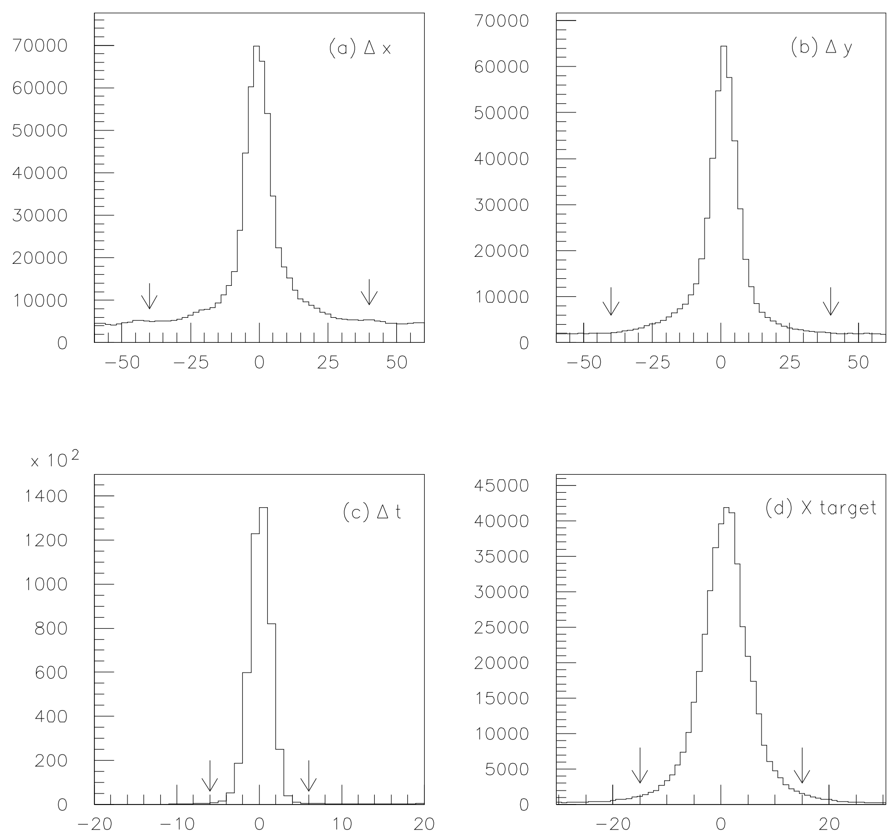

Figure 4.8: Distribution of (a) Δx, (b) Δy, (c) Δt and (d) Δx_{targ} for SP4 in a typical run. Arrows show location of cut limits.

At energies of a few GeV, electrons deposit more than 99.9% of their energy in the shower counter in an electromagnetic shower. The Čerenkov light produced and collected in the shower counter's phototubes is directly proportional to the electrons energy. Hadronic particles such as pions lose their energy mostly through ionization,

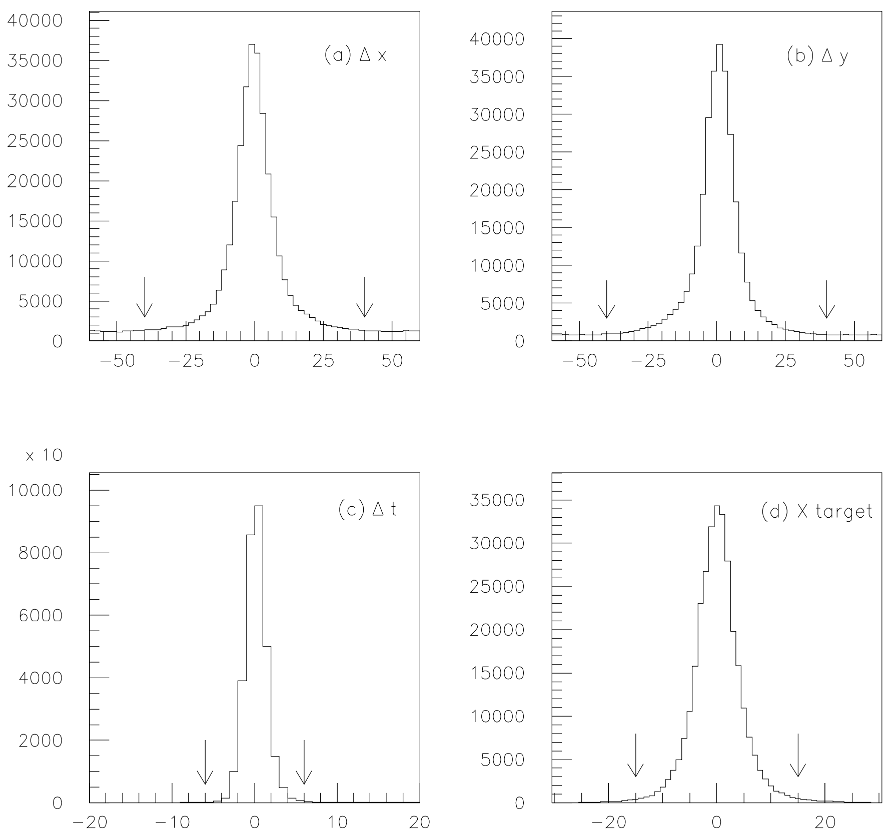

Figure 4.9: Distribution of (a) Δx, (b) Δy, (c) Δt and (d) Δx_{targ} for SP7 in a typical run. Arrows show location of cut limits.

producing a smaller signal in the shower counter than electrons at the same energy. A powerful way to distinguish electrons from pions is to cut on the ratio of the shower counter energy to the tracking momentum, E/p. Electrons can be considered nearly massless at this energy scale and therefore have a E/p of 1.0. This ratio for pions will be noticeably less than 1.0. The solid lines in Figure 4.10 show the distribution of E/p of all events in a typical run for the two spectrometers. The dotted lines then show the distribution of the ratio after all cuts defined previously had been made. To make the sample even cleaner, a cut was made that E/p must be between 0.7 and 2.5.

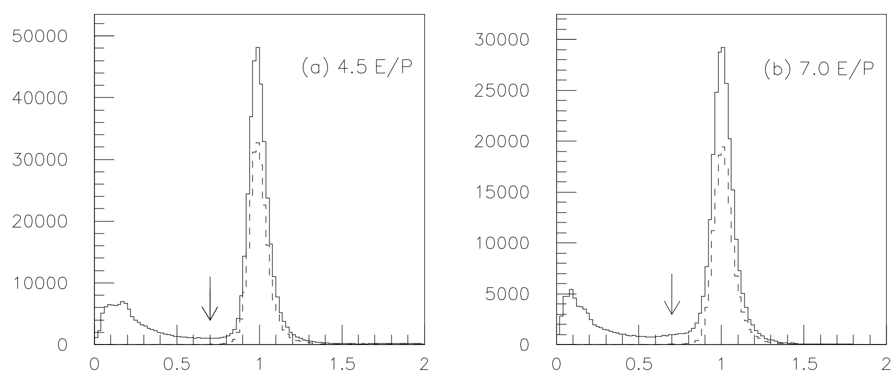

Figure 4.10: Distribution of E/p for (a) SP4 and (b) SP7 for a typical run. Solid line is before cuts and dotted line is after all cuts except the one on E/p. Arrows show location of lower limit cut.

The maximum of 2.5 was chosen to allow for clusters containing two electron hits. Since the momentum from tracking is used to calculate kinematic quantities, the incorrect shower energy is not a problem. A stricter maximum cut would have decreased the efficiency without significantly improving the electron purity.

At 9.7 GeV, the neural net was inefficient because a typical shower can be almost completely contained in a single shower counter block. The neural net depends on the multiblock pattern of a cluster to decide whether it is a pion or an electron. Therefore, no cut was made on the neural net result.

Beam cuts	Detector cuts
goodspill between 0.5 and 1.75 of average	MAIN-OR trigger (effective)
badspill < 700	C1 > 50 and C2 > 50
toroid 2 within $0.75 \times 10^9 e^-$ of average	non-edge centered cluster
beam spot size between 0.5 and 10 mm	$\lvert\Delta x\rvert < 40$ mm and $\lvert\Delta y\rvert < 40$ mm
distance to target center < 12 mm	$\lvert\Delta t\rvert < 6$ ns and $\lvert\Delta x_{targ}\rvert < 15$ mm
distance to raster center < 16 mm	E/p between 0.7 and 2.5
helicity matches algorithm	

Table 4.2: Summary of electron identification cuts.

Once an event passed all the cuts, the appropriate bins in the x versus Q^2 and W^2 versus Q^2 arrays were incremented. For the average run, over 50% of the clusters looped over passed the electron cuts. This corresponded to about 450,000 good electron events per run. Table 4.2 summarizes the cuts made for identifying electrons.

4.2.3 Absolute Efficiencies

Unlike the case of asymmetries for which detector efficiencies cancel out in the ratio, absolute detector efficiencies are required to determine absolute cross sections. These efficiencies may be dependent on event rate. However, this effect is taken into account separately using a dead-time correction discussed in the next section. In order to not "double correct" for rate effects, the detector efficiencies were determined in the limit of low event rate.

In order to find an electron event, the analysis algorithm requires that there be a proper trigger, a cluster in the shower counter and a valid track through the spectrometer. The absolute efficiency of the spectrometer as a whole should be equal to the product of the efficiencies of all subsystems that identify the electron. Cuts to effectively eliminate pions from the electron sample invariably throw away a fraction of good electrons, thereby lowering the absolute efficiency of the device.

The general procedure for calculating efficiencies is to first establish, independent of the quantity (or cut) for which an efficiency is being calculated, that an electron event occurred. One counts the number of times an undeniable (TRUE) electron event occurs. When such an event occurs, one also counts the number of the times the quantity in questions indicates a good electron. Dividing the second sum by the first yields the efficiency of the quantity.

Another common method of determining a detector's efficiency is to compare the measured height of the elastic peak to that predicted from well known formulas. However, due to the poor resolution of our spectrometers and low counting rate, this was not feasible. Also, this assumes that the efficiency does not depend on kinematic quantities such as W^2, which was not a valid assumption for our experiment.

Where possible, it is desirable to calculate efficiencies for each W^2 bin used in accumulating electron counts. Often, W^2 binning is not possible. In these cases, we assumed that there was no W^2 dependence to the efficiency and increased the error on the efficiency appropriately.

The first question is how efficient is the MAIN-OR trigger itself without which no data would have been recorded. Unfortunately, there is no direct way of determining a specific value for the trigger's efficiency. However, assuming no electronic problems, the designers of the spectrometer are confident that the MAIN-OR efficiency is essentially 100%. We make a conservative estimate of 1% systematic error on this value.

The efficiency of the cut on the logical AND of the MAIN efficiency trigger and the shower efficiency trigger is similarly impossible to obtain directly. As previously mentioned, the shower efficiency trigger was included to correct for an electronic problem with the MAIN efficiency trigger (see Sec. 4.2.2). The trigger cut's efficiency is believed to be 100%, but due to the problems experienced a higher error of 2% is associated with this estimate.

Since the analysis loops over clusters, one also needs to know the efficiency of finding a cluster when a good electron event exists in the trigger. Inefficiencies could be due to the shower counter itself or limitations of the cluster finding algorithm. For this calculation, a good electron was assumed to exist in the trigger if the ADC values from both Čerenkov detectors are over 100 and their relative timing was consistent with the flight time for a speed-of-light electron. The cluster finding efficiency for SP4 was 99.9% while for SP7 it was 99.1%. The statistical error on both these efficiencies is 0.2%. Note that this represents only the chance that a cluster exists given that the Čerenkov detectors claim there is a good electron. It does not verify that the cluster is a valid one for an electron. That is the job of the track–cluster matching cuts for which a separate efficiency is calculated below.

As mentioned previously, the efficiencies needed to be determined in the limit of low event rate. In order to contribute to the efficiency calculation, a cluster was required to be isolated in space and time. This was done by asserting that the cluster was the only one in the trigger and that the trigger was separated from previous and subsequent triggers by at least 80 nanoseconds. It was observed that the effect of including and removing this requirement had an effect on the efficiency of the order of the dead-time correction ($\sim$4%).

The next loop in analysis was over the tracks in a trigger, given that a cluster is present. Therefore the efficiency of having a track, any track, when there is a good electron event is needed. It is possible for this efficiency to have a non-negligible kinematic dependence; however, binning in W^2 as desired is not possible since one must determine W^2 from the track itself. When a track is not present, one cannot know to which W^2 to attribute the inefficiency. Binning using a W^2 calculated from the cluster's energy is possible and can give some indication of a kinematic dependence despite the inferior resolution. Another method is to bin by shower counter block since each block has a strong but highly smeared W^2 correlation.

Both these methods showed no kinematic dependence to at least the 2% level if the bins (blocks) on the edge of the acceptance were ignored. For both targets in SP4, the tracking efficiency was 100%. This can be expected due to the higher rate in the spectrometer which results in more noise in the detectors and thus a higher probability of some track being found. The tracking efficiencies in SP7 were found to be 97.2% for the NH_3 target and 96.3% for the ND_3 target. The efficiency of a spectrometer should be independent of the target in the low count rate limit, so the small difference was taken as an indication of systematic error. Including observed bin to bin and block to block variations, an overestimate of 2% was made for the error on all tracking efficiency values.

In order to avoid correlation complexities, the spectrometer cuts were arranged into three groups for calculation of efficiencies. The first two groups were simply the two Čerenkov cuts. The third group, called the track–cluster matching cut, or TkCl cut, included all the other cuts (Δx, Δy, Δt, Δx_{targ} and E/P). For determination of one cut's efficiency, a stricter cut was required on the other two for assumption of a TRUE electron event. Also, a neural net cut was used to make the TRUE electron sample even purer. Despite the neural net's low efficiency for the 9.7 GeV data, sufficient statistics were available for accurate efficiency determination within systematic errors.

Each cut group's efficiency was binned in W^2 both using tracking P and shower

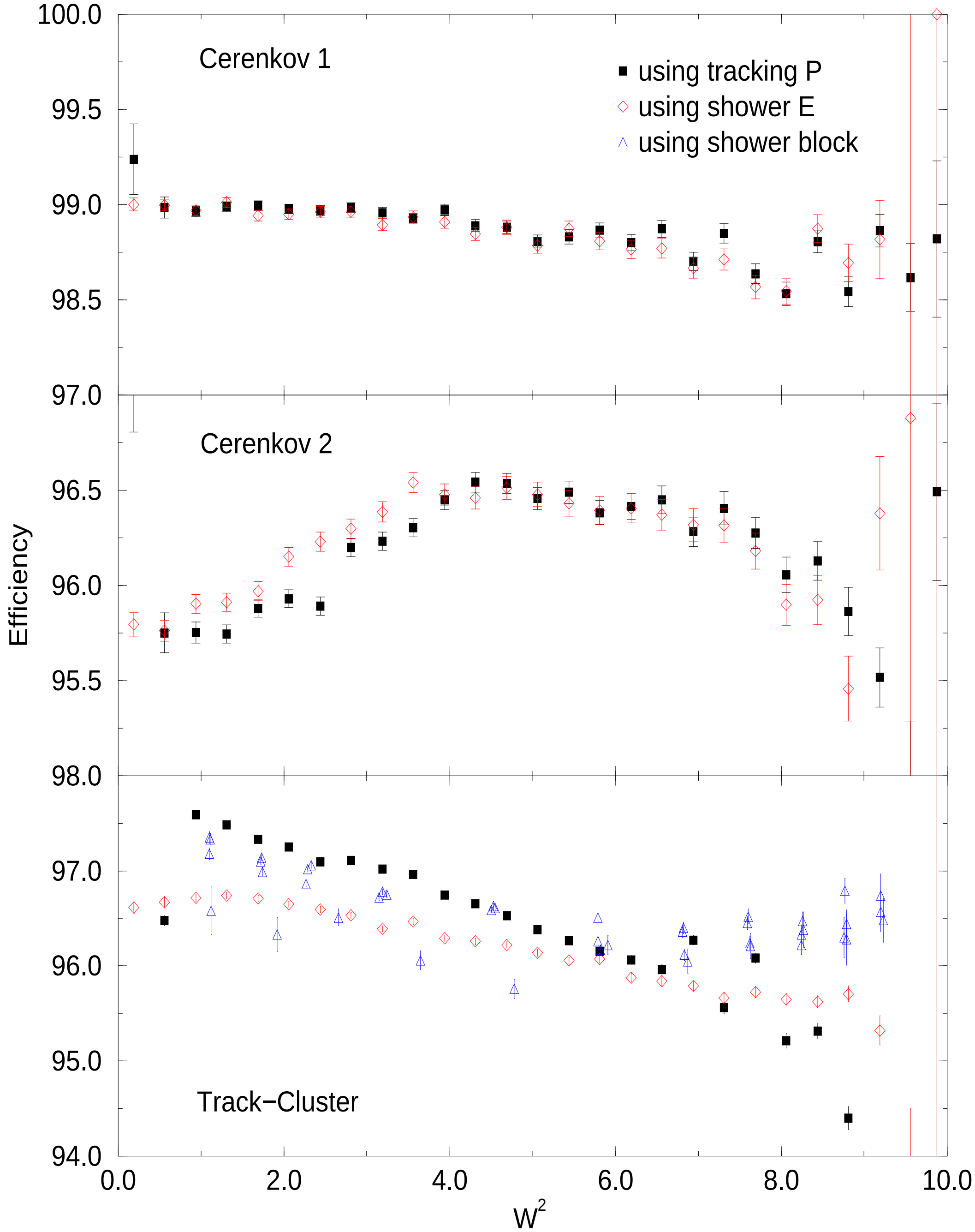

Figure 4.11: Cut group efficiencies for SP4 with $^{15}NH_3$ target.

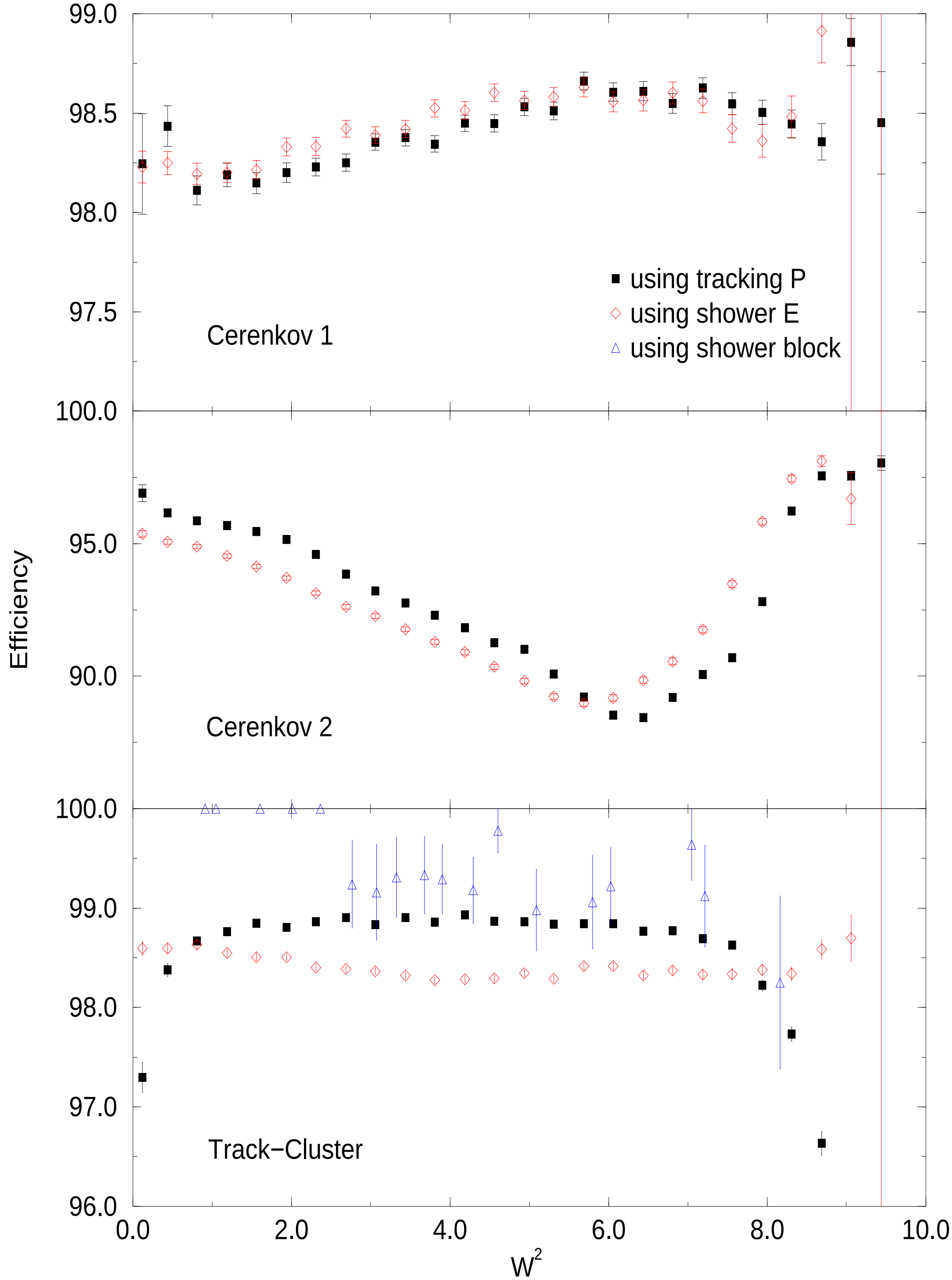

Figure 4.12: Cut group efficiencies for SP7 with $^{15}NH_3$ target.

E for the calculation. In addition, because of the TkCl cut's obvious dependence on both of these quantities, the TkCl cut efficiency was also binned by shower counter block as an additional check. Figure 4.11 shows the efficiency results for SP4 with the $^{15}NH_3$ target. Error bars are statistical only. The W^2 of a shower block was determined by averaging W^2 calculated from the P of each track pointing to the block. The shower block calculation was done using date from only one run for each target, which accounts for the large error bars. Both Čerenkov cuts are consistent between the two binning methods. For the track–cluster matching cut, the disagreement is far more than the statistical errors. Some of the discrepancy can be attributed to the poor resolution which can cause the methods to attribute the same event to W^2 bins more than 1 GeV^2 apart. To account for the difference, a systematic error of 2% was given to the combined group cut efficiency.

Possible correlations were investigated by calculating three additional efficiencies for each pairing of the three cut groups. For example, to look for correlation between the Čerenkov 1 cut and the TkCl cut, we applied strict cuts on Čerenkov 2 and the neural net to define an electron. From this definition, the efficiencies were calculated for passing the Čerenkov 1 cut with no requirement on the TkCl cut, passing the TkCl cut with no requirement on the Čerenkov 1 cut, and passing both cuts. The product of the first two efficiencies should equal the last efficiency if no correlation exists. This was found to be the case within statical error for all pairings of the cut groups.

4.3 Asymmetry and Cross Section Difference

After good electron samples were obtained for each spin alignment, the asymmetry and cross section differences for each kinematic point were found. For each run, we calculated an electron rate by dividing the scattered electron count by the incident charge (measured in number of electrons) and correcting for dead-time. This can be written as follows

$$R^{\downarrow\uparrow} = \frac{N^{\downarrow\uparrow} d^{\downarrow\uparrow}}{Q^{\downarrow\uparrow}} \qquad R^{\uparrow\uparrow} = \frac{N^{\uparrow\uparrow} d^{\uparrow\uparrow}}{Q^{\uparrow\uparrow}}$$

in which $N^{\downarrow\uparrow}(N^{\uparrow\uparrow})$ is the number of electrons counts, $d^{\downarrow\uparrow}(d^{\uparrow\uparrow})$ is the dead-time correction, and $Q^{\downarrow\uparrow}(Q^{\uparrow\uparrow})$ is the incident charge for spins aligned (anti-aligned). Because of the rate dependence of the dead-time correction, it was calculated and applied separately for each spin alignment.

Raw asymmetries and rate differences were then calculated for each run using

$$A_{||}^{raw} = \left(\frac{R^{\downarrow\uparrow} - R^{\uparrow\uparrow}}{R^{\downarrow\uparrow} + R^{\uparrow\uparrow}}\right)\frac{1}{P_b P_t} \tag{4.4}$$

$$\Delta R = \frac{R^{\downarrow\uparrow} - R^{\uparrow\uparrow}}{P_b P_t} \tag{4.5}$$

where P_b and P_t are the polarization of the beam and target, respectively. Statistically weighted averages of the raw asymmetries, $\overline{A_{||}^{raw}}$, and rate differences, $\overline{\Delta R}$, over runs with identical targets were then formed. Additional corrections were then applied to arrive at final asymmetries and cross section differences for each target using

$$A_{||} = \frac{\overline{A_{||}^{raw}}}{f(1 + C_1)} + \mathcal{A} \tag{4.6}$$

$$\Delta\sigma_{||} = d\sigma^{\downarrow\uparrow} - d\sigma^{\uparrow\uparrow} = \frac{\overline{\Delta R}}{\varepsilon(1 + C_1)\mathcal{D}} + \mathcal{S} \tag{4.7}$$

where f is the target dilution factor and ε is the spectrometer's efficiency. The terms $\mathcal{A}$ and $\mathcal{S}$ are additive corrections to the asymmetry and cross section difference, respectively, for radiative effects and detector resolution. The factor $\mathcal{D}$ normalizes for target density and corrects for spectrometer acceptance as well as unpolarized radiative and resolution effects. The term involving C_1 is a correction for the polarized proton in ^{15}N and as written above applies only to the proton target. A more complex correction is required for the deuteron target which will be discussed below.

The spin structure function, g_1 can be obtained from either $A_{||}$ or $\Delta\sigma_{||}$, but the two methods involve different model assumptions and require different corrections. For the case of $A_{||}$, the largest error comes from the dilution factor, f, while for $\Delta\sigma_{||}$ the largest errors come from the target density, spectrometer acceptance and efficiency. At the start of the analysis, it was not clear which method would produce the most accurate values for g_1 and therefore both methods were pursued. Determination of the dilution factor for $^{15}ND_3$ proved intractable because no accurate models for the deuteron existed in the resonance region. Therefore, $A_{||}$ was not determined for the deuteron.

The procedure for calculating the spectrometer efficiency was outlined in section 4.2.3. The rest of the corrections will be discussed below.

4.3.1 Dead-time correction

The discriminators used to form triggers for E143 had an output pulse width of 25 ns and a double pulse resolution of 8 ns. A second signal arriving within the output width will not produce a new trigger, resulting in loss of data. The discriminators were operated in updating mode: if a second signal came within the first 8 ns, it would not be seen; if a second signal occurred after 8 ns but before 25 ns, the discriminator would still only output one signal, but the output width would be extended. Due to slight mis-timing between various signals and jitter forming the trigger, the effective output pulse width was actually 32 ns.

The dead-time correction can be found from the output pulse width and beam spill length using Poisson statistics. This procedure is complicated by the width's extendibility and double pulse resolution. Using a Monte Carlo simulation [51], a probability matrix, $M(i,j)$ was generated. The elements of this matrix give the probability during each spill of observing i hits when there were j actual hits such that

$$\mathrm{freq}_{obs}(i) = \sum_j M(i,j) \times \mathrm{freq}_{real}(j) \tag{4.8}$$

where $\mathrm{freq}_{obs}(i)$ is the observed frequency of spills with i hits in the run and $\mathrm{freq}_{real}(j)$ is the actual frequency of spills with j hits in the run.

Although in principal the summation index j may go to infinity, in practice spills with more than 10 hits were very rare. For the Monte Carlo simulation, j was taken up to 16. The matrix $M(i,j)$ can be inverted to solve for $\mathrm{freq}_{real}(j)$. The dead-time correction factor to the electron rate, d, is then defined as

$$d = \frac{\sum_{j=1}^{j=16} j \times \mathrm{freq}_{real}(j)}{\sum_{i=1}^{i=16} \min(4,i) \times \mathrm{freq}_{obs}(i)} \tag{4.9}$$

where the term $\min(4,i)$ appears due to the fact the a maximum of 4 events per spill can be recorded. A beam spill length of 2200 ns was used in the Monte Carlo simulation. The systematic error on the correction was determined by varying the beam spill length from 1800 ns to 2600 ns and was found to be less than 0.1%. The statistics for the quantity $\mathrm{freq}_{obs}(i)$ were collected on a per run basis at the same time the electron counts were summed. Figure 4.13 shows the dead-time corrections used for $^{15}NH_3$ as a function of event rate (MAIN-OR triggers per spill).

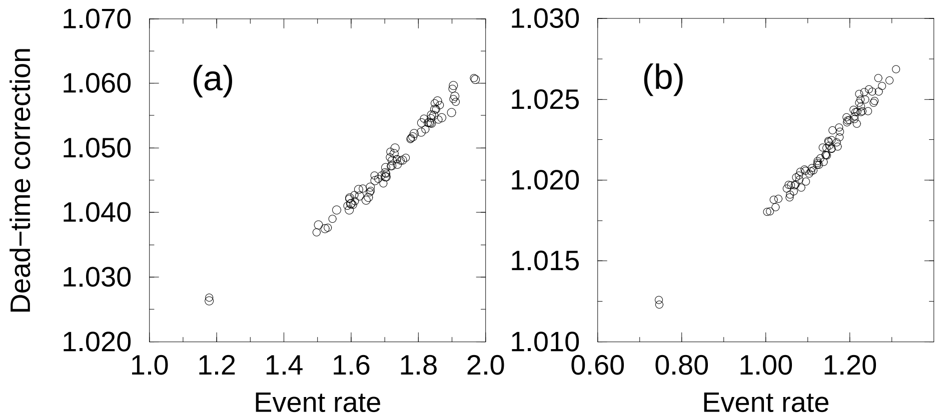

Figure 4.13: Dead-time corrections as a function of event rate (MAIN-OR triggers per spill) for $^{15}NH_3$ at (a) 4.5° and (b) 7.0°.

4.3.2 Nitrogen correction

A correction was required due to the polarization of the ^{15}N carried by the unpaired proton which is assumed to be in a $P_{1/2}$ state. Measurements were made after the standard runs to determine the ^{15}N polarization as a function of the proton and deuteron polarizations. The correction for the proton target was calculated using

$$C_1 = -\frac{1}{3}\frac{1}{3}\frac{P_N}{P_p}G_{\mathrm{EMC}} \tag{4.10}$$

where P_N is the ^{15}N polarization and P_p is the proton polarization. $G_{\mathrm{EMC}} = F_2^A/F_2^D$ is an EMC effect correction[55] where F_2^A and F_2^D are the F_2 structure functions for ^{15}N and the deuteron respectively. The first factor of $-\frac{1}{3}$ is from the Clebsch-Gordan coefficient relating the proton spin to the nitrogen polarization, and the second $\frac{1}{3}$ is due the ratio of 3 protons for every ^{15}N in the ammonia molecule. The value for C_1 was found to be 0.023 with a relative error of 20%.

For the deuteron target, the situation is more complicated. The ratio of the ^{15}N polarization to the deuteron polarization is much larger. The replacement of protons with deuterons was not complete with a contamination of about 1.5% $^{15}NH_3$ in $^{15}ND_3$. Also, both the proton and deuteron targets contained about 2% ^{14}N

contamination which introduces both a polarizable proton and neutron. For the proton target, this last contribution is negligible due to the smaller cross section and polarizability of the neutron. For the deuteron target, Equation 4.7 must be replaced with [56]

$$\Delta\sigma_{||} = C_1 [\Delta R - C_2 \Delta R^p] \frac{1}{\varepsilon \mathcal{D}} + \mathcal{S} \tag{4.11}$$

where C_1 and C_2 are the correction factors which depend on the polarizations and amount of contaminant polarizable protons and deuterons. The term ΔR^p is the rate result for the proton target calculation. The values for the correction factors for the deuteron target were $C_1 = 1.016$ and $C_2 = 0.04$.

4.3.3 Positron subtraction

The decay of neutral pions produced in the target creates photons which in turn create electron-positron pairs as they pass through the target. If an electron is created with high enough energy and within the physical acceptance of the spectrometer, it will pass all the spectrometer cuts. This not only adds a background to the cross section measurement but also the asymmetry since pair production may also have an asymmetry.

One can measure this process by reversing the fields of the magnets on the spectrometers and conducting special runs that measure only positrons. The rate of positrons will be in one-to-one correspondence to the electron rate contribution of the electrons from pair production. Subtracting the positron rate from the measured electron rate will then give the electron rate from scattered electrons only. For the 29 GeV and 16 GeV running in the deep-inelastic region, this contamination was significant and a correction was made.

At 9.7 GeV, the positron contamination was found to be nearly non-existent. The positron run for $^{15}NH_3$ which had 1.15×10^{15} incident electrons (twice that of a normal run) produced only a total of 25 counts over all the resonance region bins (< 4 GeV^2) for SP4. There were only 5 counts for SP7. No bin had a count rate difference of greater than 2. The positron run for $^{15}ND_3$ had similar results. A typical $^{15}NH_3$ electron run at half the incident charge has on the order of 250,000 counts in the resonance region bins. The lack of pions at low W^2 is understandable since this corresponds to low energy transfer and thus less energy is available for pion production.

In order to obtain statistically viable results, many hours of positron runs would be needed. With limited beam time, this was not worth the loss of statistics to the electron rate measurement. Therefore, in the resonance region analysis no positron subtraction was done. In any case, the correction is small.

4.3.4 The MC_ASK Monte Carlo program

The rest of the correction factors were obtained using a Monte Carlo program developed at Old Dominion University by S. Kuhn and F. Wesselman called MC_ASK[57]. The input for the Monte Carlo routine consisted of parameter and geometry files describing all target constituents, detector elements and magnetic fields. Histograms of model polarized cross sections were calculated using the routines from the radiative correction program (RCSLACPOL) developed at SLAC by L. Stuart for analysis of the E143 deep inelastic data[10–13]. These model cross sections were processed by MC_ASK to produce simulated electron counts that had been radiated, tested for acceptance, and resolution smeared. Comparison between input and output data then provided the desired corrections for radiative effects, resolution, acceptance, and dilution factor.

The cross section histograms produced by RCSLACPOL were binned in scattering angle θ and outgoing electron energy E'. For each spectrometer, four histograms were generated over the possible values of initial electron energy E: two for the cross sections and two for the parallel asymmetries of the reactions p(e,e$'$) and d(e,e$'$). The sets of histograms at different E values (100 histograms for $2.5 < E < 11.5$) were needed to handle incoming external radiative loss and nuclear motion.

Modifications were made to the RCSLACPOL routines used in the E143 deep inelastic analysis to add models for the resonance asymmetries and Fermi smearing. The resonance region was divided into three separate parts. The first region included bins with $W < 1.6$ GeV. This region contains the strongest resonances including the Δ, S_{11} and D_{13}. The second region contained the range $1.6 \leq W \leq 1.8$ GeV and the third included all bins with $W > 1.8$ GeV. This choice of kinematic regions was motivated both by the raw data and a parameterization of the resonance region used in calculations by Burkert and Li[58]. In the first region, two fit parameters were used corresponding to the asymmetry of the Δ resonance and the asymmetry off all other resonances (mostly S_{11} and D_{13}). A third fit parameter was used for the asymmetry of the resonance in the the third region. In the intermediate region, a simple linear interpolation between the first and third regions was used.

The simulated data were found to best fit the raw data for the proton target when $A_1 = -0.5$ for the Δ resonance, $A_1 = 1.5$ for the S_{11} and D_{13} resonances in the first region [3], and $A_1 = -1.0$ for the third region. For the deuteron target, the kinematic regions were slightly shifted with separations at $W = 1.5$ and $W = 1.7$ to give a better fit to the data. The deuteron D-state was accounted for by multiplying the average asymmetry of free protons and neutrons with the "canonical" correction factor $(1 - 1.5w_D) \approx 0.925$. The best fit was obtained using $A_1 = -0.5$ for the Δ resonance and $A_1 = 0.5$, and $A_1 = 0.0$ for the first and third regions respectively. Although this is a simplistic model, the data are described well enough to provide the desired corrections.

Unpolarized cross sections were calculated from a fit by Stuart *et al.*[59]. Non-resonant asymmetries were taken from a parametrization of all existing deep-inelastic data (Fit III of Ref. [12]), which was extrapolated into the resonance region.

Internal radiative effects were applied in RCSLACPOL following the prescription of Kukhto and Shumeiko[60, 61]. Internal radiative effects are due to higher order processes beyond the one-photon exchange shown in Fig. 2.1. The higher order processes included in RCSLACPOL are shown in Figure 4.14.

The largest correction is from internal bremsstrahlung (see Fig. 4.14a), in which the electron radiates a real photon due to interaction with the target nucleon's field. This interaction can happen before and/or after the scattering, changing the incoming and/or outgoing electron energy. In RCSLACPOL's calculation, the peaking approximation was used, which assumes that bremsstrahlung may not occur both before and after the scattering. This approximation changed a two dimensional integral over all possible incoming and outgoing electron energies into two one dimensional integrals, which vastly reduced the computational time. Results from a test run that calculated the full two dimensional integral showed no discernable difference from the peaking approximation.

The other internal radiative effects included in RCSLACPOL were vacuum polarization and the electron vertex correction. The contribution from the nucleon vertex and the two-photon exhange process were determined to be negligible.

The MC_ASK program was run in two different modes. In one mode the count differences for opposite spin alignments were obtained by simulating scattering only

[3] The fact that $A_1 = 1.5$ is above the theoretical limit of unity should not cause much concern. The unphysical value can be the result of a non-negligible A_2 contribution and/or inaccurate separation of resonant and non-resonant cross sections

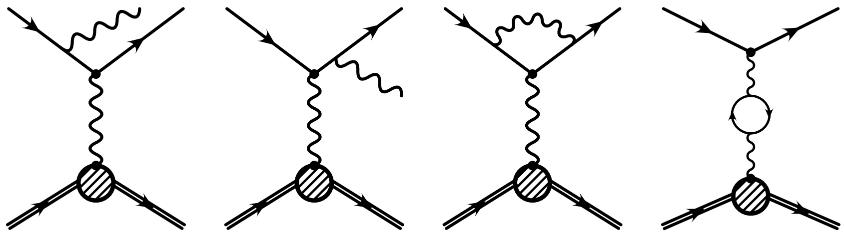

Figure 4.14: Internal radiative processes corrected for in the RCSLACPOL program.

on the polarized protons in the proton target or polarized deuterons in the deuteron target. In the second mode, the total unpolarized counts due to all target constituents were calculated. Fermi smearing was modeled using a flat momentum distribution up to 153 MeV/c. In each case, the same steps were followed to simulate the actual trajectory of an electron through the target and spectrometer. First, a scattering point was chosen randomly within the beam raster pattern. The target material traversed before the scattering was determined and the incoming external radiative loss calculated. Scattering angle and final electron energy were randomly chosen according to the appropriate cross section histogram from RCSLACPOL. The effects of multiple scattering and the target magnetic field were taken into account and the target thickness after the scattering point was used to determine the outgoing external radiative loss.

The path of the electron was then simulated through the spectrometer elements. If the electron reached the shower counter, several quantities were calculated and written to file. Each simulated event was then analyzed in essentially the same way as the real data to "reconstruct" the kinematic quantities and make histograms with identical binning to the data using the reconstructed Q^2 and W^2. The results were normalized to the actual target thickness and number of incident electrons of the experimental data.

The simulated data are in good agreement with the real data, indicating that both the acceptance of the spectrometers and the cross sections and asymmetries are well-modeled by the Monte Carlo code. The total unpolarized counts integrated over the resonance region agree with the data to better than 2.2% for SP4 and 3.4% for SP7 (see Figure 4.15. The measured and simulated polarized count rate differences (ΔR from Eq. 4.5) also agree well within statistical errors (see Figure

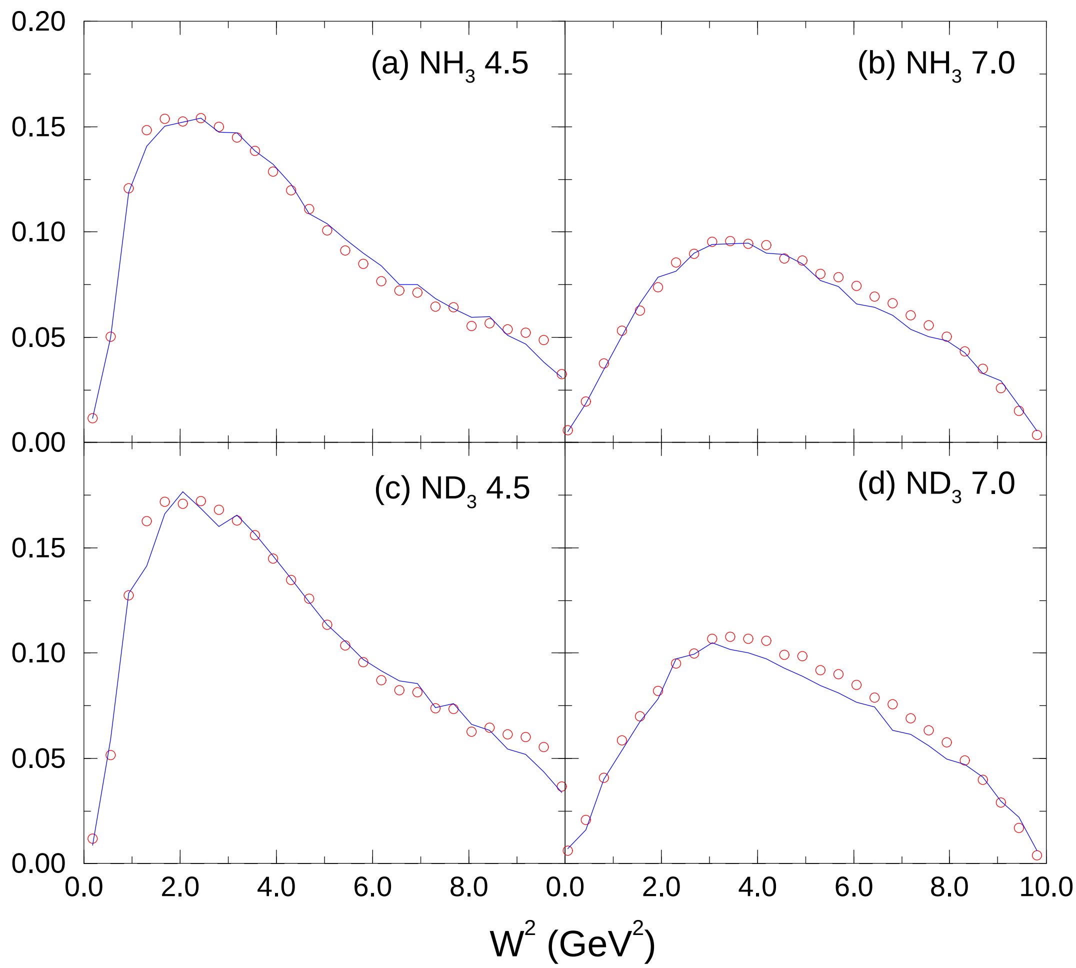

Figure 4.15: Comparison of data (diamonds) to Monte Carlo simulation (line) for unpolarized count rate from the entire target.

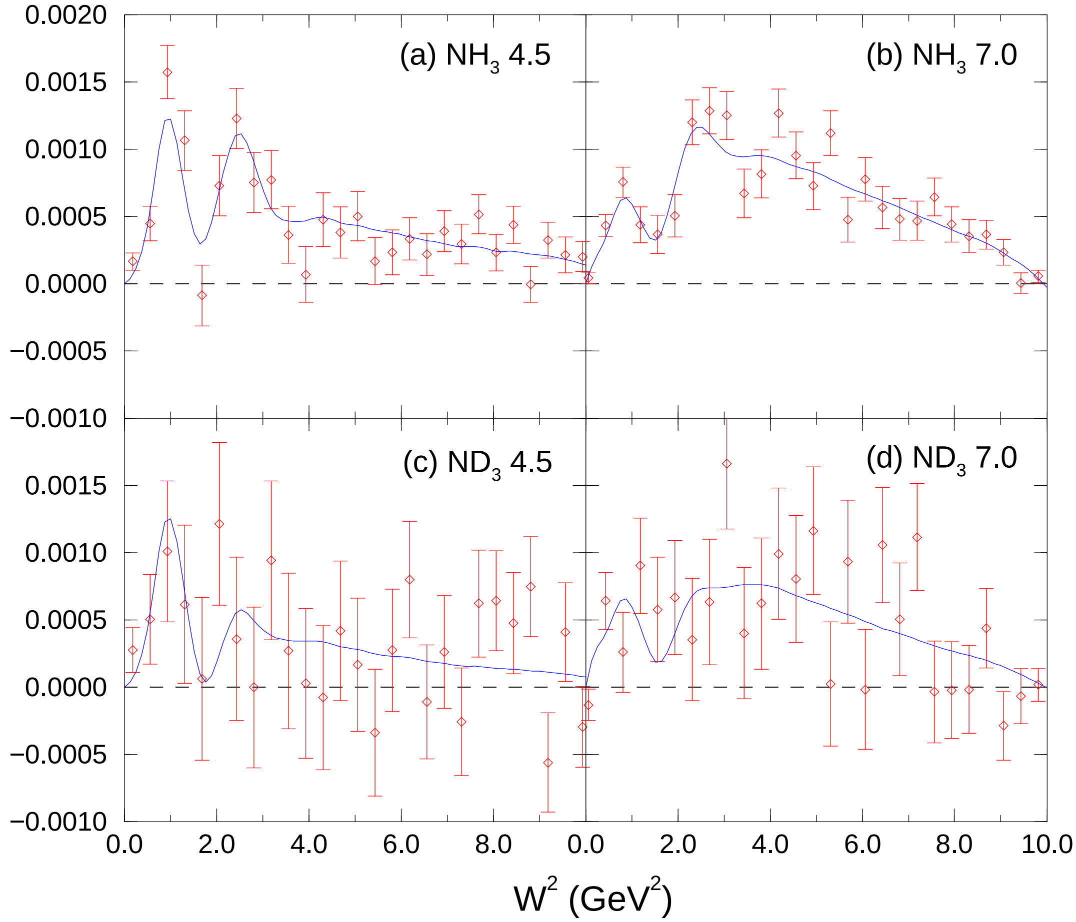

Figure 4.16: Comparison of data (diamonds) to Monte Carlo simulation (line) for polarized count rate difference (ΔR from Eq. 4.5). Error bars represent statistical error.

4.16). A χ^2 calculation over the (quasi-) elastic region ($W^2 < 1$ GeV^2), where model uncertainties are minimal, yielded (per degree of freedom) 1.424 for $^{15}NH_3$ SP4, 1.259 for $^{15}NH_3$ SP7, 0.020 for $^{15}ND_3$ SP4, and 1.005 for $^{15}ND_3$ SP7.

4.3.5 Dilution factor

The dilution factor is needed to correct the measured asymmetry for the counts that are not due to the the polarizable protons (deuterons) in the target. For the $^{15}NH_3$ target, the count rate asymmetry in Eq. 4.4 can be rewritten as

$$A_{rate} = \frac{R^{\downarrow\uparrow} - R^{\uparrow\uparrow}}{R^{\downarrow\uparrow} + R^{\uparrow\uparrow}} = \frac{R_p^{\downarrow\uparrow} - R_p^{\uparrow\uparrow}}{R_p^{\downarrow\uparrow} + R_{other}^{\downarrow\uparrow} + R_p^{\uparrow\uparrow} + R_{other}^{\uparrow\uparrow}} \qquad (4.12)$$

where R_p is the rate from the polarizable protons and R_{other} is the rate from all other target constituents. Since the other constituents are assumed to be unpolarized, their contribution to the numerator cancels. To remove the contribution of the other constituents from the denominator, one must divide A_{rate} by

$$f = \frac{R_p^{\downarrow\uparrow} + R_p^{\uparrow\uparrow}}{R_p^{\downarrow\uparrow} + R_{other}^{\downarrow\uparrow} + R_p^{\uparrow\uparrow} + R_{other}^{\uparrow\uparrow}}. \qquad (4.13)$$

This defines the dilution factor f.

The Monte Carlo program was used to calculate simulated cross sections for the polarizable protons only and for the full $^{15}NH_3$ target. Dividing the first quantity by the later then yielded the dilution factor. This was done for each bin in W^2. The only target input necessary was the ratio of the target densities for the free protons versus the total for all constituents. Since the dilution factor is applied before resolution and radiative corrections according to Eq. 4.6, it was calculated using simulated cross sections that were radiated and resolution smeared. Figure 4.17 shows the dilutions factors for the each spectrometer on the $^{15}NH_3$ target as a function of W^2. The periodic wiggles are due to counting statistics and spectrometer granularity rather than actual physical structure.

As mentioned at the start of this chapter, using a dilution factor for $^{15}ND_3$ calculated from MC_ASK to calculate $A_{||}$ for the deuteron was not considered worthwhile because current models for the deuteron in the resonance region are not sufficiently accurate. The purpose of the proton asymmetry analysis is to check the results obtained using the cross section difference. However, the dilution factor for $^{15}ND_3$

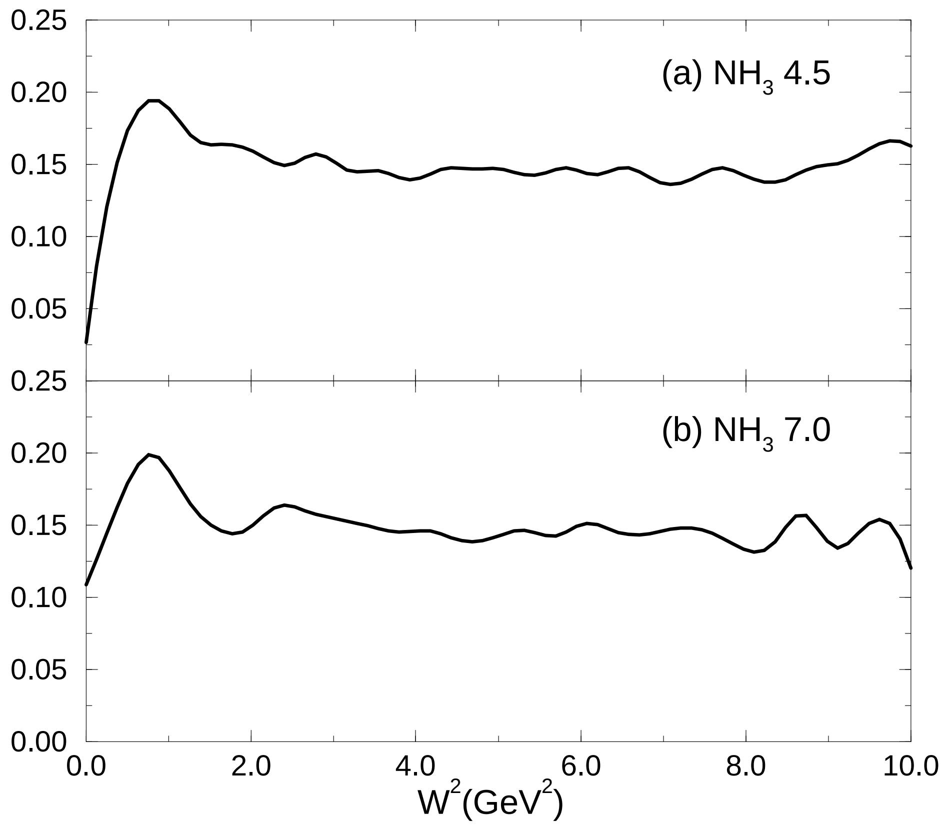

Figure 4.17: Dilution factor for $^{15}NH_3$ target as a function of W^2 for (a) SP4 and (b) SP7

was calculated in order to estimate the systematic error on the dilution factor for $^{15}NH_3$. Uncertainty in the target constituent ratio and uncertainty in the size of nuclear effects were the major source of error in the dilution factor. The relative systematic error was estimated to be 4.3% for SP4 and 3.6% for SP7.

4.3.6 Resolution and radiative corrections: $A_{||}$

After applying the above corrections to the measured asymmetry, one is left with an asymmetry that is radiated and resolution smeared. Both radiative effects and resolution smearing place counts in the wrong kinematic bin. The Monte Carlo program was run twice to calculate corrections to $A_{||}$. The first time radiative corrections and resolution smearing were turned off and a pure Born[4] asymmetry was found. The second time both effects were turned on and a radiated, resolution smeared asymmetry was calculated. The difference between the two for each W^2 bin was used as an additive correction on the measured asymmetry. From Eq. 4.6, one sees that $\mathcal{A}$ can be written as

$$\mathcal{A} = A_{||}^{\mathbf{Born}} - A_{||}^{\mathbf{RC}} \tag{4.14}$$

where $A_{||}^{\mathbf{Born}}$ is the Born asymmetry and $A_{||}^{\mathbf{RC}}$ is the radiated, resolution smeared asymmetry.

To determine the systematic error on the correction, the process was repeated, varying MC_ASK's input asymmetries by 50% and the width of the hodoscope finger's by 20%. The maximum difference of the results obtained were taken as an estimate of the systematic error.

4.3.7 Acceptance, target density, resolution and radiative corrections: $\Delta\sigma_{||}$

To convert from the rate difference, which has already been corrected for detector efficiency and ^{15}N, to a cross section difference, one must divide by target density and correct for spectrometer acceptance, radiative effects, and resolution. This was done using MC_ASK to calculate the ratio between the unpolarized radiated, resolution

[4]The term Born is used in the analysis to represent the base theoretical value from the one-photon exchange diagram. No radiative effects or corrections related to the spectrometer are applied.

smeared count rate, $R^{\mathbf{RC}}$, and the unpolarized Born cross section, $\sigma^{\mathbf{Born}}$. Therefore, $\mathcal{D}$ from Eq. 4.7 can be expressed as

$$\mathcal{D} = \frac{R^{\mathbf{RC}}}{\sigma^{\mathbf{Born}}} \tag{4.15}$$

The uncertainty of this factor was estimated by comparing the integrated measured total count rate from data to that from the MC_ASK program for each bin. The percentage difference was added in quadrature to the error in the absolute amount of polarized target material (3% for $^{15}NH_3$, 5% for $^{15}ND_3$).

The cross section difference is obtained by dividing the rate difference by this factor. However, this factor only corrects for the unpolarized part of the radiative effects and resolution smearing. A correction for the polarized part was made by adding $\mathcal{A}$ from above multiplied by the unpolarized Born cross section. Therefore, $\mathcal{S}$ from Eq. 4.7 can be expressed as

$$\mathcal{S} = \mathcal{A}\,\sigma^{\mathbf{Born}} \tag{4.16}$$

The uncertainty of this correction was estimated using the same procedure as for $\mathcal{A}$ in the previous section.

Chapter 5

Results and Conclusions

This chapter presents the results from the resonance region analysis: the spin structure functions g_1^p and g_1^d, the virtual photon-nucleon asymmetry A_1^p, as well as integrals of Γ_1^p and Γ_1^d. Also, the integral for the neutron, Γ_1^n, was calculated from the proton and deuteron results. The integrals are plotted as a function of Q^2 along with results from DIS experiments at higher Q^2. The Q^2 evolution is compared to various theoretical predictions.

5.1 Spin Structure Functions g_1^p and g_1^d

The cross section difference $\Delta\sigma_{||}$ can be expressed in terms of g_1 and g_2 by substituting Eqs. 2.17 and 2.18 into 2.9 which gives

$$\Delta\sigma_{||} = \frac{4\sigma_{Mott}}{M\nu}\tan^2\frac{\theta}{2}\left[(E + E'\cos\theta)g_1 - \frac{Q^2}{\nu}g_2\right] \tag{5.1}$$

Using Eqs. 2.31 and 2.32, $\Delta\sigma_{||}$ can also be expressed in terms of g_1 and A_2,

$$\Delta\sigma_{||} = \frac{4\sigma_{Mott}}{M\nu}\tan^2\frac{\theta}{2}\left[(E + E'\cos\theta + \frac{Q^2}{\nu})g_1 - QF_1A_2\right] \tag{5.2}$$

Since only $\Delta\sigma_{||}$ was measured in the resonance region at 9.7 GeV, in order to extract g_1, assumptions need to be made about either g_2 or A_2. The positivity limit requires that $|A_2| \leq \sqrt{R}$. Existing data[62] suggests that R is small in the resonance region, $R = 0.06 \pm 0.02$ for $1 < Q^2 < 8$ GeV^2 and $W^2 < 3$ GeV^2. Extrapolation of R due to deep-inelastic scattering into the resonance region indicates that the contribution to the value of R from the resonances themselves is probably smaller than 0.06. For this reason we assume $A_2 = 0$ (corresponding to $g_1 = -g_2$) for this analysis. The alternate possibilities $g_2 = 0$ and $g_2 = g_2^{\mathrm{WW}}$ were considered to determine the uncertainty in g_1, due to lack of knowledge of A_2. Even in the extreme case of $A_2 = 0.3$ (i.e. if $R = 0.09$ and $A_2 = \sqrt{R}$), the extracted values of g_1 are shifted by less than 0.014, which is small compared to the statistical error on each point.

Using Eq. 5.2 with $A_2 = 0$, values for g_1^p and g_1^d were calculated for both spectrometers from the measured cross section differences. As defined in Eq. 2.43, g_1^d is the structure function per nucleon. Therefore, in extracting g_1^d from Eq. 5.2, the cross section difference was first divided by two.

The results are plotted in Figure 5.1 (circles) and tabulated in the Appendix (Tables B.1–B.4). The full length of the error bars corresponds to the statistical and systematic uncertainties added in quadrature. The cross bars indicate the statistical errors alone, which dominate the total.

The triangles in the plots for g_1^p represent the data of Baum *et al.*[2] which were taken at SLAC at similar kinematics and converted to g_1 by assuming $A_2 = 0$. Within errors, the two measurements agree well. The results are clearly negative in the area of the $\Delta(1232)$ resonance at $W^2 \approx 1.5$ GeV^2 and strongly positive just above $W^2 = 2$ GeV^2 where the S_{11} and D_{13} resonance are important. This peak is less pronounced for the deuteron.

All plots include curves for our Monte Carlo simulation (solid line) and the AO model of Burkert and Li[58](dashed line) which comes from a parameterization of previous photon-induced and electron-induced single pion production data. One of the reasons that the AO model does not accurately reproduce the data is that it does not include decay channels other than single-π emission such as multi-pion or eta production. The deuteron results lack sufficient statistical significance to draw any conclusions about their agreement with models.

The contributions to the systematic error on g_1 considered were:

Data:

Uncertainties in the beam and target polarization, cut efficiencies, dead-time, beam charge calibration, and ^{15}N correction combined for a total relative error on the measured count rate difference of 5.5% for $^{15}NH_3$ and 6.7% for $^{15}ND_3$

Acceptance/Target:

Uncertainty in the spectrometer acceptance was estimated by comparing the integrated measured total count rate from the data to that from the MC_ASK

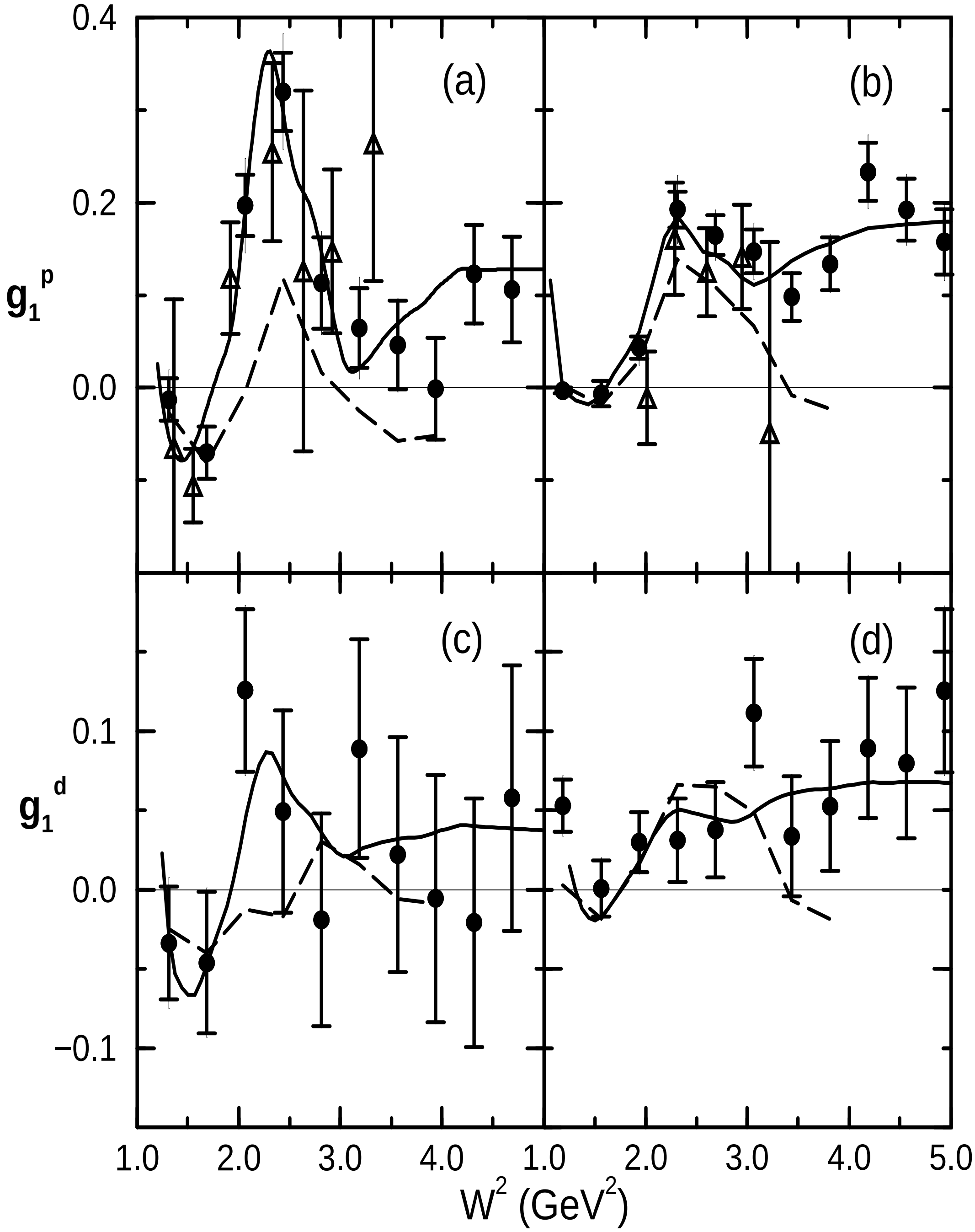

Figure 5.1: Results for g_1 in the resonance region as a function of W^2 for the proton at (a) 4.5° and (b) 7.0° and for the deuteron at (c) 4.5° and (d) 7.0°. The present data (circles) are plotted together with the data of Baum *et al.* [2] (triangles), our Monte Carlo simulation (solid line), and the model of AO of Burkert and Li [58] (dashed line). The full error bars correspond to statistical and systematic errors added in quadrature, while the cross bars indicate statistical error only.

program. The percentage difference was added in quadrature to the error in the absolute amount of polarized target material (3% for $^{15}NH_3$, 5% for $^{15}ND_3$) and used as a relative error on the correction factor $\mathcal{D}$.

Kinematic Variables θ and E':

Systematic errors in the measurement of the kinematic variables E' and θ were estimated by varying both the central spectrometer angle by 2% and the central ray by 6-7 mm in the first two hodoscopes.

Resolution:

The uncertainty from resolution effects were estimated by changing the hodoscope resolution ("finger width") by 20%.

Radiative Corrections:

The uncertainty due to radiative effect calculation was estimated by varying the free parameters of the input asymmetry by 50%.

A_2 Model:

The uncertainty in g_1 due to the model assumption that $A_2 = 0$ was estimated by making two alternate assumptions, $g_2 = 0$ and $g_2 = g_2^{\mathrm{WW}}$.

For each category, the listed change(s) were made to the specified input quantity and g_1 recalculated. The absolute difference between the recalculated value of g_1 and the base g_1 was taken as the absolute systematic error on g_1 due to that quantity. In the case several changes were tried for a category, the one resulting in the largest difference from the base g_1 was taken for the error. The errors form each category were combined in quadrature to give the overall systematic error. The results are tabulated in the Appendix in Tables B.7–B.10.

5.2 Virtual photon-proton asymmetry $A_1 + \eta A_2$

As shown in Eq. 2.37, the parallel asymmetry $A_{||}$ is related to the virtual photon-nucleon asymmetries by

$$A_{||} \;=\; D(A_1 + \eta A_2) \tag{5.3}$$

where $D = (1 - E'\epsilon/E)/(1 + \epsilon R)$ is the depolarization factor. To calculate D we set $R = 0.1$, which is consistent with its maximum estimated value in the resonance region as discussed in Section 5.1. The factors ϵ and η are both of the order unity in the resonance region.

We make no assumption about A_2 but simply calculate the values for the quantity $A_1 + \eta A_2$ for the proton from the measured parallel asymmetry $A_{||}$. The results are plotted in Figure 5.2 (circles) and tabulated in the Appendix (Tables B.5 and B.6). The full length of the error bars corresponds to the statistical and systematic uncertainties added in quadrature. The cross bars indicate the statistical errors alone, which dominate the total. The triangles in the figure correspond to the Baum *et al.*[2] data which are in in good agreement with our results. The solid line is from our Monte Carlo simulation.

The contributions to the systematic error on $A_1 + \eta A_2$ were determined in the same manner as the systematic errors on g_1. The uncertainty categories considered were:

Data:

Uncertainties in the beam and target polarization, dead-time, beam charge calibration, and ^{15}N correction combined for a total relative error of 4.5% on the measured raw asymmetry.

Dilution factor:

Uncertainty in dilution factor was estimated by combining the variation of the dilution factor calculation for $^{15}ND_3$ over the resonance region with the the uncertainties in the target material fractions and a 2.2% error from nuclear models (EMC effect and Fermi smearing). The relative systematic error on $A_{||}$ was estimated to be 4.3% for SP4 and 3.6% for SP7.

Kinematic Variables θ and E':

Systematic errors in the measurement of the kinematic variables E' and θ were estimated by varying both the central spectrometer angle by 2% and the central ray by 6-7 mm in the first two hodoscopes.

Resolution:

The uncertainty from resolution effects were estimated by changing the hodoscope resolution ("finger width") by 20%.

Radiative Corrections:

The uncertainty due to radiative effect calculation was estimated by varying the free parameters of the input asymmetry by 50%.

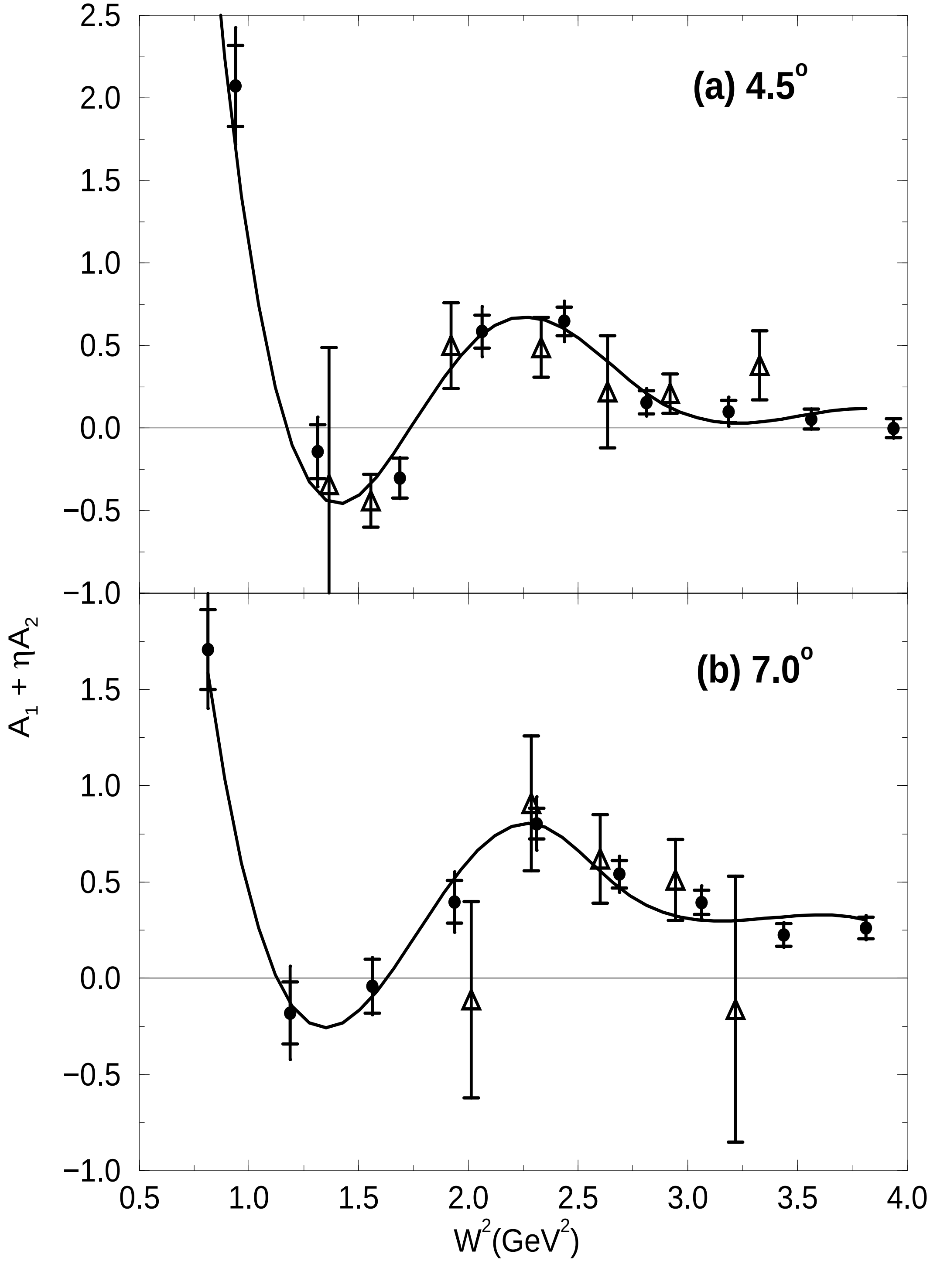

Figure 5.2: Results for $A_1 + \eta A_2$ in the resonance region as a function of W^2 for the proton at (a) 4.5° and (b) 7.0°. The present data (circles) are plotted together with the data of Baum *et al.* [2] (triangles), and our Monte Carlo simulation (solid line). Errors are indicated as in Fig. 5.1.

R Model:

The uncertainty due assumption that $R = 0.1$ was estimated by varying this value ± 0.1.

For each category, the listed change(s) were made to the specified input quantity and $A_1 + \eta A_2$ recalculated. The absolute difference between the recalculated value and the base $A_1 + \eta A_2$ was taken as the absolute systematic error due to that quantity. In the case several changes were tried for a category, the one resulting in the largest difference from the base $A_1 + \eta A_2$ was taken for the error. The errors form each category were combined in quadrature to give the overall systematic error. The results are tabulated in the Appendix in Tables B.11 and B.12.

As a cross check, we converted $A_{||}$ to $\Delta\sigma_{||}$ according to Eq. 2.13 using the unpolarized cross section calculated by the Monte Carlo simulation. Then g_1 was calculated as in the previous section. The results are shown in Figure 5.3 in which the circles represent g_1 from $\Delta\sigma_{||}$ and the squares represent g_1 from $A_{||}$. The agreement of the two methods is better than 3% on average.

5.3 Integrals Γ_1^p, Γ_1^d, and Γ_1^n

In Section 2.5, two sum rules were described which predict values for the integrals Γ_1^p and Γ_1^d: the Ellis-Jaffe sum rule at $Q^2 \to \infty$ and the GDH sum rule at $Q^2 = 0$. Several recent papers[38–40, 63, 64] have dealt with the question of how the integrals evolve from one limit to the other. Of key interest is the effect of the nucleon's resonances in this evolution.

Our results in the resonance region can be used to estimate values for these integrals at two comparatively low Q^2 points. We integrated our measured g_1 in the resonance region over x using the bins with $1 < W^2 < 4$ GeV^2 which covered $x > 0.125$ for SP4 and $x > 0.255$ for SP7. The lower limit on W^2 excluded the elastic peak. The average Q^2 of these bins is approximately 0.5 ± 0.05 GeV^2 for SP4 and 1.2 ± 0.1 GeV^2 for SP7. The contribution to the integral from g_1 at $x < 0.125$ was taken from the Q^2-dependent parameterization (Fit III) of the world data in Ref. [12]. The integrals for Γ_1^n were obtained from the proton and deuteron results according to Eq. 2.43 using a D-state probability of 5%.

The statistical error on the combined result was calculated from the entire data set at 9.71 GeV which was analyzed up to $W^2 = 10$ ($x \approx 0.03$). This is a valid estimate of the statistical uncertainty since the Q^2-dependent fit is mostly constrained

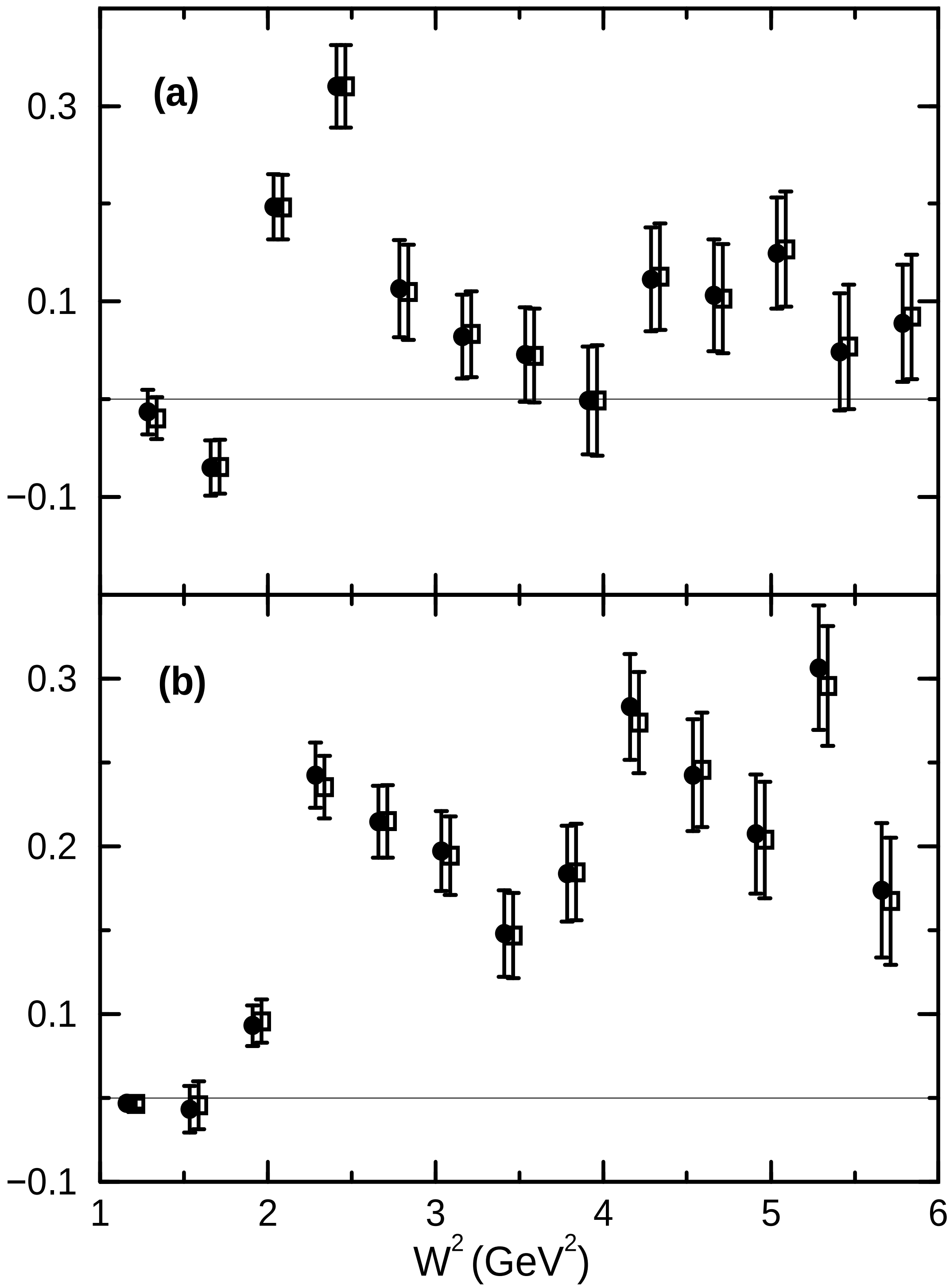

Figure 5.3: Comparison of g_1^p results extracted from cross section differences (circles) and asymmetries (squares) as a function of W^2 at (a) 4.5° and (b) 7.0°. The plotted W^2 values are slightly shifted to separate the two data sets. Only statistical errors are shown.

in the Q^2 region of interest (0.5 - 1.2 GeV^2) by the same 9.71 GeV data set.

The contribution to the systematic error from the resonance region data was determined from the systematic error of the individual g_1 points. The absolute errors on g_1 from each of the systematic uncertainty categories listed in Section 5.1 were integrated over the resonance region and then added together in quadrature. For the fit region, the systematic errors of the data points contributing most strongly to the fit were added together linearly (they are strongly correlated). An additional error was added due to the extrapolation to zero from the last measured datum at $x = 0.03$. This error was estimated to be equal to the value the fit yields for the integral below $x < 0.03$.

Q^2 (GeV^2)		Γ_1^{res} ($\pm$stat.$\pm$syst.)	Γ_1^{tot} ($\pm$stat.$\pm$syst.)
0.5	p	0.026$\pm$0.008$\pm$0.008	$0.049 \pm 0.008 \pm 0.013$
0.5	d	0.000$\pm$0.013$\pm$0.008	$0.003 \pm 0.013 \pm 0.010$
0.5	n		$-0.043 \pm 0.029 \pm 0.025$
1.2	p	0.040$\pm$0.003$\pm$0.004	$0.100 \pm 0.005 \pm 0.012$
1.2	d	0.026$\pm$0.006$\pm$0.004	$0.043 \pm 0.008 \pm 0.007$
1.2	n		$-0.006 \pm 0.019 \pm 0.020$

Table 5.1: Integrals Γ_1 of the structure functions g_1 for the proton (p), deuteron (d) and neutron (n). The measured sum Γ_1^{res} for the resonance region ($W^2 < 4$ GeV^2) is listed separately from the totals Γ_1^{tot}, which includes the deep-inelastic region as given by fits to the world's data.

Table 5.1 lists the results of the integral calculations for the proton, deuteron and neutron. Values for the proton and neutron are plotted as circles in Figure 5.4. Full error bars represent the combined statistical and systematic errors while the cross bars indicate the statistical error only. Data at higher Q^2 from previous deep-inelastic experiments are also plotted which include E143[10, 11, 13] (squares), CERN[5–8] (triangles), and E142[9] (inverted triangles). The solid curve at high Q^2 corresponds to an evolution of the deep-inelastic results due to changing α_s[35] (see Eq. 2.59). The solid curve at low Q^2 is the GDH slope (see Eq. 2.64).

The dotted line shows the predictions of Burkert and Ioffe[39, 40] which combines the contributions from the resonance using the AO code[58] with the nonresonant contributions using a simple higher-twist-type form fitted to the deep-inelastic data. Their model is constrained to fit both the GDH and deep-inelastic limits and.

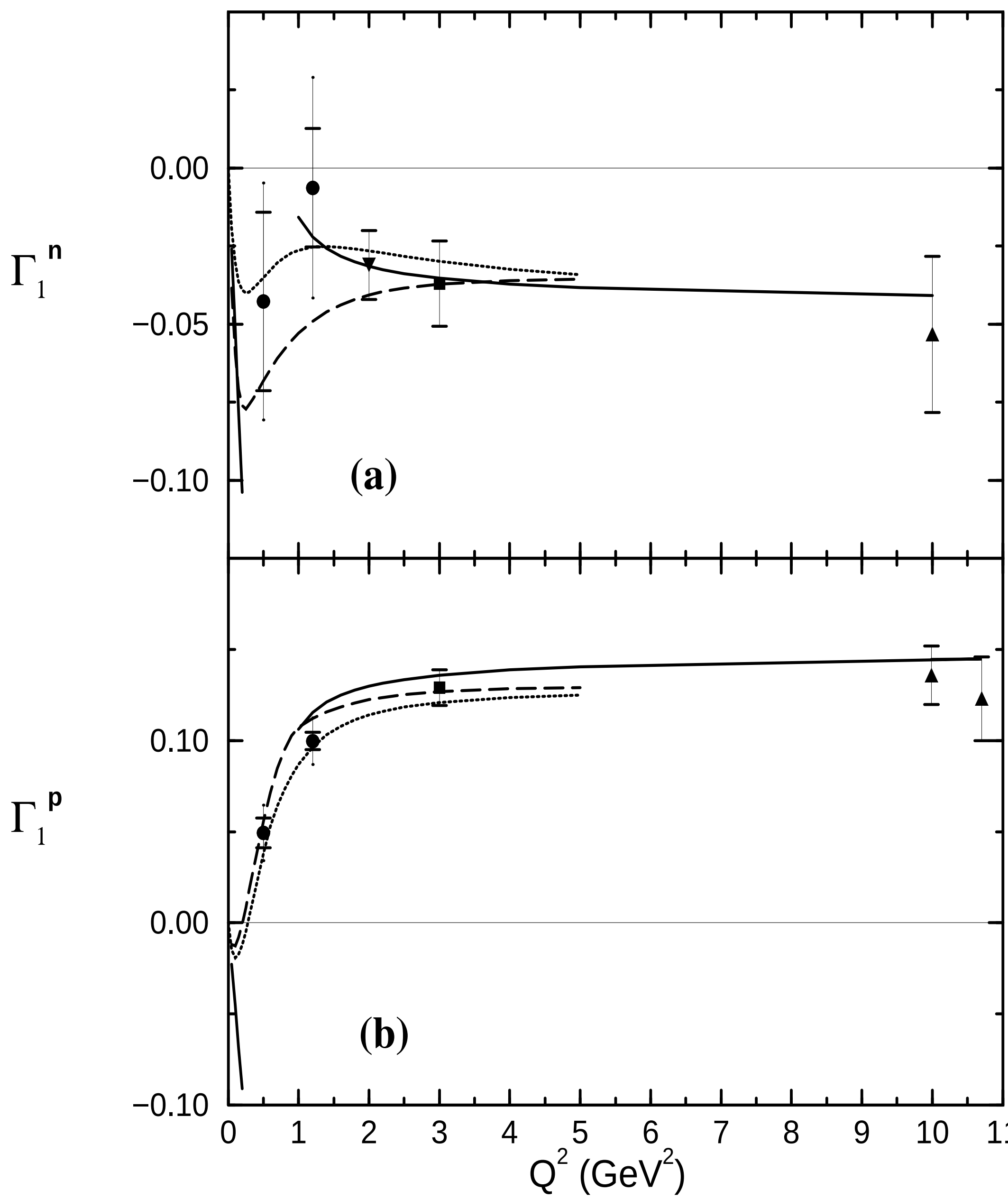

Figure 5.4: Integrals of g_1 at several fixed values of Q^2 for (a) the neutron and (b) the proton. The results of this analysis (circles) are plotted together with data from CERN[5–8] (triangles), E143 deep-inelastic[10, 11, 13] (squares), and E142[9] (inverted triangle). The curves correspond to the evolution[29] of the deep-inelastic results due to changing α_s (solid line), the predictions of Burkert and Ioffe[39, 40] (dotted line), the model of Soffer and Teryaev[38](dashed line), and the GDH approach to $Q^2 = 0$ (solid line). Errors are indicated as in Fig. 5.1.

The dashed curve represents the model by Soffer and Teryaev[38] which states that whereas the integral over g_1 for the proton has a strong Q^2 dependence at low Q^2, the integral over $g_1 + g_2$ should vary smoothly from high Q^2 where $g_2 \approx 0$ down to $Q^2 = 0$. Using their simple prediction for this integral and subtracting the contribution from g_2 alone using the Burkhardt-Cottingham sum rule[37]. Both models agree quite well with our data.

5.4 Conclusions

The g_1^p and g_1^d spin structure functions calculated from the E143 resonance region data allow us for the first time to determine the integrals $\Gamma_1^p(Q^2)$ and $\Gamma_1^n(Q^2)$ at Q^2 below 2 GeV^2. The integrals in this region are observed to vary rapidly in contrast to the nearly flat behavior in the deep-inelastic region. Recent models that interpolate in the non-perturbative region between the the deep-inelastic and GDH limits describe the data well. More precise data over a larger range of Q^2 in the non-perturbative region will be taken in the future by several experiments scheduled for Hall B at TJNAF[65–67], MAMI and HERMES (HERA).

Appendix A

The E143 Collaboration

K. Abe,[15] T. Akagi,[12,15] P. L. Anthony,[12] R. Antonov,[11] R. G. Arnold,[1] T. Averett,[16,‡‡]
H. R. Band,[18] J. M. Bauer,[7] H. Borel,[5] P. E. Bosted,[1] V. Breton,[3] J. Button-Shafer,[7]
J. P. Chen,[16,8] T. E. Chupp,[8] J. Clendenin,[12] C. Comptour,[3] K. P. Coulter,[8] G. Court,[12,*]
D. Crabb,[16] M. Daoudi,[12] D. Day,[16] F. S. Dietrich,[6] J. Dunne,[1] H. Dutz,[12,**] R. Erbacher,[12,13]
J. Fellbaum,[1] A. Feltham,[2] H. Fonvieille,[3] E. Frlez,[16] D. Garvey,[9] R. Gearhart,[12] J. Gomez,[4]
P. Grenier,[5] K. A. Griffioen,[11,17] S. Hoibraten,[16,§] E. W. Hughes,[12,‡‡] C. E. Hyde-Wright,[10]
J. R. Johnson,[18] D. Kawall,[13] A. Klein,[10] S. E. Kuhn,[10] M. Kuriki,[15] R. Lindgren,[16] T. J. Liu,[16]
R. M. Lombard-Nelsen,[5] J. Marroncle,[5] T. Maruyama,[12] X. K. Maruyama,[9] J. McCarthy,[16]
W. Meyer,[12,**] Z.-E. Meziani,[13,14] R. Minehart,[16] J. Mitchell,[4] J. Morgenstern,[5]
G. G. Petratos,[12,‡] R. Pitthan,[12] D. Pocanic,[16] C. Prescott,[12] R. Prepost,[18] P. Raines,[11]
B. A. Raue,[10,†] D. Reyna,[1] A. Rijllart,[12,††] Y. Roblin,[3] L. S. Rochester,[12] S. E. Rock,[1]
O. A. Rondon,[16] I. Sick,[2] L. C. Smith,[16] T. B. Smith,[8] M. Spengos,[1,11] F. Staley,[5] P. Steiner,[2]
S. St.Lorant,[12] L. M. Stuart,[12] F. Suekane,[15] Z. M. Szalata,[1] H. Tang,[12] Y. Terrien,[5] T. Usher,[12]
D. Walz,[12] F. Wesselmann,[10] J. L. White,[1,12] K. Witte,[12] C. C. Young,[12] B. Youngman,[12]
H. Yuta,[15] G. Zapalac,[18] B. Zihlmann,[2] D. Zimmermann[16]

[1]*The American University, Washington, D.C. 20016*
[2]*Institut für Physik der Universität Basel, CH–4056 Basel, Switzerland*
[3]*LPC IN2P3/CNRS, University Blaise Pascal, F–63170 Aubiere Cedex, France*
[4]*TJNAF, Newport News, Virginia 23606*
[5]*DAPNIA-Service de Physique Nucleaire Centre d'Etudes de Saclay, F–91191 Gif/Yvette, France*
[6]*Lawrence Livermore National Laboratory, Livermore, California 94550*
[7]*University of Massachusetts, Amherst, Massachusetts 01003*
[8]*University of Michigan, Ann Arbor, Michigan 48109*
[9]*Naval Postgraduate School, Monterey, California 93943*
[10]*Old Dominion University, Norfolk, Virginia 23529*
[11]*University of Pennsylvania, Philadelphia, Pennsylvania 19104*
[12]*Stanford Linear Accelerator Center, Stanford, California 94309*
[13]*Stanford University, Stanford, California 94305*
[14]*Temple University, Philadelphia, Pennsylvania 19122*
[15]*Tohoku University, Sendai 980, Japan*
[16]*University of Virginia, Charlottesville, Virginia 22901*
[17]*The College of William and Mary, Williamsburg, Virginia 23187*
[18]*University of Wisconsin, Madison, Wisconsin 53706*

Appendix B

Data Tables

This appendix contains tables of results for g_1^p, g_1^d, and $A_1 + \eta A_2$ (proton) for each bin used in the analysis. Included in the tables are many of the correction and raw data terms used to obtain the final values. The headings on the table columns refer to Equations 4.6 and 4.7. The one correction not included in the tables is the ^{15}N correction which does not vary by bin. The reader is referred to Section 4.3.2 for the appropriate values.

W^2	Q^2	$\overline{\Delta R}$	ε	$\mathcal{D}$	$\mathcal{S}$	$\Delta\sigma$	g_1
0.94	0.56	14.56±1.84	0.924	0.54	191	3027±358	0.315±0.037±0.050
1.31	0.55	9.85±2.04	0.923	1.63	-714	-74±133	-0.013±0.023±0.023
1.69	0.54	-0.79±2.07	0.923	1.89	-244	-289±116	-0.070±0.028±0.006
2.06	0.53	6.73±2.06	0.922	2.07	282	627±106	0.197±0.033±0.039
2.44	0.52	11.34±2.07	0.920	2.01	229	828±109	0.320±0.042±0.046
2.81	0.50	6.97±2.05	0.924	2.02	-120	245±108	0.113±0.050±0.025
3.19	0.49	7.16±2.01	0.923	2.65	-166	120±80	0.064±0.043±0.034
3.56	0.48	3.38±1.97	0.923	2.67	-59	75±78	0.046±0.048±0.015
3.94	0.47	0.65±1.90	0.922	2.54	-29	-2±79	-0.001±0.055±0.010
4.31	0.46	4.41±1.83	0.922	2.84	-7	158±68	0.123±0.053±0.015
4.69	0.45	3.53±1.76	0.920	2.81	-10	124±66	0.106±0.057±0.010
5.06	0.43	4.62±1.67	0.917	2.95	-8	158±60	0.149±0.057±0.013
5.44	0.42	1.56±1.59	0.916	2.92	-10	47±58	0.049±0.060±0.009
5.81	0.41	2.17±1.53	0.915	3.06	-6	69±53	0.078±0.060±0.008
6.19	0.40	3.06±1.45	0.914	3.00	-8	101±52	0.123±0.063±0.014
6.56	0.39	2.01±1.41	0.914	2.97	-7	65±51	0.085±0.066±0.009
6.94	0.38	3.58±1.40	0.913	2.97	-5	124±50	0.174±0.071±0.019
7.31	0.37	2.71±1.33	0.908	2.91	-5	95±49	0.143±0.074±0.013
7.69	0.35	4.72±1.33	0.911	3.07	-6	159±47	0.255±0.074±0.042
8.06	0.34	2.10±1.23	0.899	2.89	-4	75±46	0.127±0.079±0.016
8.44	0.33	3.98±1.24	0.902	3.04	-4	137±44	0.248±0.080±0.020
8.81	0.32	-0.02±1.20	0.889	2.92	-5	-6±45	-0.011±0.086±0.004
9.19	0.31	2.86±1.18	0.878	2.90	-4	106±45	0.214±0.092±0.029
9.56	0.30	1.75±1.09	0.810	2.67	-5	74±49	0.159±0.105±0.036
9.94	0.28	1.61±0.88	0.791	2.30	-5	82±47	0.185±0.107±0.037

Table B.1: Cross section difference $\Delta\sigma$ (nb/GeV-sr) for $^{15}NH_3$ and structure function g_1^p at 4.5°. The values of W^2 and Q^2 (GeV^2) are given at bin centers. $\overline{\Delta R}$ (see Eq. 4.5) represents the experimental average raw count rate difference ($\times 10^{-4}$ counts/gigaelectron). Efficiency ε is absolute. Correction terms $\mathcal{D}$ ($\times 10^{-6}$) and $\mathcal{S}$ are from Eq. 4.7.

W^2	Q^2	$\overline{\Delta R}$	ε	$\mathcal{D}$	$\mathcal{S}$	$\Delta\sigma$	g_1
0.81	1.31	6.78±1.00	0.867	4.13	35	215±27	0.109±0.013±0.021
1.19	1.28	3.95±1.19	0.867	28.87	-19	-4±5	-0.003±0.003±0.003
1.56	1.26	3.29±1.29	0.866	8.65	-49	-8±16	-0.007±0.014±0.005
1.94	1.23	4.51±1.39	0.863	12.77	4	43±12	0.043±0.012±0.015
2.31	1.20	10.63±1.49	0.859	10.01	45	162±16	0.193±0.020±0.031
2.69	1.18	11.31±1.52	0.853	10.69	3	121±16	0.165±0.022±0.016
3.06	1.15	10.93±1.57	0.847	11.45	-12	95±15	0.147±0.024±0.019
3.44	1.12	5.86±1.57	0.844	11.83	1	57±15	0.098±0.026±0.009
3.81	1.10	7.07±1.55	0.839	11.80	2	70±15	0.134±0.029±0.012
4.19	1.07	10.94±1.54	0.836	11.78	5	110±15	0.233±0.031±0.025
4.56	1.04	8.18±1.48	0.830	11.79	4	83±14	0.193±0.033±0.020
4.94	1.02	6.23±1.48	0.829	12.03	3	63±14	0.158±0.036±0.022
5.31	0.99	9.46±1.41	0.820	12.02	3	94±14	0.257±0.037±0.027
5.69	0.96	4.00±1.39	0.813	11.89	3	42±14	0.124±0.040±0.015
6.06	0.93	6.46±1.35	0.806	11.87	3	67±13	0.210±0.042±0.027
6.44	0.91	4.71±1.30	0.805	11.72	2	50±13	0.168±0.044±0.018
6.81	0.88	4.02±1.28	0.811	11.50	2	43±13	0.155±0.047±0.022
7.19	0.85	3.97±1.23	0.819	11.05	2	44±13	0.167±0.050±0.022
7.56	0.83	5.49±1.18	0.824	10.60	2	62±13	0.251±0.052±0.033
7.94	0.80	3.84±1.13	0.839	9.88	2	46±13	0.198±0.056±0.022
8.31	0.77	3.17±1.07	0.864	9.22	1	39±13	0.180±0.058±0.025
8.69	0.75	3.27±0.96	0.863	8.51	1	43±12	0.210±0.060±0.021
9.06	0.72	2.02±0.81	0.836	6.80	1	35±14	0.180±0.069±0.021
9.44	0.69	0.05±0.59	0.767	4.62	1	2±16	0.012±0.085±0.006
9.81	0.67	0.30±0.23	0.509	1.52	2	38±28	0.217±0.159±0.105

Table B.2: Cross section difference $\Delta\sigma$ (nb/GeV-sr) for $^{15}NH_3$ and structure function g_1^p at 7.0°. The values of W^2 and Q^2 (GeV^2) are given at bin centers. $\overline{\Delta R}$ (see Eq. 4.5) represents the experimental average raw count rate difference ($\times 10^{-4}$ counts/gigaelectron). Efficiency ε is absolute. Correction terms $\mathcal{D}$ ($\times 10^{-6}$) and $\mathcal{S}$ are from Eq. 4.7.

W^2	Q^2	$\overline{\Delta R}$	ε	$\mathcal{D}$	$\mathcal{S}$	$\Delta\sigma$	g_1
0.94	0.56	9.34±4.82	0.923	0.57	318	2012±932	0.210±0.097±0.034
1.31	0.55	5.71±5.44	0.923	1.45	-794	-389±414	-0.067±0.071±0.042
1.69	0.54	0.60±5.59	0.923	1.67	-421	-379±369	-0.092±0.089±0.030
2.06	0.53	11.22±5.57	0.921	1.88	160	801±326	0.252±0.102±0.032
2.44	0.52	3.32±5.59	0.920	1.87	86	255±330	0.099±0.128±0.013
2.81	0.50	0.00±5.53	0.922	2.10	-67	-82±290	-0.038±0.134±0.010
3.19	0.49	8.72±5.45	0.923	2.34	-65	331±256	0.178±0.138±0.025
3.56	0.48	2.51±5.33	0.924	2.43	-35	73±241	0.045±0.148±0.008
3.94	0.47	0.28±5.14	0.923	2.52	-27	-16±224	-0.011±0.156±0.005
4.31	0.46	-0.70±4.95	0.922	2.70	-18	-53±202	-0.041±0.157±0.005
4.69	0.45	3.87±4.78	0.921	2.71	-17	135±195	0.116±0.167±0.013
5.06	0.43	1.56±4.54	0.919	2.78	-19	36±181	0.034±0.171±0.007
5.44	0.42	-3.07±4.32	0.916	2.73	-17	-144±175	-0.149±0.181±0.015
5.81	0.41	2.54±4.16	0.917	2.82	-17	79±163	0.089±0.183±0.014
6.19	0.40	7.32±3.96	0.913	2.88	-18	260±153	0.315±0.185±0.039
6.56	0.39	-0.97±3.85	0.912	2.72	-19	-62±158	-0.081±0.206±0.013
6.94	0.38	2.43±3.83	0.913	2.75	-18	75±155	0.105±0.217±0.017
7.31	0.37	-2.30±3.63	0.908	2.68	-16	-117±152	-0.175±0.227±0.019
7.69	0.35	5.69±3.63	0.910	2.75	-18	205±147	0.327±0.235±0.041
8.06	0.34	5.80±3.34	0.900	2.67	-19	223±141	0.379±0.240±0.064
8.44	0.33	4.31±3.39	0.903	2.79	-17	151±137	0.273±0.247±0.030
8.81	0.32	6.65±3.28	0.886	2.69	-17	266±140	0.511±0.268±0.074
9.19	0.31	-4.86±3.21	0.872	2.59	-18	-242±144	-0.490±0.293±0.083
9.56	0.30	3.36±2.99	0.815	2.41	-17	152±154	0.326±0.331±0.076
9.94	0.28	-2.35±2.41	0.803	2.19	-16	-156±139	-0.353±0.315±0.046

Table B.3: Cross section difference $\Delta\sigma$ (nb/GeV-sr) for $^{15}ND_3$ and structure function g_1^d at 4.5°. The values of W^2 and Q^2 (GeV^2) are given at bin centers. $\overline{\Delta R}$ (see Eq. 4.5) represents the experimental average raw count rate difference ($\times 10^{-4}$ counts/gigaelectron). Efficiency ε is absolute. Correction terms $\mathcal{D}$ ($\times 10^{-6}$) and $\mathcal{S}$ are from Eq. 4.7.

W^2	Q^2	$\overline{\Delta R}$	ε	$\mathcal{D}$	$\mathcal{S}$	$\Delta\sigma$	g_1
0.81	1.31	2.32±2.62	0.854	7.77	30	60±39	0.031±0.020±0.006
1.19	1.28	7.99±3.14	0.853	7.41	36	158±49	0.106±0.033±0.020
1.56	1.26	5.12±3.43	0.852	9.38	-59	2±42	0.002±0.035±0.015
1.94	1.23	5.86±3.70	0.846	11.53	2	59±37	0.060±0.038±0.014
2.31	1.20	3.11±3.98	0.843	10.47	23	53±44	0.062±0.053±0.011
2.69	1.18	5.52±4.07	0.839	10.81	1	56±44	0.076±0.060±0.009
3.06	1.15	14.32±4.19	0.831	11.29	-1	144±44	0.223±0.068±0.025
3.44	1.12	3.46±4.19	0.827	11.39	5	39±44	0.068±0.076±0.008
3.81	1.10	5.33±4.16	0.823	11.67	3	55±43	0.105±0.082±0.011
4.19	1.07	8.45±4.14	0.821	11.84	4	84±42	0.179±0.089±0.021
4.56	1.04	6.81±3.99	0.815	11.71	2	69±41	0.160±0.095±0.016
4.94	1.02	9.81±3.98	0.813	11.77	2	100±41	0.251±0.103±0.030
5.31	0.99	0.22±3.82	0.802	11.74	1	-1±40	-0.002±0.109±0.004
5.69	0.96	7.72±3.76	0.796	11.82	1	80±39	0.234±0.115±0.028
6.06	0.93	-0.13±3.64	0.791	11.59	0	-4±39	-0.013±0.123±0.005
6.44	0.91	8.68±3.51	0.793	11.14	-1	94±39	0.317±0.132±0.039
6.81	0.88	4.19±3.45	0.797	11.13	-1	44±38	0.158±0.138±0.025
7.19	0.85	9.32±3.32	0.807	10.54	-1	104±38	0.401±0.147±0.058
7.56	0.83	-0.27±3.19	0.812	10.13	-1	-7±38	-0.029±0.155±0.005
7.94	0.80	-0.17±3.07	0.828	9.50	-2	-6±38	-0.025±0.166±0.005
8.31	0.77	-0.14±2.88	0.853	8.88	-2	-6±37	-0.026±0.171±0.005
8.69	0.75	3.88±2.59	0.847	8.10	-3	51±37	0.247±0.179±0.028
9.06	0.72	-2.43±2.18	0.823	6.62	-3	-48±39	-0.245±0.200±0.033
9.44	0.69	-0.51±1.60	0.761	4.50	-3	-18±46	-0.094±0.246±0.018
9.81	0.67	0.10±0.63	0.507	1.50	-4	7±81	0.039±0.460±0.025

Table B.4: Cross section difference $\Delta\sigma$ (nb/GeV-sr) for $^{15}ND_3$ and structure function g_1^d at 7.0°. The values of W^2 and Q^2 (GeV^2) are given at bin centers. $\overline{\Delta R}$ (see Eq. 4.5) represents the experimental average raw count rate difference ($\times 10^{-4}$ counts/gigaelectron). Efficiency ε is absolute. Correction terms $\mathcal{D}$ ($\times 10^{-6}$) and $\mathcal{S}$ are from Eq. 4.7.

W^2	Q^2	$\overline{A_{\parallel}^{raw}}$	f	$\mathcal{A}$	$A_1 + \eta A_2$	g_1
0.94	0.56	13.01±1.65	0.193	4.5	2.073±0.246±0.255	0.309±0.037
1.31	0.55	7.15±1.49	0.168	-49.2	-0.144±0.163±0.131	-0.019±0.022
1.69	0.54	-0.51±1.46	0.164	-18.7	-0.302±0.121±0.033	-0.069±0.028
2.06	0.53	4.78±1.47	0.158	24.2	0.586±0.099±0.115	0.196±0.033
2.44	0.52	7.99±1.46	0.149	20.0	0.648±0.086±0.087	0.320±0.042
2.81	0.50	5.01±1.48	0.157	-10.5	0.157±0.070±0.045	0.110±0.049
3.19	0.49	5.36±1.51	0.146	-20.6	0.101±0.066±0.058	0.067±0.044
3.56	0.48	2.60±1.54	0.146	-7.9	0.055±0.059±0.017	0.045±0.048
3.94	0.47	0.54±1.60	0.139	-4.0	-0.001±0.057±0.011	-0.001±0.056
4.31	0.46	3.98±1.66	0.147	-1.0	0.118±0.051±0.013	0.125±0.054
4.69	0.45	3.43±1.72	0.147	-1.7	0.088±0.048±0.012	0.103±0.056
5.06	0.43	4.97±1.81	0.147	-1.6	0.121±0.046±0.014	0.154±0.059
5.44	0.42	1.91±1.90	0.142	-2.0	0.039±0.046±0.006	0.054±0.064
5.81	0.41	2.85±1.98	0.148	-1.5	0.057±0.043±0.007	0.084±0.064
6.19	0.40	4.42±2.08	0.143	-1.9	0.086±0.043±0.009	0.138±0.069
6.56	0.39	3.10±2.14	0.148	-1.9	0.053±0.040±0.006	0.091±0.069
6.94	0.38	5.54±2.15	0.138	-1.4	0.100±0.040±0.011	0.185±0.074
7.31	0.37	4.56±2.27	0.139	-1.5	0.076±0.040±0.008	0.149±0.078
7.69	0.35	8.11±2.28	0.148	-1.9	0.122±0.035±0.013	0.254±0.074
8.06	0.34	4.27±2.47	0.141	-1.5	0.063±0.038±0.007	0.139±0.084
8.44	0.33	7.77±2.43	0.139	-1.6	0.112±0.036±0.011	0.263±0.085
8.81	0.32	-0.02±2.51	0.148	-1.9	-0.004±0.033±0.001	-0.010±0.082
9.19	0.31	6.29±2.57	0.152	-1.7	0.074±0.032±0.008	0.193±0.082
9.56	0.30	4.35±2.76	0.163	-2.2	0.044±0.030±0.004	0.120±0.083
9.94	0.28	6.33±3.43	0.165	-2.0	0.062±0.036±0.006	0.178±0.102

Table B.5: Virtual photon-nucleon asymmetry $A_1 + \eta A_2$ for the proton at 4.5°. The values of W^2 and Q^2 (GeV^2) are given at bin centers. $\overline{A_{\parallel}^{raw}}$ (see Eq. 4.4) represents the experimental average raw parallel asymmetry. The dilution factor f and correction term $\mathcal{A}$ are from Eq. 4.6. The value of g_1 in the last column is calculated from the asymmetry and includes only statistical error.

W^2	Q^2	$\overline{A_\parallel^{raw}}$	f	$\mathcal{A}$	$A_1 + \eta A_2$	g_1
0.81	1.31	20.19±2.96	0.199	21.0	1.707±0.206±0.224	0.102±0.012
1.19	1.28	8.39±2.49	0.171	-63.8	-0.179±0.160±0.181	-0.003±0.003
1.56	1.26	5.95±2.29	0.149	-43.5	-0.040±0.140±0.050	-0.004±0.014
1.94	1.23	6.90±2.12	0.147	4.5	0.397±0.111±0.110	0.046±0.013
2.31	1.20	14.01±1.98	0.163	33.9	0.804±0.081±0.112	0.186±0.019
2.69	1.18	14.33±1.94	0.160	2.3	0.541±0.071±0.062	0.165±0.022
3.06	1.15	13.08±1.88	0.154	-9.2	0.394±0.064±0.057	0.145±0.023
3.44	1.12	7.06±1.88	0.150	0.7	0.226±0.059±0.027	0.097±0.026
3.81	1.10	8.67±1.90	0.145	1.6	0.263±0.056±0.029	0.135±0.029
4.19	1.07	13.54±1.91	0.146	4.2	0.381±0.051±0.040	0.224±0.030
4.56	1.04	10.91±1.99	0.139	3.6	0.296±0.052±0.031	0.196±0.034
4.94	1.02	8.40±2.00	0.141	3.3	0.210±0.047±0.022	0.154±0.035
5.31	0.99	13.91±2.09	0.147	3.1	0.305±0.044±0.031	0.246±0.036
5.69	0.96	6.07±2.12	0.143	3.3	0.134±0.043±0.014	0.117±0.038
6.06	0.93	10.48±2.19	0.151	3.0	0.197±0.039±0.020	0.188±0.037
6.44	0.91	8.26±2.27	0.146	2.8	0.153±0.040±0.015	0.157±0.041
6.81	0.88	7.29±2.31	0.144	2.6	0.129±0.039±0.014	0.142±0.043
7.19	0.85	7.85±2.41	0.148	2.7	0.128±0.037±0.013	0.151±0.044
7.56	0.83	11.72±2.50	0.145	2.8	0.181±0.037±0.018	0.229±0.047
7.94	0.80	8.75±2.60	0.135	2.6	0.140±0.040±0.014	0.189±0.054
8.31	0.77	8.25±2.76	0.136	2.3	0.124±0.040±0.012	0.178±0.057
8.69	0.75	10.50±3.08	0.158	2.0	0.129±0.037±0.013	0.197±0.056
9.06	0.72	9.12±3.64	0.135	2.5	0.126±0.048±0.012	0.205±0.079
9.44	0.69	0.68±4.97	0.149	1.6	0.011±0.058±0.003	0.018±0.099
9.81	0.67	15.78±12.58	0.147	2.8	0.183±0.142±0.019	0.334±0.260

Table B.6: Virtual photon-nucleon asymmetry $A_1 + \eta A_2$ for the proton at 7.0°. The values of W^2 and Q^2 (GeV^2) are given at bin centers. $\overline{A_\parallel^{raw}}$ (see Eq. 4.4) represents the experimental average raw parallel asymmetry. The dilution factor f and correction term $\mathcal{A}$ are from Eq. 4.6. The value of g_1 in the last column is calculated from the asymmetry and includes only statistical error.

W^2	Q^2	RC	A_2	θ	E'	Resol	Data	A/T
0.94	0.56	0.0108	0.0259	0.0245	0.0194	0.0187	0.0162	0.0087
1.31	0.55	0.0052	0.0045	0.0008	0.0153	0.0106	0.0061	0.0097
1.69	0.54	0.0006	0.0027	0.0039	0.0035	0.0005	0.0006	0.0004
2.06	0.53	0.0321	0.0060	0.0104	0.0178	0.0022	0.0060	0.0036
2.44	0.52	0.0386	0.0080	0.0161	0.0065	0.0084	0.0127	0.0071
2.81	0.50	0.0135	0.0042	0.0055	0.0148	0.0075	0.0093	0.0072
3.19	0.49	0.0297	0.0038	0.0031	0.0124	0.0014	0.0085	0.0050
3.56	0.48	0.0131	0.0034	0.0021	0.0003	0.0004	0.0045	0.0024
3.94	0.47	0.0093	0.0030	0.0001	0.0018	0.0021	0.0010	0.0006
4.31	0.46	0.0092	0.0029	0.0056	0.0012	0.0055	0.0070	0.0041
4.69	0.45	0.0027	0.0027	0.0048	0.0030	0.0026	0.0063	0.0038
5.06	0.43	0.0051	0.0026	0.0066	0.0001	0.0004	0.0087	0.0051
5.44	0.42	0.0067	0.0025	0.0022	0.0008	0.0001	0.0032	0.0028
5.81	0.41	0.0013	0.0023	0.0034	0.0024	0.0005	0.0047	0.0038
6.19	0.40	0.0002	0.0022	0.0053	0.0024	0.0002	0.0073	0.0099
6.56	0.39	0.0026	0.0021	0.0037	0.0006	0.0006	0.0052	0.0049
6.94	0.38	0.0107	0.0021	0.0075	0.0003	0.0007	0.0099	0.0101
7.31	0.37	0.0053	0.0020	0.0061	0.0019	0.0005	0.0083	0.0062
7.69	0.35	0.0335	0.0020	0.0109	0.0006	0.0159	0.0145	0.0084
8.06	0.34	0.0040	0.0019	0.0054	0.0014	0.0082	0.0074	0.0089
8.44	0.33	0.0043	0.0019	0.0105	0.0016	0.0030	0.0141	0.0086
8.81	0.32	0.0030	0.0018	0.0005	0.0004	0.0005	0.0001	0.0001
9.19	0.31	0.0028	0.0017	0.0090	0.0032	0.0043	0.0122	0.0242
9.56	0.30	0.0112	0.0017	0.0067	0.0093	0.0030	0.0093	0.0303
9.94	0.28	0.0290	0.0016	0.0078	0.0141	0.0029	0.0108	0.0126

Table B.7: Systematic errors (absolute) on g_1^p calculation at 4.5° by category as listed in Section 5.1. RC is shorthand for Radiative Corrections, A/T for Acceptance/Target.

W^2	Q^2	RC	A_2	θ	E'	Resol	Data	A/T
0.81	1.31	0.0081	0.0105	0.0091	0.0107	0.0056	0.0050	0.0049
1.19	1.28	0.0004	0.0005	0.0002	0.0025	0.0004	0.0006	0.0007
1.56	1.26	0.0021	0.0004	0.0004	0.0011	0.0041	0.0019	0.0011
1.94	1.23	0.0015	0.0023	0.0027	0.0138	0.0013	0.0022	0.0012
2.31	1.20	0.0234	0.0088	0.0113	0.0110	0.0063	0.0077	0.0043
2.69	1.18	0.0021	0.0066	0.0093	0.0060	0.0009	0.0088	0.0047
3.06	1.15	0.0069	0.0052	0.0080	0.0098	0.0035	0.0091	0.0055
3.44	1.12	0.0002	0.0031	0.0052	0.0016	0.0009	0.0053	0.0041
3.81	1.10	0.0010	0.0039	0.0069	0.0029	0.0023	0.0072	0.0051
4.19	1.07	0.0062	0.0061	0.0118	0.0060	0.0019	0.0123	0.0142
4.56	1.04	0.0033	0.0046	0.0095	0.0066	0.0086	0.0101	0.0066
4.94	1.02	0.0139	0.0038	0.0077	0.0080	0.0024	0.0082	0.0092
5.31	0.99	0.0017	0.0052	0.0123	0.0105	0.0054	0.0137	0.0156
5.69	0.96	0.0069	0.0039	0.0059	0.0028	0.0007	0.0064	0.0088
6.06	0.93	0.0083	0.0039	0.0099	0.0062	0.0037	0.0111	0.0199
6.44	0.91	0.0059	0.0039	0.0078	0.0048	0.0023	0.0088	0.0106
6.81	0.88	0.0060	0.0038	0.0071	0.0053	0.0043	0.0081	0.0170
7.19	0.85	0.0049	0.0037	0.0077	0.0069	0.0009	0.0088	0.0164
7.56	0.83	0.0087	0.0038	0.0114	0.0009	0.0104	0.0134	0.0245
7.94	0.80	0.0035	0.0037	0.0089	0.0006	0.0017	0.0105	0.0164
8.31	0.77	0.0169	0.0036	0.0080	0.0092	0.0045	0.0095	0.0068
8.69	0.75	0.0051	0.0036	0.0093	0.0118	0.0026	0.0112	0.0068
9.06	0.72	0.0028	0.0035	0.0079	0.0076	0.0029	0.0095	0.0139
9.44	0.69	0.0015	0.0033	0.0005	0.0019	0.0045	0.0004	0.0014
9.81	0.67	0.0405	0.0033	0.0095	0.0498	0.0305	0.0114	0.0753

Table B.8: Systematic errors (absolute) on g_1^p calculation at 7.0° by category as listed in Section 5.1. RC is shorthand for Radiative Corrections, A/T for Acceptance/Target.

W^2	Q^2	RC	A_2	θ	E'	Resol	Data	A/T
0.94	0.56	0.0094	0.0172	0.0163	0.0106	0.0140	0.0118	0.0084
1.31	0.55	0.0343	0.0035	0.0042	0.0210	0.0065	0.0047	0.0066
1.69	0.54	0.0257	0.0036	0.0052	0.0133	0.0019	0.0007	0.0005
2.06	0.53	0.0162	0.0076	0.0133	0.0145	0.0032	0.0135	0.0101
2.44	0.52	0.0090	0.0025	0.0050	0.0061	0.0008	0.0044	0.0032
2.81	0.50	0.0078	0.0016	0.0018	0.0050	0.0013	0.0005	0.0004
3.19	0.49	0.0140	0.0032	0.0085	0.0058	0.0020	0.0143	0.0106
3.56	0.48	0.0053	0.0018	0.0021	0.0004	0.0003	0.0044	0.0031
3.94	0.47	0.0047	0.0015	0.0005	0.0002	0.0003	0.0005	0.0004
4.31	0.46	0.0037	0.0013	0.0019	0.0015	0.0004	0.0019	0.0014
4.69	0.45	0.0047	0.0013	0.0052	0.0014	0.0003	0.0087	0.0065
5.06	0.43	0.0054	0.0013	0.0015	0.0010	0.0014	0.0035	0.0026
5.44	0.42	0.0051	0.0014	0.0066	0.0037	0.0016	0.0088	0.0080
5.81	0.41	0.0082	0.0010	0.0039	0.0042	0.0008	0.0072	0.0063
6.19	0.40	0.0065	0.0025	0.0137	0.0011	0.0051	0.0226	0.0280
6.56	0.39	0.0110	0.0010	0.0035	0.0025	0.0011	0.0038	0.0036
6.94	0.38	0.0103	0.0010	0.0045	0.0014	0.0021	0.0087	0.0087
7.31	0.37	0.0106	0.0011	0.0075	0.0017	0.0002	0.0101	0.0084
7.69	0.35	0.0100	0.0019	0.0140	0.0004	0.0216	0.0239	0.0175
8.06	0.34	0.0393	0.0021	0.0161	0.0062	0.0238	0.0275	0.0309
8.44	0.33	0.0110	0.0014	0.0116	0.0011	0.0031	0.0203	0.0152
8.81	0.32	0.0365	0.0025	0.0216	0.0102	0.0030	0.0364	0.0474
9.19	0.31	0.0519	0.0022	0.0207	0.0125	0.0026	0.0304	0.0517
9.56	0.30	0.0198	0.0014	0.0137	0.0113	0.0116	0.0244	0.0659
9.94	0.28	0.0094	0.0014	0.0148	0.0285	0.0060	0.0211	0.0233

Table B.9: Systematic errors (absolute) on g_1^d calculation at 4.5° by category as listed in Section 5.1. RC is shorthand for Radiative Corrections, A/T for Acceptance/Target.

W^2	Q^2	RC	A_2	θ	E'	Resol	Data	A/T
0.81	1.31	0.0038	0.0029	0.0025	0.0021	0.0019	0.0010	0.0010
1.19	1.28	0.0061	0.0080	0.0077	0.0123	0.0001	0.0055	0.0062
1.56	1.26	0.0124	0.0018	0.0001	0.0050	0.0040	0.0035	0.0025
1.94	1.23	0.0031	0.0032	0.0037	0.0113	0.0001	0.0039	0.0028
2.31	1.20	0.0068	0.0028	0.0037	0.0061	0.0017	0.0024	0.0017
2.69	1.18	0.0032	0.0030	0.0043	0.0018	0.0002	0.0050	0.0036
3.06	1.15	0.0049	0.0079	0.0121	0.0030	0.0063	0.0151	0.0113
3.44	1.12	0.0032	0.0031	0.0036	0.0023	0.0023	0.0039	0.0033
3.81	1.10	0.0036	0.0031	0.0054	0.0002	0.0024	0.0067	0.0054
4.19	1.07	0.0042	0.0047	0.0090	0.0055	0.0003	0.0115	0.0125
4.56	1.04	0.0015	0.0038	0.0079	0.0032	0.0004	0.0104	0.0080
4.94	1.02	0.0070	0.0055	0.0123	0.0092	0.0012	0.0166	0.0177
5.31	0.99	0.0008	0.0034	0.0001	0.0002	0.0008	0.0003	0.0003
5.69	0.96	0.0007	0.0044	0.0111	0.0030	0.0038	0.0155	0.0195
6.06	0.93	0.0026	0.0033	0.0006	0.0000	0.0008	0.0009	0.0014
6.44	0.91	0.0015	0.0052	0.0148	0.0140	0.0073	0.0214	0.0240
6.81	0.88	0.0071	0.0034	0.0073	0.0020	0.0035	0.0108	0.0193
7.19	0.85	0.0133	0.0057	0.0184	0.0021	0.0114	0.0272	0.0438
7.56	0.83	0.0016	0.0034	0.0013	0.0008	0.0019	0.0016	0.0025
7.94	0.80	0.0037	0.0032	0.0011	0.0006	0.0005	0.0012	0.0016
8.31	0.77	0.0030	0.0029	0.0012	0.0021	0.0001	0.0011	0.0009
8.69	0.75	0.0121	0.0030	0.0110	0.0029	0.0042	0.0174	0.0130
9.06	0.72	0.0117	0.0026	0.0108	0.0126	0.0047	0.0154	0.0203
9.44	0.69	0.0076	0.0024	0.0042	0.0000	0.0012	0.0053	0.0151
9.81	0.67	0.0062	0.0023	0.0017	0.0003	0.0005	0.0043	0.0233

Table B.10: Systematic errors (absolute) on g_1^d calculation at 7.0° by category as listed in Section 5.1. RC is shorthand for Radiative Corrections, A/T for Acceptance/Target.

W^2	Q^2	RC	R	θ	E'	Resol	Data	Dilut
0.94	0.56	0.0062	0.1879	0.0728	0.0601	0.0783	0.0873	0.0834
1.31	0.55	0.0083	0.0130	0.0033	0.1156	0.0328	0.0354	0.0338
1.69	0.54	0.0006	0.0273	0.0051	0.0163	0.0052	0.0019	0.0018
2.06	0.53	0.0837	0.0529	0.0077	0.0541	0.0019	0.0145	0.0138
2.44	0.52	0.0548	0.0583	0.0070	0.0114	0.0098	0.0211	0.0201
2.81	0.50	0.0339	0.0141	0.0014	0.0201	0.0040	0.0106	0.0101
3.19	0.49	0.0544	0.0090	0.0008	0.0118	0.0027	0.0106	0.0101
3.56	0.48	0.0146	0.0049	0.0004	0.0003	0.0008	0.0045	0.0043
3.94	0.47	0.0102	0.0001	0.0000	0.0022	0.0020	0.0009	0.0008
4.31	0.46	0.0011	0.0104	0.0006	0.0012	0.0010	0.0055	0.0053
4.69	0.45	0.0062	0.0078	0.0004	0.0008	0.0001	0.0043	0.0041
5.06	0.43	0.0042	0.0105	0.0005	0.0002	0.0000	0.0057	0.0055
5.44	0.42	0.0045	0.0034	0.0002	0.0004	0.0004	0.0021	0.0020
5.81	0.41	0.0032	0.0049	0.0002	0.0006	0.0006	0.0028	0.0026
6.19	0.40	0.0016	0.0073	0.0003	0.0002	0.0002	0.0041	0.0039
6.56	0.39	0.0027	0.0044	0.0002	0.0008	0.0002	0.0026	0.0025
6.94	0.38	0.0042	0.0084	0.0003	0.0003	0.0004	0.0047	0.0045
7.31	0.37	0.0024	0.0063	0.0002	0.0002	0.0001	0.0036	0.0034
7.69	0.35	0.0031	0.0099	0.0003	0.0002	0.0000	0.0057	0.0054
8.06	0.34	0.0021	0.0050	0.0001	0.0000	0.0000	0.0030	0.0028
8.44	0.33	0.0010	0.0088	0.0002	0.0005	0.0000	0.0052	0.0050
8.81	0.32	0.0012	0.0003	0.0000	0.0001	0.0002	0.0000	0.0000
9.19	0.31	0.0027	0.0056	0.0001	0.0002	0.0002	0.0035	0.0033
9.56	0.30	0.0003	0.0033	0.0001	0.0003	0.0000	0.0021	0.0021
9.94	0.28	0.0018	0.0045	0.0001	0.0004	0.0001	0.0030	0.0028

Table B.11: Systematic errors (absolute) on proton $A_1 + \eta A_2$ calculation at 4.5° by category as listed in Section 5.2. RC is shorthand for Radiative Corrections.

W^2	Q^2	RC	R	θ	E'	Resol	Data	Dilut
0.81	1.31	0.0119	0.1538	0.0664	0.1216	0.0218	0.0634	0.0507
1.19	1.28	0.0361	0.0161	0.0055	0.1735	0.0028	0.0242	0.0194
1.56	1.26	0.0186	0.0036	0.0010	0.0126	0.0386	0.0163	0.0131
1.94	1.23	0.0165	0.0356	0.0086	0.1006	0.0010	0.0163	0.0130
2.31	1.20	0.0742	0.0718	0.0150	0.0171	0.0141	0.0258	0.0206
2.69	1.18	0.0103	0.0481	0.0088	0.0196	0.0012	0.0237	0.0190
3.06	1.15	0.0342	0.0350	0.0057	0.0105	0.0043	0.0200	0.0160
3.44	1.12	0.0106	0.0200	0.0029	0.0039	0.0024	0.0100	0.0080
3.81	1.10	0.0075	0.0231	0.0030	0.0021	0.0013	0.0115	0.0092
4.19	1.07	0.0054	0.0332	0.0040	0.0004	0.0004	0.0164	0.0131
4.56	1.04	0.0071	0.0257	0.0028	0.0005	0.0005	0.0127	0.0102
4.94	1.02	0.0009	0.0181	0.0018	0.0004	0.0003	0.0089	0.0072
5.31	0.99	0.0037	0.0261	0.0024	0.0004	0.0014	0.0133	0.0106
5.69	0.96	0.0052	0.0113	0.0010	0.0007	0.0004	0.0056	0.0045
6.06	0.93	0.0022	0.0165	0.0013	0.0001	0.0006	0.0085	0.0068
6.44	0.91	0.0002	0.0127	0.0009	0.0003	0.0005	0.0065	0.0052
6.81	0.88	0.0058	0.0106	0.0007	0.0014	0.0007	0.0055	0.0044
7.19	0.85	0.0036	0.0103	0.0007	0.0008	0.0002	0.0055	0.0044
7.56	0.83	0.0015	0.0145	0.0009	0.0020	0.0012	0.0079	0.0063
7.94	0.80	0.0023	0.0110	0.0006	0.0004	0.0007	0.0060	0.0048
8.31	0.77	0.0008	0.0096	0.0005	0.0014	0.0014	0.0054	0.0043
8.69	0.75	0.0033	0.0098	0.0005	0.0002	0.0010	0.0056	0.0045
9.06	0.72	0.0017	0.0094	0.0004	0.0024	0.0029	0.0055	0.0044
9.44	0.69	0.0006	0.0008	0.0000	0.0014	0.0025	0.0004	0.0003
9.81	0.67	0.0098	0.0130	0.0005	0.0016	0.0012	0.0080	0.0064

Table B.12: Systematic errors (absolute) on proton $A_1 + \eta A_2$ calculation at 7.0° by category as listed in Section 5.2. RC is shorthand for Radiative Corrections.

Appendix C

MLIB: A C++ Library for DAQ Program Interfaces

It was clear in E143 that both the hardware and software components of the data acquisition (DAQ) system in End Station A (ESA) were being pushed to their limits. For the following experiment in ESA, conducted by the E154 collaboration[68], the DAQ team decided to rework the whole system. As part of this redesign, they decided to upgraded the DAQ software to have a modern graphical user interface (GUI) using X-windows. To this end, a simple and easy to use library was needed that could be used by physicist programmers with rudimentary knowledge of the C programming language. As an active member of the E154 collaboration and the DAQ team, I was given the task to create this library. This appendix chapter includes the documentation for the library that resulted called MLIB. The E154 ran successfully in October and November of 1995, and MLIB continues to be used and developed in preparation for the upcoming E155 experiment[69] and other future ESA experiments.

C.1 Introduction

Mlib is a C++ class library intended to make development of Motif applications for data acquisition and control easier. It contains classes for handling common DAQ operations such as logging and displaying dynamic values. Mlib is based on MotifApp by Douglas Young from the book **Object-Oriented Programming with C++ and OSF/Motif** (ISBN 0-13-630252-1).

The reader of this document is assumed to have experience writing standard C programs. C++ and X window programming experience is not necessary for basic usage of Mlib, but is definitely useful and required in writing extensions. The purpose of this document is not to be a complete reference to Mlib. It only provides a general overview and tutorial to writing Mlib programs. Together with the man pages for the classes one should have all the information needed for simple application development. For more advanced use, one will almost surely need to consult the general X window programming references.

In general, programming using Mlib consists of creating class objects, configuring the objects, and writing callbacks for the objects. For those unfamiliar with C++ and X window programming, let us take a moment to clarify these terms.

A class can be thought of as a data type. Therefore, the simplest classes in C++ are the native types: char, int, float, etc. An *object* is simply an instance of a class. Therefore, in the declaration **int x**, x is an int object.

Standard C gives the programmer the ability to build more complicated data types using structures which serve as containers for one or more related instances of other types (which can of course include other structures). This is called *encapsulation*, making it easier for the programmer to manage related data items as a whole.

The *class* construct in C++ extends the structure "concept" in many ways. The most major extension is the concept of class methods, functions which are specific to a class. It is true that one can write functions in standard C that operate on a specific structure type and that pointers to these functions can even be stored as fields in the structure itself. However, the syntax of calling these functions through the pointers is messy. Also, if you have a function that does a similar operation on two different structures such as print, you are restricted by standard C into given each of the functions a different name (i.e. printCoord, printEmployee, etc).

With a class in C++, both of the above problems are taken care of. C++ defines a syntax that lets you access methods just like you do fields in a structure. For instance, if CoordPtr is a pointer to an object of class Coord which has the method Print that takes no arguments, you would call it like this: `CoordPtr->Print()`.

Methods names are specific to a class so C++ could not care less if you have two classes which have methods with the same name just like standard C doesn't care if you have two structures with the fields of the same name. This *overloading* of function names is often called *polymorphism*. In addition, two methods in the same class can have the same name as long as their arguments are different.

C++ classes can also have regular data fields just like structures. However, as you will learn if you read a detailed C++ introduction, "good class design" is to hide these fields from the end user giving access when necessary through class methods. You will most often encounter this in Mlib when you desire the actual X window widget pointer contained in a class. This is almost always obtained through the baseWidget method (i.e. `dialogPtr->baseWidget()`). This design principle relates to one of the big buzzwords of object-oriented programming (OOP) called *data abstraction*.

To avoid confusion in the future, let me also mention another concept of OOP, *inheritance*. This relates to the fact that classes can be derived from other classes. A *subclass* inherits all the data members and methods of its *superclass* and can override them and define new ones to create additional functionality. A common example is a Student class being a subclass of a Person class. In some circles, the term parent and child are used instead of subclass and superclass. I will use the latter terms to prevent confusion with the parent and child relationships of Motif widgets which belong to the Mlib classes. To add to the complexity, some classes may contain instances of other classes as data members. I will refer to this relationship as the *container* and *element* class. The prevalent example of this is the CtrlWindow class which contains instances of the MenuBar, DashBoard, Logger, and ControlPanel as elements.

So finally, what is a *callback*? A critical concept to remember when writing X windows programs is that you are doing event driven programming. Events in X windows include a wide variety of actions such as a mouse clicks, key presses, and timeouts (i.e. alarms). It is the programmer's job to tell the X window event manager what to do with an event, if anything. This is done by creating callbacks which are functions attached to an event that are called when the event occurs. In general, X window programming consists of creating and laying out widgets, writing callback functions and attaching them to the appropriate events.

C.2 Creating an Mlib Application

Again, let me stress that the only functions a Mlib user needs to write are callback functions and other miscellaneous functions that these callbacks may call. The user does not have to (and cannot) create the usual C main() function. Instead, one should consider the setup callback for the application's main window to be the "effective" main() for the program. This brings us to two lines all Mlib programs will have in common.

```
Application *testApp = new Application ( "Test", &parseCmdArgs );
MainWindow *theCtrlWindow = new CtrlWindow( "Test", &setupCtrl );
```

These two lines should appear in the global variable section of your code, normally right after the include statements. These lines create an instance of the Application and CtrlWindow classes. In C++ terms, they are called *constructor* calls. The token "new" is actually a unary operator which allocates memory for the given type. Following the type name is an argument list to use for initializing the class. The argument list structure can be different for each class. In fact, the CtrlWindow constructor takes an optional third argument telling it whether to construct a DashBoard or WhiteBoard object using the `CwDashBoard` or `CwWhiteBoard` enumerated constants. The default is to make a DashBoard object. Both DashBoard and WhiteBoard are subclasses of the UpdateBoard class.

The Application class handles the details of the program which concern the application as a whole and also serves as a central manager for all the main windows in the program (yes, you can have more than one). The first argument to Application's constructor is a name to give the application. This is important for specifying X resources which will be discussed latter. The last argument is a pointer to the callback function to use for parsing command line arguments. You can simply pass the constant `NULL` if you wish to ignore them. The callback must follow the following prototype:

```
void parseCmdArgs ( int argc, char **argv );
```

The arguments argc and argv are the same as for main().

The CtrlWindow class is a subclass of the MainWindow class which contains elements relative to data acquisition and control programs: a MenuBar, a DashBoard or WhiteBoard, a Logger, and a ControlPanel (see Figure C.1). These will be discussed in more detail later. CtrlWindow's constructor has an argument structure like

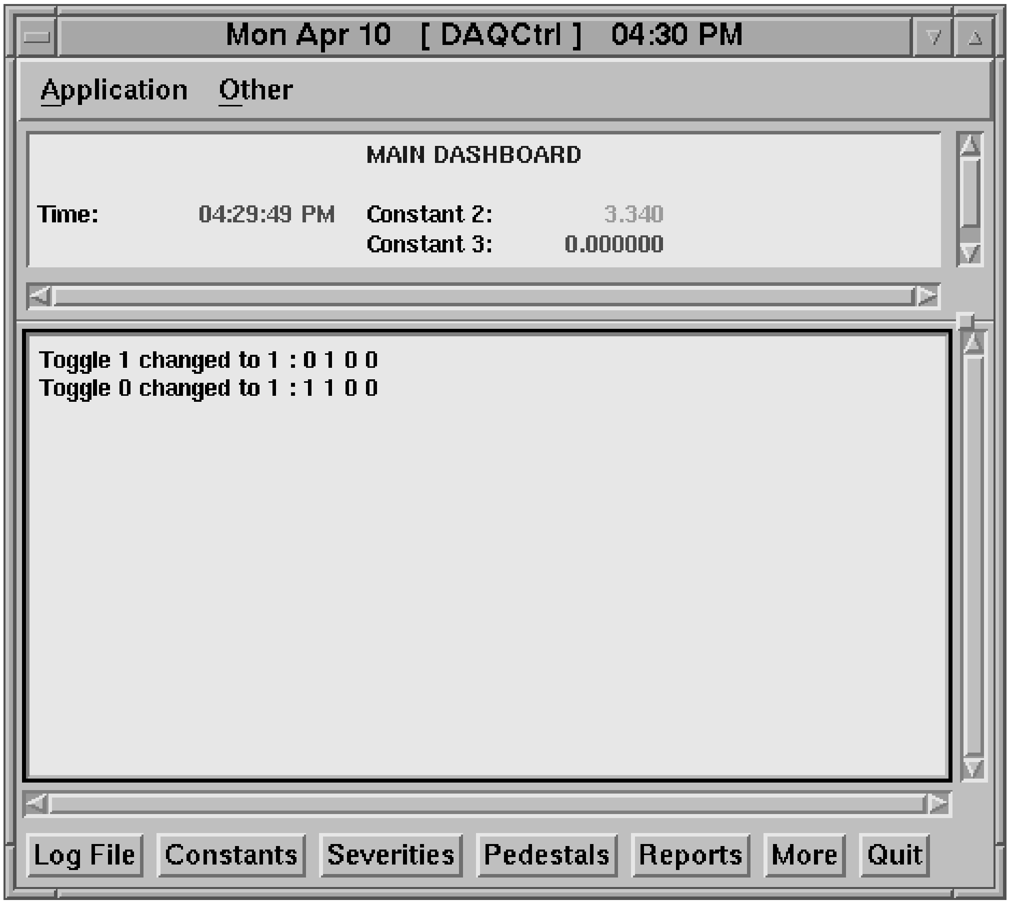

Figure C.1: Example of CtrlWindow object.

that of Application. In this case, the callback is for setting up the the CtrlWindow's elements. It must have the following prototype:

```
void setupCtrl ( CtrlWindow *cw, MenuBar *menuBar,
    UpdateBoard *board, Logger *logger, ControlPanel *control );
```

Actually, since C++ is a strong typed language, these prototypes (or at least the definition of the callback functions themselves) must appear before the constructor calls. The skeleton for a basic Mlib program can be written as in Figure C.2. Note that you must include the declaration file for each class to be used. To be compatible with VMS, the names of all the files are completely in lowercase. If you feel adventurous, browsing these include files would be instructive.

```
// Skeleton of a basic Mlib program

#include "application.hh"
#include "ctrlwindow.hh"
#include "dashboard.hh"
#include "logger.hh"
#include "menubar.hh"
#include "controlpanel.hh"

void parseCmdArgs ( int, char ** );
void setupCtrl ( CtrlWindow *, MenuBar *,
    UpdateBoard *, Logger *, ControlPanel * );

Application *testApp = new Application ( "Test", &parseCmdArgs );
MainWindow *theCtrlWindow = new CtrlWindow( "Test", &setupCtrl );

void parseCmdArgs ( int argc, char **argv )
{
  // Insert details of argument processing here
  // NOTE: only the Application has been created at this point
  // save the options passed and wait until setupControls is
  // called to process them if other objects (like CtrlWindow)
  // are involved
}

void setupCtrl ( CtrlWindow *cw, MenuBar *menuBar,
    UpdateBoard *board, Logger *logger, ControlPanel *control)
{
  // Insert details of setup for CtrlWindow here
}
```

Figure C.2: Skeleton of a basic MLIB program.

C.3 The CtrlWindow Class

The CtrlWindow Class has four predefined elements: a MenuBar, a DashBoard or WhiteBoard, a Logger, and a ControlPanel. The MenuBar and ControlPanel are two different interfaces with the purpose of providing the user a way to execute commands. These commands involve generating calls to the programmer's callbacks or to methods of objects in the program. The MenuBar uses the familiar pulldown menu interface which is common to the majority of X window applications. The ControlPanel uses the nearly as common button bar interface.

It is the UpdateBoard and Logger elements which make CtrlWindow specific to data acquisition and control programming. UpdateBoard's subclasses, DashBoard and WhiteBoard, provide a simple way for the programmer to display data which may change on a periodic basis (such as number of triggers or even tape number). The Logger class serves as a central control for messages to the user. Each message can be assigned a severity class and severity level which are compared with internal settings in the Logger so it can decide whether a message should be written to screen, be written to file, and/or ring the bell.

The main purpose of the CtrlWindow callback (**setupCtrl** in the previous examples) is to setup CtrlWindow's elements appropriately for your program. As a general rule, the first element that should be setup in the callback is the Logger. Its setup is quite simple and you may wish to log messages during the setup of other items. The next item to setup should be the DashBoard, and then the MenuBar and ControlPanel. The latter two are usually done concurrently as will be described later.

A note to C++ purists: Undoubtably, one will be allocating memory for new objects in the CtrlWindow callback. Ideally, these objects should be destroyed in the CtrlWindow destructor and therefore the memory released. But how would one do this? The only method I can see is attach another callback to the CtrlWindow called from its destructor. The programmer would need to save pointer to all the newly created objects in the CtrlWindow setup callback as global variables so this second callback would have access to delete them. Pretty messy. However, the CtrlWindow when used properly exists for the entire scope of the program and is only destroyed when the program is exited. Therefore, a callback for the CtrlWindow's destructor seems unnecessary since all existing objects will be deleted then.

C.4 The Logger Class

The setup of the Logger element usually consists of a single call to the initializeSeverities method of the Logger.

```
logger->initializeSeverities( sevClasses, XtNumber(sevClasses) );
```

The first argument is a list of character strings which are the names of the *severity* classes you which to use. `XtNumber()` is a macro made available through the X windows include files which you get through application.hh. It simply counts the number of elements in a list. This is what is required as the second argument. Back in the global variable portion of your code, you should create a list like the following.

```
char *sevClasses[] = { "DATSERV", "BEAM", "SPECT A", "SPECT B" };
```

Each severity class has three severity thresholds corresponding to the screen writing, file logging, and bell ringing actions. These are initialized to 0, 0, and 2 respectively. This means that when a message is logged to the Logger for a severity class, it will always be written to screen and to file, but the bell will only be rung if the severity level of the message is 2 or higher. These values are manipulated with the setSeverity and getSeverity methods.

```
void setSeverity( char *class, int scrn, int file=0, int bell=2);
int  getSeverity( char *class, int *scrn, int *file, int *bell);
```

There is another setting for the Logger to turn on and off file logging no matter what the severity threshold. By default it is off since it doesn't as yet have a valid log file name. One can control file logging with the setLogFile and fileLogOn methods as in with the following code.

```
if ( logger->setLogFile( "file.log" ) )
  logger->fileLogOn();
```

It is important for users to note that the log file is not physically opened until file logging is turned on. Also, when logging is turned off, the file is closed. The default mode is to overwrite the present log file on the next call to fileLogOn. An optional second argument to setLogFile allows one to change the mode of the file open so it does an append instead. The possible values for this argument is a bitwise-OR of any of the following constants from <fstream.h>

`ios::out`	Open for output.
`ios::ate`	Set the initial position to the end of the file.
`ios::app`	Seek to end of file before each write.
`ios::trunc`	Guarantee a fresh file; discard any previous contents.
`ios::nocreate`	Guarantee an existing file; fail if file did not already exist.
`ios::noreplace`	Guarantee a new file; fail if file already existed.

A common example would be

```
logger->setLogFile( "file.log", ios::out|ios::app );
```

An essential step at this point is also to save a pointer to the Logger object in a global variable so that your other callbacks can access the Logger for logging messages. The standard name to use is `theLogger` which should be declared in the global section of the code as follows

```
Logger *theLogger;
```

and then initialized inside `setupCtrl()` with

```
theLogger = logger;
```

In your callbacks, you can then log messages using the **log** method as in the following example

```
theLogger->log( "ERROR: Invalid type", 1, "BEAM" );
```

where the first argument is the message string, the second is the severity level and the third is the severity class. The last two arguments are optional and will default to `0` and `"Default"` respectively. Severity class matching is case insensitive.

C.5 The UpdateBoard Classes

C.5.1 The DashBoard Class

The second element you should setup in the CtrlWindow callback is the UpdateBoard. The default UpdateBoard is the DashBoard, which is a text drawing area divided into rows and columns. The row height varies with the font used, but the column width can be set to an arbitrary number of characters by the programmer using the setColumnWidth method. Both row and column start at zero. On this grid of row-column cells, the programmer places items of various types which are identified with the following enumerated constants:

```
enum DashBoardItemType {
  DbInternalString, DbInternalInt,
  DbInternalFloat,  DbInternalDouble,
  DbInternalShort,  DbInternalChar,
  DbExternalString, DbExternalInt,
  DbExternalFloat,  DbExternalDouble,
  DbExternalShort,  DbExternalChar
};
```

These allows one to display character characters, strings, shorts, integers, floats, and doubles. The external items have pointers to areas in memory where the dashboard can obtain the item's value whenever it needs it. The internal items create internal storage space where the programmer must write the desired data each update period of the dashboard. This is done in (what else) a callback that the user must register with the DashBoard during setup and that will get called at each update interval.

Items are added to the DashBoard using the addItem method which has the following prototype:

```
int addItem ( DashBoardItemType type, ... );
```

The $(\cdots)$ mean this method takes a variable number of arguments. In this case, the additional arguments consist of alternating DashBoard item options and values for the options finally terminated with a `DbNULL`. The options are identified with the following enumerated constants:

```
enum DashBoardItemOptions {
  DbNULL, DbLabel, DbLabelForeground, DbLabelBackground,
  DbRow, DbColumn, DbValue, DbValueForeground,
  DbValueBackground, DbValueSize, DbFormat,
  DbExternalLabel, DbExternalFormat
};
```

Tables C.1 and C.2 summarizes the valid values for each of these options. If an item is not given a label, or the label is later set to `NULL`, then the item's value is left

option	*value type*	*description*
DbColumn	int	column to place item in
DbExternalLabel	char *	string to use for item's label
DbExternalFormat	char *	printf()–style format string
DbFormat	char *	printf()–style format string
DbLabel	char *	string to use for items label
DbLabelBackground	UpdateBoardColor	background color for label
DbLabelForeground	UpdateBoardColor	foreground color for label
DbRow	int	row to place item in
DbValue	*varies*	see Table C.2
DbValueBackground	UpdateBoardColor	background color for label
DbValueForeground	UpdateBoardColor	foreground color for label
DbValueSize	int	bytes needed for DbInternalString

Table C.1: Summary of DashBoard item options.

aligned in the row-column cell. Otherwise, the label is left aligned and the value is right aligned. The default available colors of UpdateBoardColor type include Black, White, Red, Green, Blue, and Yellow. These can be extended using the addColor method (see the UpdateBoard man page).

The DbInternalString is the most complex item since it requires that you tell it how much space it needs to allocate internally for storing strings. Any string passed with the DbValue option will be copied to this storage area so the programmer is responsible for making sure it fits. The DbValueSize option is ignored for all other item types. The difference between DbLabel and DbExternalLabel is that the former makes an internal copy of the string passed while the latter only stores the pointer. The same is true between DbFormat and DbExternalFormat.

The integer returned from the addItem method is the index of that item in the DashBoard. This must be used in order to change any of the options for the item later. This is especially critical for internal item which must be set by the user. Since the indices returned are in numerical order of creation starting at zero, it is not strictly necessary for the programmer to store these values so in general they can be ignored.

Items can be easily removed with the removeItem method which takes the index of the item to remove as its sole argument. The clearItems method will remove all

item type	*value type*	*description*
DbInternalString	char *	initial string value to store
DbInternalChar	char	initial character value to store
DbInternalShort	short	initial short value to store
DbInternalInt	int	initial integer value to store
DbInternalFloat	float	initial float value to store
DbInternalDouble	double	initial double value to store
DbExternalString	char *	character string in global memory
DbExternalChar	char *	pointer to a char in global memory
DbExternalShort	short *	pointer to a short in global memory
DbExternalInt	int *	pointer to an integer in global memory
DbExternalFloat	float *	pointer to a float in global memory
DbExternalDouble	double *	pointer to a double in global memory

Table C.2: Summary of value types for `DbValue`.

items. Neither of these methods actually frees memory, but just marks it as usable for newly added items. The freeItems will actually free all allocated memory for DashBoard items. Also, if clearItems or freeItems are used, indices for newly added items will restart at zero.

```
void removeItem ( int index );
void clearItems();
void freeItems();
```

One potential problem is that DashBoard has been designed such that the dashboard item data must be maintained in a contiguous block of memory. If a new item being added will not fit in the chunk of memory currently allocated, the dashboard will allocate a new larger space, copy over the contents from the older memory, free the older memory and then add the new items data into the newer memory chunk. This becomes a significant overhead problem for dashboards with large number of items. You can prevent this by using the setAllocationStep method which takes on integer argument representing the number of additional items it should allocate memory for each time it needs to expand the item data memory. So if you know that your dashboard is going to have 5000 items, before using the addItem method, you should use `dashboard->setAllocationStep(5002)` so that the necessary memory is allocated in one step with a little extra to be safe. If you will be adding more items later on at a smaller pace, you should reset the allocation step down to 20 or so.

Another useful method that programmers will often use for setting up the dashboard is the changePeriod method. This method takes a single integer argument representing the number of milliseconds between updates. The start and stop methods are used to start and stop the timeouts ticking. By default, the dashboard is in the stopped mode.

The program user can double click on any item in the dashboard to popup a small dialog which allows the user to change the period value and stop and start the timeouts. In fact, it is this dialog which contains all the timing functions the prompt the call to the DashBoard's callback. The default dialog for a DashBoard is an instance of the UpdateBoardDialog class. Using the setDialog method, any descent class of the UpdateDialog can be attached to the DashBoard. However only instances of the UpdateBoardDialog class know how to call the DashBoard's callback directly.

It is common for the programmer to want all the application's UpdateBoards to use the same timeout. This can easily done by using the getDialog method to get a pointer to the UpdateBoardDialog attached to the main DashBoard in the CtrlWindow and then using the setDialog method on the other UpdateBoards. These two methods have the following prototypes.

```
UpdateDialog *getDialog (); }
void setDialog (UpdateDialog *dlgPtr);
```

Figure C.3 shows an example setup for the CtrlWindow's DashBoard element. It is assumed that the function **mainDashShow()** which is registered as the DashBoard's callback has been previously defined (or at least prototyped). The callback will be described in detail in the next section.

C.5.2 The DashBoard Update Callback

The callback for updates is registered with its DashBoard using the setClient method which takes two arguments. The first is a pointer to the callback function itself. The second is a user data item of type `void *`. This means any 4 byte (on most machines) can be used and will be passed as the second argument to the callback function which has the prototype

```
void setupCtrl ( CtrlWindow *cw, MenuBar *menuBar,
    UpdateBoard *board, Logger *logger, ControlPanel *control)
{
  DashBoard *dashboard = (DashBoard *) board;

    ⋮

// Without a label, the item value will be left aligned
// which is a nice way to do a title
dashboard->addItem( DbInternalString,
    DbRow, 0, DbColumn, 1, DbValueSize, 40,
    DbValue, "MAIN DASHBOARD", DbValueForeground, Blue, DbNULL );

dashboard->addItem( DbInternalString,
    DbLabel, "Time:", DbRow, 2, DbColumn, 0,
    DbValueSize, 20, DbValueForeground, Red, DbNULL);
dashboard->addItem( DbExternalFloat,
    DbLabel, "Constant 2:", DbValue, &(constants[1]),
    DbValueForeground, Green, DbRow, 2, DbColumn, 1,
    DbFormat, "%6.3f", DbNULL );
dashboard->addItem( DbExternalFloat,
    DbLabel, "Constant 3:", DbValue, &(constants[2]),
    DbValueForeground, Red, DbRow, 3, DbColumn, 1, DbNULL );

dashboard->setColumnWidth(20);            // set width of each column
dashboard->setClient(&mainDashShow,NULL); // set callback function
dashboard->changePeriod(10000);           // set period to 10 seconds
dashboard->start();                       // starts dashboard update

    ⋮

}
```

Figure C.3: Example code for setup of DashBoard element in CtrlWindow's callback.

```
Boolean mainDashShow( DashBoard *obj, void *cd,
     int size, DashItem *items);
```

The first argument is a pointer to the DashBoard object itself. The third argument is the total number of items in the DashBoard. The last argument will not be discussed here as it is for advanced use only for gaining more speed in resetting item values (see the DashBoard man page).

I suggest that the first few lines of code in your callback be a check that size is what you think it is. If not, write an error message to the Logger or stderr and return the Boolean constant `FALSE`.

The main purpose of the callback is to set the current values of any internal items the DashBoard might contain. However, it can be used to change the value of any option for any item. This can be useful for setting the foreground of a value to `Red` if it is out of range, for example. It is not a good idea to try and take advantage of the timeout implied with the callback to do other periodic operations not related to the dashboard since the user can always stop, start and change the update period. Also, the callback is only called if the dashboard is managed on the screen (i.e. not iconified).

The method to use for reseting item option values is setItem which has the following prototype:

```
void setItem ( int index, ... );
```

which is similar to the addItem method except now the first argument is the index of the DashBoard item whose options are begin reconfigured. To avoid the overhead of variable argument processing, one can also use the following methods for setting some of the most commonly changed options.

```
void setExternalChar(  int n, char *ptr );
void setExternalShort( int n, short *ptr );
void setExternalInt( int n, int *ptr );
void setExternalFloat( int n, float *ptr );
void setExternalDouble( int n, double *ptr );

void setLabelColor( int n,
                UpdateBoardColor fore, UpdateBoardColor back );
void setValueColor( int n,
                UpdateBoardColor fore, UpdateBoardColor back );
```

The significance of the return value of the callback is to signal the dashboard whether it needs to redraw itself or not. X window draw operations are pretty time intensive, so if you know that none of your items have changed in value, simply return the Boolean constant `FALSE`. Otherwise, return the Boolean constant `TRUE`. If you want the dashboard to redraw only specific items, you can use the drawItem or drawItems methods. To clear the text of a previously drawn item, you can use the clearCell method. The prototypes are:

```
void  drawItem ( int  index );
void drawItems ( int *indices, int numIndices );
void clearCell ( int row, int column);
```

The sole argument to drawItem is the index of the item that needs to be redrawn. The first argument to drawItems is a pointer to a list of indices; the second argument is the number of indices in the list. After using these methods, you should return `FALSE` to prevent the dashboard from redrawing again.

C.5.3 The WhiteBoard Class

WhiteBoard is simply a drawing area for drawing text at arbitrary line and character positions. It can also do simple graphics through methods inherited from the UpdateBoard class. The programmer is completely responsible for doing the drawing on the widget through a registered callback that will be called whenever the text needs to be redrawn. It can also be arranged for this callback to be called at periodic intervals and by user generated events.

Line and character coordinates start at zero. Most methods support specifying the coordinates as separate integer arguments or by the `CharIndex` structure defined as follows.

```
typedef struct {
  int c, l;
  UpdateBoardAnchor anchor;
} CharIndex;
```

It is best to use a fixed-space font for writing on the WhiteBoard as char coordinates are converted to pixel coordinates by multiplying by the font character width which is only an average for proportional fonts. When doing graphics (lines, polygons, etc), the anchor is used to specify the exact pixel location in the character cell. The possible values for `UpdateBoardAnchor` are defined in the following enumeration.

```
enum UpdateBoardAnchor {
  UbCenter, UbNorth, UbNorthEast, UbEast, UbSouthEast, UbSouth,
  UbSouthWest, UbWest, UbNorthWest
};
```

The programmer is responsible for making sure the WhiteBoard is the correct size to fit the desired text written to it. This can be done using the setDimensions method defined as follows

```
void setDimensions( int chars, int lines);
```

which set the width in characters and height in lines of the drawing area. Also, the setWidth and setHeight methods, which take one integer argument, can be used to set each dimension individually.

The drawing callback is set as in the DashBoard class with the setClient method which is prototyped as follows

```
void setClient ( WhiteBoardCallback cb, void *clientData );
```

where the callback now has the following structure.

```
typedef void ( *WhiteBoardCallback )
   ( WhiteBoard *board, void *clientData,
                WhiteBoardEventData *eventData);
```

The first argument is a pointer to the WhiteBoard object itself. The second argument will be the clientData as specified in the setClient method. The third argument will be a pointer to a `WhiteBoardEventData` structure contain the event information. This structure is defined as follows

```
enum UpdateEventType {
    UbTimeout, UbExpose, UbChangeFont, UbChangeUser, UbResize
};

typedef struct {
  UpdateEventType  type;
  CharIndex        first;
  CharIndex        last;
} WhiteBoardEventData;
```

which specifies the type of event plus the character coordinates of the the top, left character and bottom, right character describing the rectangular region that needs to be redrawn. For all events but the **UbExpose** event, this will describe the full view port of the scrolled window. In comparison, the DashBoard callback is only called on **UbTimeout** and **UbChangeUser** events and the others are handled internally.

To obtain the currently viewed line/char coordinates of the WhiteBoard outside of the drawing callback, one can call the getViewport method. The getViewPixels method can be used to obtain the pixel coordinates of the viewport. The setViewAnchor method can be used to set the top, left pixel of the current viewport.

```
void getViewport( int *leftChar, int *topLine,
    int *rightChar, int *bottomLine );
void getViewport( CharIndex *first, CharIndex *last );
void getViewPixels( int *x1, int *y1, int *x2, int *y2 );
void setViewAnchor( int x, int y );
```

There are two variants of the draw method are available for drawing text on the WhiteBoard defined as follows

```
void draw ( int char, int line, char *string );
void draw ( CharIndex *coord, char *string );
```

Both draw the given string at the specified coordinates in the current color. The current color can be changed with the setColor method which takes two arguments of type `UpdateBoardColor`. Possible values include `Black`, `White`, `Red`, `Green`, `Blue`, `Yellow`, and `Transparent`, but these can be extended using the **addColor** method (see the UpdateBoard man page). The first argument specifies the foreground color and the second argument specifies the background color.

Text can be cleared using one of the following methods.

```
void clearLines ( int line, int numLines=1 );
void clearChars ( int char, int line, int numChars);
void clearChars ( CharIndex *coord, int numChars);
void clear();
```

The clearLines method will clear the number of lines given as its second argument starting from the line given as the first argument on down. The clearChars methods will clear the specified number of characters starting at the given position. The clear method clear the whole board.

Refer to the UpdateBoard and WhiteBoard man pages for information on the many graphics drawing methods available. You will probably also want to refer to the Xlib documentation as these methods are mostly wrappers around the Xlib primitive functions.

For manipulating the frequency of update events, the WhiteBoard also has the changePeriod, start and stop methods which work the same as for the DashBoard. It also has an attached UpdateDialog with the same methods for manipulation as the DashBoard.

C.6 The Cmd Class and Cmd Interfaces

The Cmd (short of command) class in the Mlib library is used as a kind of "handle" to connect an action to one or more graphical interface components. For instance, an action that you might wish to be performed is ejecting an exabyte tape. You would create an instance of an ExecCmd class (a subclass of the Cmd class) and set its client callback to your function which will eject the tape. Then you would create some interface component such as a push button which will invoke the Cmd.

So that you can avoid getting into the gory details of creating the graphical interfaces at the X window programming level, a CmdInterface class exists with many subclasses for the various types of widgets you may desire (buttons, menus, etc).

The Cmd and CmdInterface classes are examples of what is called *abstract classes* in C++ terminology. This means that they serve only as superclasses from which classes which the programmer will really use are derived. In fact, one is forbidden from creating instances of the Cmd and CmdInterface classes as they are "missing" all the code necessary to do the job.

Another abstract class in this "family" is the OptionCmd class which is itself a subclass of the Cmd class. This class is used for actions which have two or more choices associated with them. The user's choice would be passed to a callback you register as an additional argument. Similarly, the abstract class OptionCmdInterface exists for commands subclassed from OptionCmd (toggles, option menus, etc). This establishes two types of commands: scalar commands which are not derived from the OptionCmd class and option commands which are. Confused? Please bear with the following discussion and examples and hopefully all will become clear.

C.6.1 Scalar Commands

The most versatile of the scalar commands is the ExecCmd class. This allows the programmer to connect any arbitrary action to a CmdInterface through a callback function. In fact, several callback functions may be registered with the command. A typical example of an ExecCmd follows.

```
ExecCmd *execmd = new ExecCmd ( "Do It", TRUE );
execmd->registerClient ( &execTest, (void *) 1 );
```

The first argument to the ExecCmd constructor is a character string for its name. This will be used by any CmdInterface connected to the command for its X resource name and possibly as the label presented to the program's user on the interface widget (if not overridden in the CmdInterface constructor). The second argument is a Boolean stating whether the command should be initially active (i.e invokable). When a command is inactive, all interfaces connected to it will be displayed in insensitive mode. Buttons for instance will have their labels "grayed". Most always you will wish to pass `TRUE` here. All Cmd classes have these same first two arguments and define the activate and deactivate method, which take no arguments, for changing there sensitivity state.

An ExecCmd can be configured to call any number of callback functions. Each callback is registered with the ExecCmd using the registerClient method which takes two arguments. The first is a pointer to the callback function you wish to be called and the second is an arbitrary client data item which must be typecast as a pointer to void. The execTest() function in the example above is a callback with the following prototype:

```
void execTest( void *clientData );
```

whose argument will be the client data item passed as the second argument in the registerClient method.

All Mlib application should contain one instance of the QuitCmd class for letting the user exit the application. This command will popup a simple dialog asking the user to verify that they really wish to exit the program. If the user answers yes, then the program will exit. An example creation of the QuitCmd would be

```
Cmd *quit = new QuitCmd ( "Quit", TRUE );
```

where the first two arguments are the same as for the ExecCmd.

A third scalar command that will be used quite often is the PostCmd class. Its constructor defines a third argument which is a pointer to any subclass of UIComponent (the grand daddy superclass of all classes containing X window widgets). However, it is really only useful for the classes which define toplevel windows. When invoked, it will post (map to the screen) the toplevel window. If it is already posted, the command will dismiss (unpost) it. An example call to the PostCmd constructor would be

```
Cmd *sevcmd = new PostCmd ( "Severities", TRUE, sevdlg );
```

where `sevdlg` is a pointer to a class which contains a toplevel window, presumably a dialog where the user can set the Logger's severity attributes.

You may have noticed that the pointers to the new object of the previous scalar commands have all been of type Cmd instead of the particular subclass of Cmd created. In C++, it is always valid to assign an instance of a class to a pointer of one of it superclasses. One cannot however go the other way. Why this is done for the examples above will become evident once the CmdList class is introduced below.

Other scalar commands you may wish to investigate through their man pages are the IconifyCmd and SelectFileCmd.

The only interface which connects with the scalar commands is the ButtonInterface class. Usually the Mlib programmer does not need to create instances of this class directly because the MenuBar and ControlPanel container classes take care of it as will be discussed below. An example call to the ButtonInterface constructor would be

```
ButtonInterface *bi =
     new ButtonInterface ( gdlg->gridWidget(), execmd );
```

The first argument is the widget which will be the parent of the button created. This is usually some Manager widget. In this example it is the widget from the GridManager object which is contained in the GridDialog object to which gdlg points. More information will be given on these two classes later. The second argument is a pointer to the scalar command object to which the interface should connect. There is a third optional argument not shown in this example which is a character string to use as the label on the button. If not given, the name of the Cmd object is used as was stated above.

C.6.2 Option Commands

The option command equivalent of the ExecCmd is the OptExecCmd. An example use would be

```
OptExecCmd *optscmd = new OptExecCmd ( "Constants", TRUE,
      constnames, XtNumber( constnames ), 0 );
optscmd->registerClient( &printConstant, (void *) optscmd );
```

The third argument to the constructor is an array of character strings which will be the labels on the choices presented to the user. The fourth argument is the number of strings in the array and the fifth argument is the initial item which should appear selected. This is necessary since most option selection type interfaces, such as listboxes and option menus, must have a currently selected item at all times.

As with the ExecCmd, the first argument is a pointer to the callback function and the second an arbitrary client data item. In the above example, the pointer to the command itself is passed for the client data. The following example of a callback demonstrates how this can be useful as well as showing the prototype necessary for an OptExecCmd callback.

```
void printConstant ( void *clientData, int index ) {
  OptExecCmd *optscmd = (OptExecCmd *) clientData;
  printf( "%s\n", optscmd->getLabel( index ) );
}
```

The method getLabel returns the character string of the label at the specified index.

A specialized subclass of OptExecCmd is the OnOffCmd class in which the labels default to "Off" and "On". To be consistent with the C convention that 0 is false and 1 is true, the index of "Off" is 0 and the index of "On" is 1. An example call to the OnOffCmd constructor would be

```
OptionCmd *oocmd = new OnOffCmd ( "Logging to file:", TRUE, 0 )
```

The third argument is the initial value for the command which for this example is off. In all other respects OnOffCmd operates just like OptExecCmd as described above.

An option command which is mainly useful only for building menus is the OptMultiCmd which serves as a container for other commands to choose from. This class will be described later in the CmdList discussion. Another option command which programmers may find useful is the ChangeFontCmd which allows the user to change the font of any subclass of the TextClient class. At this time, this includes only the Logger, DashBoard, and WhiteBoard classes. See the ChangeFontCmd man page for more details

Option commands have a variety of command interfaces available. For the OnOffCmd, the best choice of interface is the ToggleInterface class which presents a toggle button for the user to switch between "On" and "Off" states. For the OptExecCmd, the most common choices are the OptionMenuInterface and ListboxInterface. The former is desirable when you which to conserve screen real estate while the latter makes large lists of options more tractable. The arguments for the constructors of these three classes are the same,

```
ToggleInterface( Widget parent, OptionCmd *cmd, char *newname );
OptionMenuInterface( Widget parent, OptionCmd *cmd, char *newname );
ListboxInterface( Widget parent, OptionCmd *cmd, char *newname );
```

where `parent` is the widget of the window in which to place the interface component, `cmd` is a pointer to the OptionCmd to connect to, and `newname`, which is optional, is a character string containing a new name for the interface if you wish it to be different than that for `cmd`. This is most useful for the ToggleInterface and OptionMenuInterface objects as this will become the value of their label.

C.6.3 The CmdList and OptMultiCmd Classes

The CmdList class is a simple container class for grouping several commands together. It is used by the MenuBar, ControlPanel and OptMultiCmd classes all of which expect pointers to CmdList objects as arguments. First one must create an empty CmdList which is easily done in the following example constructor call.

```
CmdList *cmdList = new CmdList();
```

and then commands can be added to the CmdList object using the **add** method. It takes one argument, a pointer to Cmd class object.

```
cmdList->add ( togcmd );
cmdList->add ( execcmd );
cmdList->add ( optscmd );
```

One can obtain the current size of the list using the size method which takes on arguments. Accessing the individual Cmd objects is as easy as using the normal [] array notation. For example,

```
    for ( i = 0; i < list->size(); i++)
      ((*list)[i])->deactivate();
```

As stated previously, the OptMultiCmd is an option command which contains and array of other commands as its list of options. This is most useful for creating pull-right menus in the menubar. An example call to its constructor would be

```
OptMultiCmd *multicmd = new OptMultiCmd ( "Others", TRUE, cmdList );
```

where the third argument is a pointer to a CmdList object containing the commands to use for the OptMultiCmd's options.

C.7 The MenuBar and ControlPanel Classes

Once you have created all the commands you wish to have available through the programs main window, the easiest way to make the commands accessible by the user is in the menubar or control panel. The ControlPanel defines the method **addCommands** which has the following prototype.

```
void addCommands( CmdList *cmdList );
```

It creates a push buttons for each of the Cmds in the CmdList passed no matter what their actual subclass. It lays them out from left to right on the screen, but the buttons will be wrapped onto additional lines if needed.

If the Cmd object the button is connected to is a descendent of the OptionCmd subclass, pressing the button will post a popup menu from which the user can click on the desired option. If the Cmd is an OptMultiCmd object, any OptionCmd objects that it contains will become pull-right menus from this popup menu.

Additional calls to the ControlPanel **addCommands** method will just append the commands to those that already exist on the panel. This can be useful if you have some conditionals controlling whether certain commands should be added to the panel.

The MenuBar class also defines an **addCommands** method which has two arguments

```
void addCommands( CmdList *cmdList, char *menuName );
```

The second argument is used as the name of the menu which will present the commands in the CmdList object for selection. Each call to the addCommands method will produce another menu on the menubar, filling from left to right.

Commands of the class OnOffCmd will be presented in the menu as toggles. Those of the OptMultiCmd will produce pull right menus and its commands will be processed just like another menu object. Since it is valid to have other OptMultiCmd objects as options in another, you can have as many levels of pull right menus as you desire. However, proper GUI programming style puts a limit at three levels.

Other OptionCmd class objects will be rendered as pull right menus which are radio button groups. This means that the label of each option for the command will be listed in the menu and a small indicator will be drawn next to the label of the item currently selected.

If it isn't obvious to you by now. let me stress that a single command object can have any number of interfaces connected to it. There is no need to create separate but identical commands so you can have the same action invokable in both the menubar and control panel.

This completes the discussion of those classes that are part of the CtrlWindow. However there are several other useful classes in Mlib that can be created in the CtrlWindow callback. This brings us to a discussion of dialogs and other toplevel windows.

C.8 The UpdateBoardWindow Classes

C.8.1 The DashBoardWindow Class

The DashBoardWindow consists of a toplevel window containing a DashBoard object, an Update push button, a Dismiss push button (see Figure C.4. Also, a method exists to easily add an option menu to the window. The Update button will force an update of the DashBoard object and the Dismiss button will pop down the window. The use of the option menu is explained below. The DashBoardWindow constructor has the following prototype.

```
DashBoardWindow ( char *name,
        DashBoardWindowCallback cb, Display *dpy =NULL );
```

This creates a DashBoardWindow object on display `dpy` using `name` for X resource specification. If the display is `NULL` or not given, then the application's display is used. The callback `cb` will be called when the DashBoardWindow is later initialized (i.e. one the DashBoard is created) and should be used for setting up the DashBoard. It has the following type definition.

```
typedef void ( *DashBoardWindowCallback )
    ( DashBoardWindow *dw, DashBoard *dashboard, Widget form);
```

The first argument is a pointer to the DashBoardWindow object itself. The second argument is a pointer to its DashBoard object. The third argument is the Widget of the XmForm used for laying out the DashBoard and buttons and can be used by the experienced X programmer to add additional controls. Think of this callback as the equivalent of the CtrlWindow's setup callback. See figure C.5 for an example callback.

The option menu that can be attached to the DashBoardWindow with the setOptions method can be used to make the DashBoard object effectively operate as several DashBoards at once. In the setClient method for the DashBoard object, a pointer to the DashBoardWindow can be passed as client data to the DashBoard update callback. In the update callback, the DashBoardWindow can then be queried using the current method which takes no arguments and returns the index of the option menu item currently selected. Using the index, the DashBoard items can be drawn accordingly.

The option menu is created with a call to the setOptions method which has the following prototype.

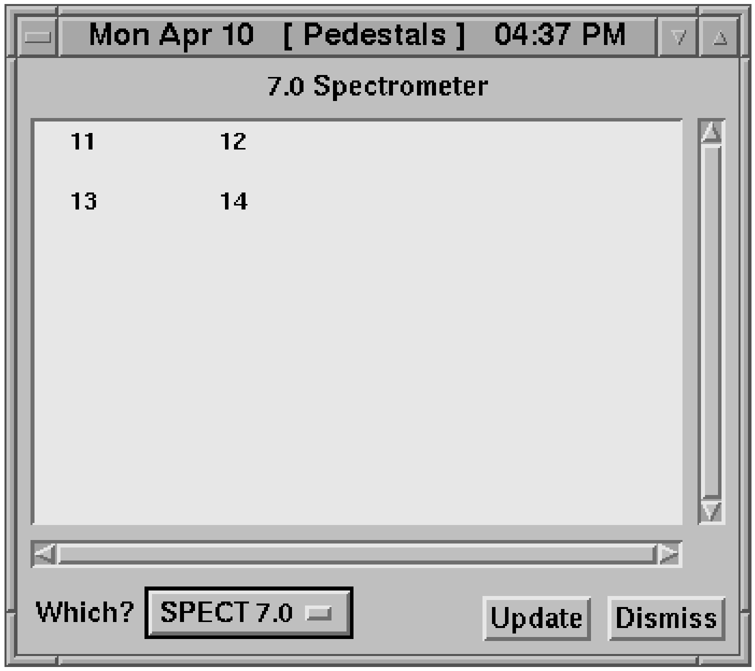

Figure C.4: Example of DashBoardWindow object.

```
void setOptions( char **list,
                int numList, int initVal, char *label =NULL );
```

The first argument is a list of character strings to use for building the menu. The second argument is the number of elements in the list. The third argument is the index (starting at zero) of the initial element to be selected in the menu. The fourth argument is an optional label for the option menu (the default is "Which?")..

A menubar and buttons can be added to the window using a CmdList in much the same way as with the CtrlWindow. This is done with the addMenu, addButton, and addButtons methods.

```
void addMenu( CmdList *list, char *menuTitle );
void addButton( Cmd *cmd );
void addButtons( CmdList *list );
```

Some other useful methods for the DashBoardWindow are list below.

```
void setTitle( char *titleString );
```

```
// list of strings for the DashBoardWindow option menu
char *pedestalLabels[] = {  "SPECT 4.5", "SPECT 7.0", "BEAM" };

// place to store pedestal data
int pedestalData[3][4] = {
  { 1, 2, 3, 4}, {11, 12, 13, 14}, {21, 22, 23, 24}
};

void setupPedestals(DashBoardWindow *dw, DashBoard *board, Widget )
{
  int i, initchoice = 1;

  // set up dashboard items (Note that the fixed width format is
  // need because w/o a label, an item's value is left aligned)
  for (i = 0; i < 4; ++i) {
    board->addItem( DbExternalInt, DbRow, i&2, DbColumn, i&1,
        DbValue, &(pedestalData[initchoice][i]),
        DbFormat, "%6d", DbNULL );
  }

  board->setClient(&pedestalShow, dw);
  board->setColumnWidth(8);
  board->changeFont( DAQFonts[0] );
  board->changePeriod( 10000 );
  board->start();

  // set up options menu
  dw->setOptions( pedestalLabels,
                XtNumber( pedestalLabels ), initchoice );
}
```

Figure C.5: Example of a DashBoardWindow setup callback.

```
DashBoard *getBoard();
void dismiss();
void iconify();
void manage();
```

The setTitle method sets the toplevel's title to given string and the getBoard method returns a pointer to the window's DashBoard object. The manage, iconify and dismiss methods can be used by the programmer to explicitly pop up, iconify and pop down the DashBoardWindow. These methods take no arguments and are of type void. See the DashBoardWindow man page and that of its superclass UpdateBoardWindow for more details.

C.8.2 The WhiteBoardWindow Class

The WhiteBoardWindow class is identical to the DashBoardWindow expect that it contains a WhiteBoard instead of a DashBoard. Therefore its constructor and callback have the following prototypes

```
WhiteBoardWindow ( char *name, WhiteBoardWindowCallback cb,
                Boolean scrolled =TRUE, Display *dpy =NULL );

typedef void ( *WhiteBoardWindowCallback )
    ( WhiteBoardWindow *dw, WhiteBoard *whiteboard, Widget form);
```

C.9 The CustomDialog Class

The CustomDialog class allows one to build popup dialogs containing a wide variety of user interface objects: labels, text entries, options menus, listboxes, toggle buttons and push buttons. A CustomDialog object is created with the following constructor.

```
CustomDialog ( Widget parent, char *name, int apply=0, int dismiss=1,
    int okay=0, unsigned char mode=XmDIALOG_FULL_APPLICATION_MODAL);
```

The first two arguments give the parent and X resource name for the dialog window. The next three arguments specify whether the dialog should have an Apply, Dismiss, and Okay button. The last argument specifies the modality of the dialog, which specifies what other windows can get focus and input when the dialog is popped up. The modality can be changed later using the setMode method (see below).

C.9.1 CustomDialog Items

A newly created CustomDialog has only a title label and the buttons specified in the constructor. New interface items can be placed on the dialog using the add method. Then they can be modified with the set method, queried with the get method and removed with the remove method.

```
enum CustomItemType {
    CmLabel, CmOptionMenu, CmToggle, CmEntry, CmButton, CmScale,
    CmListBox, CmSeparator, CmSubform, CmRadioBox, CmUser
};

int add ( CustomItemType type, int index, ...);
int set ( CustomItemType type, int index, ...);
int get ( CustomItemType type, int index, ...);
int remove ( CustomItemType type, int index );
```

The first argument of `CustomItemType` type specifies what kind of interface item to add. The second argument specifies an integer index that can be used later to manipulate and retrieve information about the item. All methods require both the type and index to identify an item, so items of different types can share the same index number. This is convenient since a label item and the entry item it labels can have the same index.

The $(\cdots)$ mean this method takes a variable number of arguments. For the add and set methods, the additional arguments consist of alternating CustomDialog item options and values for the options. For the get method, they consist of alternating CustomDialog item options and a pointer to memory in which to write the value. Finally, the list should be terminated with a `CmNULL`. The options are identified with the `CustomOptions` enumerated constants.

```
enum CustomOptions {
    CmNULL, CmString, CmName, CmValue, CmAlignment, CmSensitive,
    CmRow, CmColumn, CmWidth, CmHeight, CmCount, CmList,
    CmMax, CmMin, CmOrientation, CmMaxLength, CmShowValue,
    CmForm, CmWidget
};
```

Not all options are supported by all items. Table C.3 summarizes the valid values for options that all item types support. These mainly deal with item placement. The workarea of the dialog between the title and default buttons is divided up into a grid of rows and columns. An item is placed on this grid using the `CmColumn` and `CmRow` options. By default, an item is only one row in height and one column in width. This can be changed with the `CmHeight` and `CmWidth` options. If no `CmName` option is specified, it defaults to the string formed by the concatenation of the name of the item type and the index with the "Cm" prefix removed (eg. Button0, OptionMenu2, Label5).

As each item is added, the CustomDialog expands the grid as needed and determines the size of each grid cell for best fit. It does not shrink, but one can use the setGrid method to do it explicitly. Note that the grid only defines the relative size of all items to each other, not their explicit pixel size. This is determined dynamically at runtime by Motif. To set the explicit pixel width and height of the grid, one must use the main form's 'width' and 'height' X resources.

```
void setGrid(int widthInColumns, int heightInRows);
```

It may be necessary to play with the placement options a while to get a pleasant layout. One good strategy is to first draw a sketch of the the planned layout. Then draw equal spaced vertical lines on the drawing representing columns such that the narrowest item occupies one column. Similarly, draw equal spaced horizontal lines representing rows such that the shortest item occupies one row. Labels the rows and columns starting at zero from the top left grid cell. Now, you have a simple lookup table for each item's row, column, height and width.

Once your program is compiled, it is possible to experiment with other layout arrangements using X resources so no recompilation is required. For example, if one

option	*value type*	*description*
CmColumn	int	column to place item in
CmForm	int	subform to place item in
CmHeight	int	height of item in grid cells
CmName	char *	name to use for item's X resource specs
CmRow	int	row to place item in
CmSensitive	Boolean	whether item can be used/changed
CmWidth	int	width of item in grid cells

Table C.3: Summary of standard CustomDialog item options.

created a CustomDialog named "Buttons" that contained four button items, their layout could be controlled with the following X resources.

```
DAQCtrl*Buttons.main.Button0.cmRow:     0
DAQCtrl*Buttons.main.Button0.cmColumn:  0
DAQCtrl*Buttons.main.Button1.cmRow:     0
DAQCtrl*Buttons.main.Button1.cmColumn:  1
DAQCtrl*Buttons.main.Button2.cmRow:     1
DAQCtrl*Buttons.main.Button2.cmColumn:  0
DAQCtrl*Buttons.main.Button2.cmHeight:  2
DAQCtrl*Buttons.main.Button3.cmRow:     1
DAQCtrl*Buttons.main.Button3.cmColumn:  1
DAQCtrl*Buttons.main.Button3.cmHeight:  2
```

which gives a 2 by 2 layout of buttons with the bottom row of buttons being twice as high as the top row.

The options specific to each item type are summarized in Table C.4. At this time, one cannot modify or query the `CmList` or `CmCount` options using the **set** and **get** methods. Also, the `CmWidget` options cannot be changed using the **set** method.

The `CmUser` item type can be used by experienced X windows programmers to place arbitrary widget objects on the dialog. The **gridWidget** method, which takes no arguments, returns the widget to use as the parent for these user generated widgets.

The `CmSubform` item type is used to contain other items in its own independent grid of cells. However, the size of these subforms must be carefully specified with the height and width options when it is created so that it lays out properly in its

item	*option*	*value type*	*description*
CmButton	CmString	char *	character string for button label
CmEntry	CmMaxLength	int	maximum characters to display
	CmString	char *	character string to use for label
CmLabel	CmAlignment	unsigned char	XmALIGNMENT_BEGINNING, XmALIGNMENT_CENTER, or XmALIGNMENT_END
	CmString	char *	character string to use for label
CmListBox	CmCount	int	number of options in string list
	CmList	char **	list of strings for option labels
	CmValue	int	index of initial default option
CmOptionMenu	CmCount	int	number of options in string list
	CmList	char **	list of strings for option labels
	CmValue	int	index of initial default option
CmRadioBox	CmCount	int	number of radio buttons
	CmList	char **	list of strings for button labels
	CmOrientation	unsigned char	XmVERTICAL or XmHORIZONTAL
	CmValue	int	index of initially set button
CmScale	CmMin	int	minimum value of scale
	CmMax	int	maximum value of scale
	CmOrientation	unsigned char	XmVERTICAL or XmHORIZONTAL
	CmShowValue	Boolean	whether value should be shown on scale's top/side
	CmValue	int	initial value for scale
CmSeparator	CmOrientation	unsigned char	XmVERTICAL or XmHORIZONTAL
CmSubform			*no additional options*
CmToggle	CmString	char *	character string to use for label
	CmValue	int	initial state (1 or 0)
CmUser	CmWidget	Widget	user created widget to use for item

Table C.4: Summary of CustomDialog item specific options.

parent. If the subform contains four rows and three columns worth of items, then it should be given a height of four and a width of three. The main advantage of the subform is the 3D border it draws around itself which nicely groups its children.

The `CmSeparator` item type can be used to draw horizontal and vertical lines between rows and columns on the layout form. The position of horizontal lines will be along the top edge of its cell while vertical lines will be along the left edge. Also, separator items are not taken into account when determining the size needed for the grid. This will let one place a horizontal line on the row immediately after the last row with an item and not cause unnecessary blank space between the main form and okay, apply, and dismiss buttons.

As a efficient way of setting and getting the `CmValue` and `CmString` options, one my use the following convenience functions

```
int setValue(  CustomItemType type, int index, int value);
int setString( CustomItemType type, int index, char *string );
int getValue(  CustomItemType type, int index, int *valpointer );
int getString( CustomItemType type, int index, char **strpointer );
```

The contents of the `CmListBox` item type can be manipulated with the following methods after it has been created. See the man page for more details. List element positions start at 0 with -1 meaning the end of the list.

```
void listAdd( CustomItemType type,
        int index, char *str, int pos=-1 );
void listReplace( CustomItemType type,
        int index, char *str, int pos);
void listRemove( CustomItemType type,
        int index, int pos=-1, int count=1 );
void listClear( CustomItemType type, int index );
void listSelect( CustomItemType type, int index, int pos );
```

C.9.2 CustomDialog Callbacks

Handling the events of the dialog is the duty of the CustomDialog's callbacks which have the following prototype.

```
typedef void ( *CustomDialogCallback )
    ( CustomDialog *, CustomEventType, CustomEventData *);
```

The first argument is a pointer to the CustomDialog object itself. The second argument is the type of event which prompted the call to the callback. The third argument is a pointer to a structure containing information for the event. The possible events are identified with the `CustomEventType` enumerated constants.

```
enum CustomEventType {
    CmValueChange, CmManage,  CmApply, CmDismiss,
    CmButtonPress, CmTimeout, CmOkay
};
```

The event data structure, `CustomEventData`, is defined as follows.

```
typedef struct {
  CustomItemType  type;
  int             index;
  union {
    int     value;
    char   *string;
  };
} CustomEventData;
```

The conditions which produce the event, and thus a call to a CustomDialog callback if requested, are summarized in Table C.5 Only the `CmValueChange` and `CmButtonPress` events set the event data structure. The `type` and `index` fields are set to identify the affected item. For `CmValueChange` events, the `value` or `string` field is set to the new value of the item. Which of the two fields is used depends on whether that item has an integer or string value.

Actually, CustomDialog callbacks are never called with event type `CmOkay`. Instead separate calls for `CmApply` and `CmDismiss` are made, in that order. The main purpose for the `CmOkay` event type is in the rebindButton method (see below).

Callbacks are assigned to events using the registerClient method which has the following prototype.

```
void registerClient (CustomDialogCallback cb, int typemask );
```

event type	*cause*
`CmApply`	Apply or Okay button is pressed
`CmButtonPress`	a `CmButton` item is pressed
`CmDismiss`	Dismiss or Okay button is pressed or dialog popped down
`CmManage`	dialog is popped up
`CmOkay`	Okay button is pressed
`CmTimeout`	generated at end of each update period
`CmValueChange`	value of non-`CmUser` item changed by user

Table C.5: Summary of CustomDialog events.

The first argument is the callback function to register. The second argument is an event mask specifying for which events the callback should be called. An event mask is created by *OR*ing together the mask of each event desired. The name of the mask constant is the same as the event constant's with the word `Mask` appended (e.g. `CmApply` ⇒ `CmApplyMask`). An example call to registerClient might look like the following.

```
optdlg->registerClient ( &severityChange,
      CmManageMask | CmValueChangeMask | CmTimeoutMask );
```

This example would cause the severityChange callback to be called for `CmManage`, `CmValueChange` and `CmTimeout` events. Note that using `CmOkayMask` is effectively the same as using both `CmManage` and `CmDismiss`.

Sometimes one will want to "immediately" query the user for some information necessary to continue processing in the current function. In this case, the ask method should be used which will not return until the user dismisses the dialog. The ask method has the following prototype

```
int ask();
```

The return value will be one if the user pressed the Okay button to dismiss the dialog and zero otherwise, which can be used to decide if the answer should be processed. The values of the items in the dialog can then be obtained in the usual manner using the get method.

C.9.3 CustomDialog General Methods

Prototypes for other useful CustomDialog methods are listed below.

```
void   apply();
void   dismiss();
void   manage();
void   tick();
void   changePeriod(int msecs);
void   setTitle ( char *label );
void   setFocus( CustomItemType type, int index )
void   setMode  ( unsigned char mode );
Widget getWidget ( CustomItemType type, int index);
int    rebindButton ( int index, CustomEventType eventType);
```

The manage and dismiss methods pop up and pop down the dialog as well as generate `CmManage` and `CmDismiss` calls to the CustomDialog callbacks. The apply and tick methods generate `CmApply` and `CmTimeout` event calls to the callbacks, respectively. The `changePeriod` method sets the update period to the given number of milliseconds.

The setTitle method can be used to change the dialog title. The setFocus method gives the keyboard focus to the specified item. The setMode method can be used to change the modality of the dialog. The following list shows the valid values for the argument.

`XmDIALOG_MODELESS`	does not grab input
`XmDIALOG_FULL_APPLICATION_MODAL`	grabs input from the application
`XmDIALOG_SYSTEM_MODAL`	grabs input from all applications on X server

The getWidget method returns the widget ID of the specified item. The rebind-Button method cause the `CmButton` item with the given index to generate the event given as its second argument. Only the `CmButtonPress`, `CmApply`, `CmDismiss` and `CmOkay` events are valid options here.

Figure C.6 shows an example of a CustomDialog object for controlling an applications logfile. The code for settings up this object is shown in C.7 and the callback code is shown in C.8.

Figure C.6: Example of CustomDialog object.

```
// Simple options for binary option menu
char *OffOn[] = { "Off", "On" };

// Create dialog for changing logfile settings
CustomDialog *logfileSetup( Widget parent )
{
  CustomDialog *setdlg = new CustomDialog ( parent,
      "Log file", True, True );

  setdlg->add( CmLabel, 1, CmString, "Filename:",
      CmAlignment, XmALIGNMENT_END,
      CmRow, 0, CmColumn, 0, CmNULL );
  setdlg->add( CmEntry, 1, CmString, logger->logFile(),
      CmRow, 0, CmColumn, 1, CmNULL );
  setdlg->add( CmLabel, 2, CmString, "Log to File:",
      CmAlignment, XmALIGNMENT_END,
      CmRow, 1, CmColumn, 0, CmNULL );
  setdlg->add ( CmOptionMenu, 2, CmCount, 2,
      CmList, OffOn, CmRow, 1, CmColumn, 1,
      CmValue, logger->fileLogOnQuery(), CmNULL );

  setdlg->registerClient ( &logfileChange,
      CmManageMask | CmApplyMask );
}
```

Figure C.7: Example of code to setup a CustomDialog object.

```
// callback for CustomDialog to change Logger File Stats
void logfileChange ( CustomDialog *obj, CustomEventType type,
    CustomEventData *data)
{
  int  state, ret;
  char *file;

  // when managed, set the controls to current values
  if ( type == CmManage ) {
    obj->set( CmEntry, 1, CmString, theLogger->logFile(), CmNULL );
    obj->set( CmOptionMenu, 2, CmValue,
        (int) (theLogger->fileLogOnQuery()), CmNULL );
  }

  // get current values and set the Logger appropriately
  else if ( type == CmApply ) {
    obj->get( CmEntry, 1, CmString, &file, CmNULL );
    obj->get( CmOptionMenu, 2, CmValue, &state, CmNULL );

    ret = 1;
    if (file && strcmp ( file, theLogger->logFile() ) )
      ret = theLogger->setLogFile( file );

    if ( ret && state && file )
      ret = theLogger->fileLogOn();
    else theLogger->fileLogOff();

    obj->set ( CmOptionMenu, 2, CmValue,
        (int) (theLogger->fileLogOnQuery()), CmNULL );

    if (!ret) {
      char *buf = new char[20+STRLEN(file)];
      sprintf( buf, "Error opening %s.", file );
      theErrorDialogManager->post ( buf );
      delete buf;
    }
  }
}
```

Figure C.8: Example of a CustomDialog callback.

C.10 X Window Resources

C.10.1 Introduction

Only the attributes of the widgets absolutely necessary for a program's stable behavior are settable through the class methods defined above. Most of the widgets' characteristics (or X resources in the X-window programming lingo) are left to their defaults. In general, it is not really necessary for a program to hard code the color, size and fonts used by an application. A proper procedure would be for the program to define certain defaults for these widget attributes and allow the user to dynamically at run-time change them (at their risk) to their own preference.

This is precisely what the X resource feature of Motif allows. Whenever an X window program is run, before creating any widgets, a search is made on the file system for an app-defaults file. This file contains the programmer's suggested defaults for the program's widget resources if they differ from the standard Motif defaults. The app-defaults file is usually located in the /usr/lib/X11/app-defaults directory on unix systems. The name of the file must be the same as the application classname given to the program. This was defined by the first argument to the Application class's constructor. A user can define additional places to look for app-defaults file using the XUSERFILESEARCHPATH environment variable. An example of its use is as follows.

```
setenv XUSERFILESEARCHPATH /usr/local/lib/X11/app-defaults/%N
```

The %N is necessary and is substituted for the classname of the application. On VMS, the user defines the logical decw$user_defaults to point to the directory containing the app-defaults file (no %N needed). The name of the file is simply the classname with ".DAT" appended.

After parsing the app-default file, if present, the program will then search the user's own resource database which they may have set with the **xrdb** utility. On unix, this general corresponds to the /.Xdefaults or /.Xresources file. Only the app-defaults file will be described in this documentation, though.

Listing all the possible resource for each widget is beyond the scope of this document. There are several books available that describe the Motif widgets and their resources in detail. I recommend "The Motif Reference Manual" which is volume 6B in the O'Reilly X programming series. Section 2 in this book describes the Motif and Xt widget classes and their resources.

X resources are a lot like file pathnames where instead of directories there are widgets and child widgets, instead of files there are resources and instead of slashes, one uses dots and asterisks. Here is an example

```
DAQCtrl*Constants_popup.Constants.width:   350
```

This line out of an app-defaults file says the default width for the Constants widget, which is a direct child widget of Constants_popup, should be 350 pixels. The Constants_popup widget is itself a descendent of DAQCtrl, the main application itself. The difference here is that a dot means a direct descendent and an asterisk means merely a descendent. You can also think of the asterisk as a wild card in the same since as file matching on the shell command line.

If the DAQCtrl application happened to have two widgets in its hierarchy named Constants_popup (of course with different direct parents), the above example would affect both of them. It is also possible for more than one line of a app-defaults file to match the same widget resource because of the asterisk's wildcard nature. The more specific line is then used.

Any whitespace between the colon and value is ignored. This allows one to make the app-defaults file more readable by lining up the values to start at the same character column. Any line starting with an exclamation point is a comment.

Instead of specifying a specific widget name in a resource path, one can use a widget's class name. For instance

```
DAQCtrl*XmDialogShell.Constants.width:   350
```

states that any widget named Constants that is a child of a XmDialogShell in the DAQCtrl application should have a default width of 350 pixels.

Resources themselves have classes that they belong to. For instance, the width resource belongs to the XmCWidth class so one could use

```
DAQCtrl*XmDialogShell.Constants.Width:   350
```

which probably isn't very useful for this example since most resources of class **XmCWidth** are totally independent of each other and so shouldn't ever need to be the same width. Notice that the **XmC** prefix is removed for use of the class in the app-defaults file.

For those classes that use Motif widgets, such as CustomDialog or WhiteBoard, the man page for the class lists the widget hierarchy it contains up to the parent widget of class itself, if any. The charts describe both the widgets' name and class.

Refer to this chart to determine the widget "path" to use in specifying resources for a MLIB class's widgets.

C.10.2 Motif widget resources

I will only describe some of the more commonly used X resources below. Motif supports a very extensive list of resources for each of its widgets so if you do not see what you want below, consult the Motif reference manual.

Default fonts are best set using the ApplicationShell's **XmNbuttonFontList**, **XmNdefaultFontList**, **XmNlabelFontList**, and **XmNtextFontList** resources. Here is an example of there use.

```
DAQCtrl*defaultFontList:                -*-helvetica-*-r-*-*-14-140-*
```

This will set the default font of all widgets that use fonts. The default fonts for button, label, and text widgets can be overridden with the other resources.

To set the default colors, one simply uses a lines such as

```
DAQCtrl*background:                     gray75
DAQCtrl*XmDrawingArea.background:       seashell2
```

where one takes advantage of the wildcard nature of the asterisk. The above example sets the default background color for all the application's widgets to gray75 except for XmDrawingArea widgets which will have a background color of seashell2. Notice that the second setting overrides the first since it is more specific.

Setting the size of the main dashboard in the CtrlWindow is easily done using X resources as in the following example.

```
DAQCtrl*dashboardSW.height: 320
DAQCtrl*dashboardSW.width:  600
```

For a whiteboard window use **whiteboardSW** instead. The default timeout value for a dashboard or whiteboard can be set through a clever bit of subtlety. Remember that these updating widgets have a dialog associated with them so the user can adjust the timeout. By setting the default value of the scale widget in this dialog one can set the default timeout. The following example illustrates how this can be done.

```
DAQCtrl*dashboardSW*scale.value:  8
```

The default size of a CustomDialog is controlled by the size of its main XmForm widget. Therefore, if you have a dialog named "Constants", then you can set its default size using

```
DAQCtrl*Constants_popup.Constants.width:        350
DAQCtrl*Constants_popup.Constants.height:       500
```

Hopefully you remember that a single Cmd class instance can be used by several different interface objects. For example, a Cmd that generates a report could have both a menu item and a control panel button that invokes it. The label string used for both interfaces is take from the Cmd instance itself which means that they will have the same label. This is often not what is desired. Through X resources, it is possible to give each interface object a different label even though they share the same Cmd object. For example, if the Cmd object is named "report", and it has been added both to the CtrlWindow's control panel and to its Application menu, one can use the following.

```
DAQCtrl*XmMenuShell.Application.report.labelString: Generate Report
DAQCtrl*control.report.labelString: Report
```

This method is especially useful for PostCmd objects. Remember that a PostCmd when invoked will post its window or dialog if it is not already mapped and will unpost it if it is already mapped. So if one creates a PostCmd to add to a menu for posting a histogram dialog, one can also add the same PostCmd to the control panel of the dialog for dismissing it. If this PostCmd was named "posthist" and the histogram dialog was named "Histogram" the following X resources would be useful.

```
DAQCtrl*XmMenuShell.Application.posthist.labelString: Histogram
DAQCtrl*Histogram.control.posthist.labelString: Dismiss
```

You may have noticed that other Motif application you have seen will have special key strokes associated with menu items. Also, one character in a menu item is usually underlined and this character serves as a quick way of accessing menu items by the keyboard. Setting up this features in a MLIB application is best done in the app-defaults file with X resources. The underlined character is called a mnemonic and the special key stroke shown at the right of the item is an accelerator. The following resources show how one can set these from the app-defaults file.

```
DAQCtrl*menubar.Application.mnemonic: A
DAQCtrl*XmMenuShell.Application.Quit.mnemonic: Q
DAQCtrl*XmMenuShell.Application.Quit.accelerator: Ctrl<Key>Q
DAQCtrl*XmMenuShell.Application.Quit.acceleratorText: Ctrl+Q
```

C.10.3 MLIB class resources

Some MLIB classes are programmed to have X resources of their own unrelated to standard widget resources. The Logger is an example of such a class. It defines the **maxChars**, **fileLogOn**, and **logfile** resources. These can be set as in the following example.

```
DAQCtrl*logger*maxChars:        1000
DAQCtrl*logger*fileLogOn:       True
DAQCtrl*logger*logfile:         test.log
```

Logger severities can have their default values set from X resources also. If the logger has a severity class called "BEAM", then one could use the following.

```
DAQCtrl*bellSeverity:           3
DAQCtrl*BEAM*bellSeverity:      1
```

These resources set the default bell severity level to three for all severity classes except for "BEAM" which is set to one. The resources **screenSeverity** and **fileSeverity** also exist.

The DashBoard class defines the **columnWidth** resource for setting the default column width for items. The UpdateBoard class (and thus DashBoard and WhiteBoard) defines a **font** resource for setting the default font.

The layout of items in the CustomDialog can be controlled from X resources in the app-defaults file. Unlike regular resources, these resources will override the hard coded values given in the CustomDialog add and set methods. This feature is different because these resources are for use mainly by the programmer in designing a pleasant layout for a dialog without having to recompile to make every change. Once the desired layout is found, the values should be transfered to the appropriate add call and the layout resource removed from the app-defaults file. The resources available are **cmRow**, **cmColumn**, **cmWidth**, **cmHeight** and **cmForm**.

The names of items in the CustomDialog, if not specified using the **CmName** option in the add method, will default to the type of item with the "Cm" prefix removed and the given index number appended (e.g. Button0, Entry2, ListBox12). The following resources define a layout for an example dialog named "Buttons" with four button items.

```
DAQCtrl*Buttons.main.Button0.cmRow:     0
DAQCtrl*Buttons.main.Button0.cmColumn:  0
DAQCtrl*Buttons.main.Button1.cmRow:     0
DAQCtrl*Buttons.main.Button1.cmColumn:  1
DAQCtrl*Buttons.main.Button2.cmRow:     1
DAQCtrl*Buttons.main.Button2.cmColumn:  0
DAQCtrl*Buttons.main.Button2.cmHeight:  2
DAQCtrl*Buttons.main.Button3.cmRow:     1
DAQCtrl*Buttons.main.Button3.cmColumn:  1
DAQCtrl*Buttons.main.Button3.cmHeight:  2
```

For development purposes the Application class has a useful method, printAppTree, for printing out the applications widget hierarchy. It takes as its single argument a ostream object. Therefore if one has included the iostream.h include file, one can have a command that makes the following call

```
theApplication->printAppTree( cerr );
```

and the widget hierarchy will be printed to standard error. Knowledgeable C++ programmers can see that this can easily be used with an ofstream object for saving the hierarchy to a file or with a ostrstream object for putting the hierarchy into a string buffer. (*Note: at this time, the full hierarchy does not seem to get printed on VMS for some unknown reason*)

Bibliography

[1] M. J. Alguard *et al.*, Phys. Rev. Lett. **37**, 1261 (1976).

[2] G. Baum *et al.*, Phys. Rev. Lett. **45**, 2000 (1980).

[3] J. Ashman *et al.*, Phys. Lett. **B206**, 364 (1988).

[4] J. Ashman *et al.*, Nucl. Phys. **B328**, 1 (1989).

[5] B. Adeva *et al.*, Phys. Lett. **B302**, 533 (1993).

[6] D. Adams *et al.*, Phys. Lett. **B329**, 399 (1994).

[7] D. Adams *et al.*, Phys. Lett. **B336**, 125 (1994).

[8] D. Adams *et al.*, Phys. Lett. **B357**, 248 (1995).

[9] P. L. Anthony *et al.*, Phys. Rev. Lett. **71**, 959 (1993).

[10] K. Abe *et al.*, Phys. Rev. Lett. **74**, 346 (1995).

[11] K. Abe *et al.*, Phys. Rev. Lett. **75**, 25 (1995).

[12] K. Abe *et al.*, Phys. Lett. **B364**, 61 (1995).

[13] K. Abe *et al.*, Phys. Rev. Lett. **76**, 587 (1996).

[14] S. B. Gerasimov, Yad. Fiz. **2**, 598 (1965).

[15] S. B. Drell and A. C. Hearn, Phys. Rev .Lett. **16**, 908 (1966).

[16] J. Ellis and R. Jaffe, Phys. Rev. **D9**, 1444 (1974).

[17] J. Ellis and R. Jaffe, Phys. Rev. **D10**, 1669 (1974).

[18] M. Anselmino *et al.*, Phys. Rept. **261**, 1 (1995).

[19] J. D. Björken *et al.*, Phys. Rev. **185**, 1975 (1969).

[20] C. Callan and D. Gross, Phys. Rev. Lett. **22**, 156 (1969).

[21] K. G. Wilson, Phys. Rev. **179**, 416 (1974).

[22] S. Wandzura and F. Wilczek, Phys. Lett. **B72**, 195 (1977).

[23] T. Pussierx and R. Windmolders, SMC-NOTE, vol. 93, 1993.

[24] K. Kotthoff *et al.*, Nucl. Phys. **A264**, 484 (1976).

[25] R. Machleidt *et al.*, Phys. Rep. **149**, 1 (1987).

[26] M. Lacombe *et al.*, Phys. Rev. **C21**, 861 (1980).

[27] R. G. Roberts, *The Stucture of the Proton* (Cambridge University Press, Cambridge, 1990).

[28] J. D. Björken, Phys. Rev. **148**, 16 (1966).

[29] S. A. Larin and J. A. M. Vermaseren, Phys. Lett. **B259**, 345 (1991).

[30] R. Feynman, *Photon-Hadron Interactions* (W. A. Benjamin, Inc., Reading, Mass., 1972).

[31] J. M. Bauer, Ph.D. thesis, University of Massachusetts Amherst, 1996.

[32] M. S. Schmelling and R. D. S. Denis, Phys. Lett. **B329**, 393 (1994).

[33] S. Narison, Report No. CERN-TH-7188/94, 1994.

[34] P. D. Group *et al.*, Phys. Rev. D **50**, 1173 (1994).

[35] S. A. Larin, Phys. Lett. **B334**, 192 (1994).

[36] H. J. Lipkin, Phys. Lett. **B337**, 157 (1994).

[37] H. Burkhardt and W. N. Cottingham, Ann. Phys. **56**, 453 (1970).

[38] J. Soffer and O. V. Teryaev, Phys. Rev. D **51**, 25 (1995).

[39] V. D. Burkert and B. L. Ioffe, Phys. Lett. **B296**, 223 (1992).

[40] V. D. Burkert and B. L. Ioffe, CEBAF Preprint PR-93-034, 1993.

[41] R. Alley *et al.*, Report No. SLAC-PUB-6489, 1995.

[42] M. Swartz *et al.*, Report No. SLAC-PUB-6467, 1994.

[43] H. R. Band and R. Prepost, E143 Technical Note #110, 1996.

[44] L. G. Levchuk, Nucl. Inst. Meth. **A345**, 496 (1994).

[45] H. R. Band, E143 Technical Note #84, 1994.

[46] W. Meyer, in *Proceedings of the* 4^{th} *International Workshop on Polarized Target Materials and Techniques* (1984).

[47] M. L. Seely, Ph.D. thesis, Yale University, 1982.

[48] T. D. Averett, Ph.D. thesis, University of Virginia, 1995.

[49] C. Jeffries, Ann. Rev. Nuc. Sci. **14**, 101 (1964).

[50] D. Zimmermann, University of Virginia Polarized Target Group Report, 1993.

[51] T. J. Liu, Ph.D. thesis, University of Virginia, 1996.

[52] V. Breton, Y. Robin, and F. Tamin, E143 Technical Note #23, 1993.

[53] V. Breton *et al.*, Nucl. Instr. and Meth. **A362**, 478 (1995).

[54] V. Breton and P. Grenier, E143 Technical Note #22, 1993.

[55] J. J. Aubert *et al.*, Phys. Lett. **B123**, 275 (1983).

[56] J. Bauer and O. Rondon, E143 Technical Note #71, 1994.

[57] S. Kuhn *et al.*, E143 Technical Note #111, 1996.

[58] V. D. Burkert and Z.-J. Li, Phys. Rev. D **47**, 46 (1993).

[59] L. M. Stuart, Report No. SLAC-PUB-6305, 1996.

[60] T. Kukhto and N. Shumeiko, Nuc. Phys. **B219**, 412 (1983).

[61] I. Akusevich and N. Shumeiko, J. Phys. **G20**, 513 (1994).

[62] C. Keppel, Ph.D. thesis, American University, 1994.

[63] X. Ji and P. Unrau, Phys. Lett. **B333**, 228 (1994).

[64] V. Bernard, N. Kaiser, and U. G. Meissner, Phys. Rev. D **48**, 3062 (1993).

[65] V. Burkert *et al.*, CEBAF-PROPOSAL-91-023, 1991.

[66] S. Kuhn *et al.*, CEBAF-PROPOSAL-93-009, 1993.

[67] Z. E. Meziani *et al.*, CEBAF-PROPOSAL-94-010, 1994.

[68] E. W. Hughes *et al.*, SLAC-PROPOSAL-E-154, 1993.

[69] R. G. Arnold *et al.*, SLAC-PROPOSAL-E-155, 1993.