Untersuchungen zur automatisierten Justierung der Spiegelfacetten der H·E·S·S· Cherenkov–Teleskope

II. Institut für Experimentalphysik
Universität Hamburg

Diplomarbeit
im Studiengang Physik
vorgelegt von

René Cornils

Hamburg
Januar 2001

dem Lichtmädchen
das den Schrat aus seiner Waldhöhle führte

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Erde erreicht ein stetiger Strom sehr hochenergetischer kosmischer Teilchen, die Auskunft über eine Vielzahl astrophysikalischer Fragestellungen geben können. Seit ihrer Entdeckung widmen sich daher eine Reihe unterschiedlicher Experimente ihrer systematischen Untersuchung.

Die HEGRA–Kollaboration[1] betreibt auf der Kanarischen Insel La Palma u. a. ein Experiment zum Nachweis sehr hochenergetischer kosmischer Gamma–Strahlung im Bereich von $5 \cdot 10^{11}\,\mathrm{eV}$ (500 GeV) bis zu einigen $10^{13}\,\mathrm{eV}$ (10 TeV).

Die Erdatmosphäre ist im interessierenden Energiebereich optisch dick, so daß diese Photonen mit erdgebundenen Experimenten nicht direkt nachweisbar sind. Sie erzeugen in der Atmosphäre Kaskaden sekundärer Teilchen – sogenannte ausgedehnte elektromagnetische Luftschauer –, welche die Atmosphäre anregen, Cherenkov–Licht zu emittieren. Der Effekt der Schauerbildung verhindert zwar einen direkten Nachweis der kosmischen Gamma–Photonen, jedoch lassen sich aus der Signatur der dabei erzeugten Cherenkov–Blitze indirekt Energie, Einfallsrichtung und Beschaffenheit der primären Teilchen bestimmen. Dazu wird das Cherenkov–Licht mit speziellen Spiegelteleskopen[2] winkelabbildend auf sich in der Brennebene der Spiegel befindende hochempfindliche Kameras aus Photovervielfachern fokussiert, mit deren Hilfe die sehr kurzen Lichtblitze vermessen werden.

Um kostengünstig möglichst große Spiegelflächen realisieren zu können, bestehen die Spiegel von Cherenkov–Teleskopen aus vielen relativ kleinen Spiegelfacetten mit sphärischem Schliff. Bei den System–Teleskopen des HEGRA–Experiments bilden so jeweils dreißig Spiegelfacetten mit einem Durchmesser von 60 cm den $8{,}5\,\mathrm{m}^2$ großen Gesamtspiegel eines Teleskops. Die Justierung der Spiegelfacetten erfolgt manuell.

Eines der nun anstehenden Nachfolgeprojekte des HEGRA–Experimentes ist das Victor H·E·S·S·–Projekt[3]. Aufbauend auf den Erfahrungen mit dem Experiment auf La Palma ist ein zunächst aus vier – in einer zweiten Ausbaustufe schließlich aus bis zu

[1] HEGRA steht als Abkürzung für *High Energy Gamma Ray Astronomy*.

[2] winkelabbildende Cherenkov–Teleskope, engl. *Imaging Atmospheric Cherenkov Telescopes* (IACT)

[3] Victor Franz (Francis) Hess wurde 1936 in Würdigung seiner Entdeckung der kosmischen Strahlung mit dem Nobelpreis für Physik ausgezeichnet; zudem steht H·E·S·S· als Akronym für *High Energy Stereoscopic System*.

sechzehn – Teleskopen bestehendes stereoskopisches System abbildender Cherenkov–Teleskope geplant. Der ca. $108\,\mathrm{m}^2$ große Spiegel eines jeden Teleskops des H·E·S·S·–Projektes wird aus fast 400 Spiegelfacetten gebildet, für die eine manuelle Justierung mit vertretbarem Aufwand nicht mehr möglich ist. Deshalb ist bei jeder Spiegelfacette eine motorische Ausrichtung vorgesehen.

Mittels zweier sogenannter Aktuatoren können die Spiegelfacetten in gewissen Bereichen positioniert werden. Zur Ansteuerung der Aktuatormotoren wurden spezielle Elektronikkomponenten von Mitarbeitern der Universität Hamburg entwickelt.

Aufgabe der vorliegenden Arbeit war es, die Hardware zur Ansteuerung der Aktuatormotoren durch Auswahl geeigneter Komponenten zu vervollständigen, sie in Betrieb zu nehmen und ausgiebig zu testen. Des weiteren galt es, die zur Justierung bestimmte Software zu konzipieren und – soweit möglich – zu implementieren, denn einige Teile der Software waren zum Test der Hardwarekomponenten unabdingbar.

Kapitel 2

Astrophysik mit abbildenden Cherenkov–Teleskopen

2.1 Die kosmische Strahlung und ihre Quellen

Im Jahre 1912 entdeckte der österreichische Physiker Victor Franz (Francis) Hess in Ballonexperimenten die kosmische Strahlung [Hess 12], wofür er 1936 mit dem Nobelpreis für Physik ausgezeichnet wurde.

Hess untersuchte den Ionisationsgrad der Atmosphäre mit zunehmender Höhe über dem Erdboden. Nach der zu dieser Zeit vorherrschenden Meinung galt die Erde als Quelle der Ionisation, weshalb die Ionisation der Atmosphäre mit zunehmender Höhe abnehmen sollte. Bis zu einer Höhe von etwa einem Kilometer über dem Erdboden traf dies auch zu. Hess fand jedoch entgegen seiner Erwartung ab dieser Höhe eine rapide ansteigende Ionisation. Er folgerte daraus

> [...] a radiation of very high penetrating power enters our atmosphere from above [...]

Seither ist die kosmische Strahlung Gegenstand intensiver Forschung, bietet sie doch eine neue Informationsquelle über Objekte und Prozesse im Universum, die bis dahin wenig oder gar nicht zugänglich waren.

Die kosmische Strahlung konstituiert sich aus einer Vielzahl verschiedener Teilchen. Zum einen sind dies die geladenen Teilchen wie etwa Elektronen, Protonen, Alphateilchen und schwerere Kerne, zum anderen die ungeladene Komponente bestehend aus Photonen, Neutrinos und Neutronen.

Die geladenen Teilchen verlieren auf ihrem Weg zur Erde die Information über ihre Herkunftsrichtung, da sie durch interstellare Magnetfelder abgelenkt werden. Sie eignen sich daher nicht, um Rückschlüsse auf eine bestimmte Emissionsregion ziehen zu können. Sieht man von Streu– und Absorptionsprozessen einmal ab, erreichen die ungeladenen Teilchen die Erde dagegen aus Richtung der sie emittierenden Quelle. Mit ihnen ist also eine detaillierte Untersuchung der astrophysikalischen Bedingungen der Erzeugungsregion möglich.

Photonen eignen sich besonders gut für das Studium kosmischer Objekte, da Neutrinos wegen ihres sehr kleinen Wechselwirkungsquerschnitts schwer nachzuweisen sind und Neutronen bei einer mittleren Lebensdauer von 887 s nur bei sehr hohen Energien weite Strecken zurücklegen können.

Als erfolgreiche Nachweismethode für sehr hochenergetische Photonen hat sich die abbildende Cherenkov–Technik erwiesen, die bei den in Kapitel 2.2.2 beschriebenen Cherenkov–Teleskopen zum Einsatz kommt.

2.1.1 Das nichtthermische Universum

Das Photonenspektrum sich im thermodynamischen Gleichgewicht befindender Objekte folgt der typischen Planckverteilung

$$\frac{dN(E)}{dE} \quad \propto \quad \frac{E^3}{e^{E/kT}-1} \tag{2.1}$$

des thermischen Schwarzkörperspektrums. Dem Objekt kann eine thermodynamische Temperatur zugeordnet werden. Dabei bedeutet der Begriff Objekt nicht zwangsläufig ein gesamtes System; herrscht in einem gewissen Bereich eines Systems ein hinreichend gutes thermodynamisches Gleichgewicht, kann diesem eine *lokale Temperatur* zugeordnet werden.

Als Beispiel eines solchen Systems sei hier ein Stern angeführt. Obwohl das Gesamtsystem (der Stern) sich nicht im thermodynamischen Gleichgewicht befindet, kann in guter Näherung eine lokale Temperatur eingeführt werden (eine schöne Darstellung findet sich z. B. in [Sche 90]). So werden denn auch Sterne u. a. nach ihrer *Oberflächen–* bzw. Effektivtemperatur klassifiziert.

Entgegen dem Energiespektrum normaler Sterne, liegt einigen Objekten im Universum kein (thermisches) Schwarzkörperspektrum zugrunde. Der Emissionsregion kann somit keine thermodynamische Temperatur zugeordnet werden; sie befindet sich nicht im thermodynamischen Gleichgewicht. Man spricht in diesem Fall deshalb oft vom *nichtthermischen Universum*.

Das Energiespektrum der hochenergetischen kosmischen Strahlung über etwa 10^6 eV (1 MeV) ist von dieser Art. Es folgt über viele Größenordnungen einem Potenzgesetz der Form

$$\frac{dN(E)}{dE} \quad \propto \quad E^{-\gamma} \tag{2.2}$$

mit konstantem Exponenten γ.

In Abbildung 2.1 ist das Spektrum der geladenen kosmischen Strahlung oberhalb von 10^{12} eV (1 TeV) gezeigt. Dabei ist auszumachen, daß das Spektrum nicht über den gesamten Bereich durch einen konstanten Exponenten γ beschrieben werden kann, sondern Abknickstellen bei etwa $4 \cdot 10^{15}$ eV (genannt *Knie*) und $3 \cdot 10^{18}$ eV (genannt *Knöchel*) aufweist. Mehr zur geladenen kosmischen Strahlung findet sich in [Röhr 00].

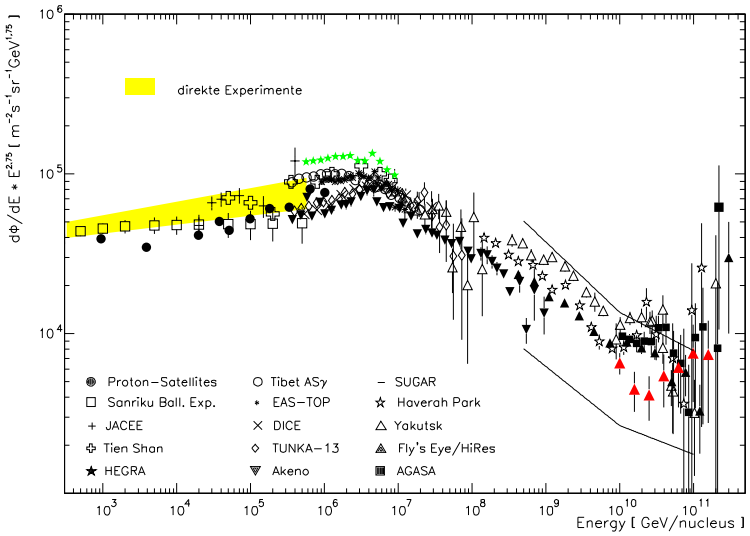


Abbildung 2.1: *Mit $E^{2.75}$ gewichteter differentieller Fluß der geladenen kosmischen Strahlung [Röhr 00]. Das um HEGRA–Daten erweiterte Kompilat entstammt [Wieb 98].*

2.1.2 Erzeugungsmechanismen hochenergetischer γ–Strahlung

Cherenkov–Teleskope sind Nachweisinstrumente für sehr hochenergetische Photonen, weshalb die zu ihrer Erzeugung beitragenden Prozesse kurz vorgestellt werden sollen. Die Darstellung folgt dabei [Long 92] und [Long 94].

Zerfall neutraler π–Mesonen

Neutrale Pionen (π^0) können aus inelastischen Stößen von Protonen mit Protonen und schwereren Kernen entstehen. Bei einer sehr kurzen mittleren Lebensdauer von $8{,}4 \cdot 10^{-17}\,\mathrm{s}$ zerfallen sie mit einer Wahrscheinlichkeit von fast 99 % elektromagnetisch in zwei Gammaquanten:

$$\pi^0 \quad \rightarrow \quad \gamma\,\gamma \tag{2.3}$$

Bremsstrahlung

Mit *Bremsstrahlung* wird die Strahlung bezeichnet, die bei der Beschleunigung von Elektronen im elektrostatischen Feld von Ionen bzw. Atomen erzeugt wird. Die mittlere Energie $\langle\hbar\omega\rangle$ der von relativistischen Elektronen der Energie E_e erzeugten Photonen ist

$$\langle\hbar\omega\rangle \quad = \quad \frac{1}{3}E_\mathrm{e} \tag{2.4}$$

wobei Photonen mit Energien bis zu E_e erzeugt werden können.

Synchrotronstrahlung

Werden relativistische geladene Teilchen in starken Magnetfeldern beschleunigt, emittieren sie Strahlung. Dieser Effekt wurde in Betatron–Experimenten entdeckt und mit *Synchrotronstrahlung* benannt. Die Energie $\hbar\omega$, bei der die meiste Strahlung eines im Magnetfeld beschleunigten relativistischen Elektrons emittiert wird, ist

$$\hbar\omega \quad \approx \quad \gamma^2 h\nu_{\mathrm{g}} \tag{2.5}$$

wobei ν_{g} die nichtrelativistische gyromagnetische Frequenz

$$\nu_{\mathrm{g}} \quad = \quad \frac{eB_{\perp}}{2\pi m_{\mathrm{e}}} \tag{2.6}$$

bezeichnet.

Inverse Comptonstreuung

Ein weiterer wichtiger Prozeß für die Hochenergie–Astrophysik ist der der sogenannten *inversen Comptonstreuung*. Dieser beschreibt die Streuung ultrarelativistischer Elektronen an niederenergetischen Photonen ($\hbar\omega \ll \gamma m_{\mathrm{e}}c^2$), wobei die Elektronen Energie an die Photonen abgeben. Die mittlere Energie $\langle\hbar\omega\rangle$ des gestreuten Photons ergibt sich zu

$$\langle\hbar\omega\rangle \quad = \quad \frac{4}{3}\gamma^2\beta^2\hbar\omega_0 \quad \approx \quad \frac{4}{3}\gamma^2\hbar\omega_0 \tag{2.7}$$

mit einer maximalen Energie des gestreuten Photons von

$$(\hbar\omega)_{\max} \quad = \quad \frac{4}{3}\gamma^2(1+\beta)^2\hbar\omega_0 \quad \approx \quad 4\gamma^2\hbar\omega_0 \tag{2.8}$$

2.1.3 Objekte mit nichtthermischen Emissionsregionen

Seit der Entdeckung der kosmischen Strahlung werden einige Objekte als Quellen derselben favorisiert. Zwar ist der Ursprung der geladenen kosmischen Strahlung bislang nicht vollständig verstanden, die Beschleunigung dieser Teilchen wird jedoch in der Regel von der Erzeugung sehr hochenergetischer Photonen begleitet, die durch die in Kapitel 2.1.2 dargestellten Mechanismen produziert werden können. Mit verschiedenen Experimenten konnten einige Objekte als Quellen sehr hochenergetischer Photonen identifiziert werden.

Innerhalb unserer Galaxie sind Supernovaüberreste und Pulsare als Quellen bekannt, während aktive Galaxienkerne extragalaktische Quellen sehr hochenergetischer Photonen sind. Eine Sonderrolle nehmen *Gamma Ray Bursts* ein, deren Natur bislang nicht vollständig geklärt werden konnte.

Die angesprochenen Objektklassen sollen im folgenden näher betrachtet werden.

2.1.3.1 Überreste von Supernovaexplosionen und Pulsare

Ein mögliches Endstadium in der Sternentwicklung wird nach einer gewaltigen Explosion des Vorgängersterns erreicht, welche mit *Supernova* bezeichnet wird. Dabei wird ein großer Teil des Vorgängersterns in das interstellare Medium geschleudert, wobei sich Schockfronten ausbilden, an denen Teilchen auf hohe Energien beschleunigt werden können. Bleibt nach einer Supernovaexplosion ein stellarer Rest bestehen, so tritt dieser in der Regel als Neutronenstern in Erscheinung. Die Theorie der Sternentwicklung unter Einbeziehung des Phänomens der Supernovae ist z. B. in [Sche 90] nachzulesen.

Eng verknüpft mit Supernovaüberresten sind *Pulsare*, bei denen es sich um rotierende Neutronensterne mit gegenüber der Rotationsachse verkippter Magnetfeldachse handelt. Geladene Teilchen werden entlang der Magnetfelder auf sehr hohe Energien beschleunigt und emittieren Synchrotronstrahlung, welche den Pulsar gebündelt verläßt. Durch die sehr gleichförmige Rotation[1] der Pulsare streichen die Emissionsregionen in nahezu konstanten Zeitintervallen – gleich dem Signal eines Leuchtturms – unter dem Blick des Beobachters vorbei. Dies führt zu den periodischen Strahlungspulsen, die für die Namensgebung der Pulsare verantwortlich zeichnen. Eine kurze Einführung in die Funktionsweise von Pulsaren findet sich in [Kunz 00], während [Lyn 93] eine ausführlichere Darstellung bietet.

Abgesehen von der direkten Emission können hochenergetische Photonen zudem durch Wechselwirkung der die Magnetosphäre des Pulsars verlassenden geladenen Teilchen mit dem expandierenden Rest des Vorgängersterns erzeugt werden. Der prominenteste Vertreter dieser Objektklasse ist der Krebsnebel [Ahar 00], dessen ungepulstes Photonspektrum in Abbildung 2.2 dargestellt ist. Bei ihm handelt es ich um ein sogenanntes *Plerion*, ein – im Gegensatz zu schalenförmigen Überresten – ausgefüllter Supernovaüberrest mit einem Pulsar im Zentrum.

2.1.3.2 Kerne aktiver Galaxien

Kerne aktiver Galaxien (AGN für engl. *active galactic nuclei*) sind extragalaktische Objekte, in deren Zentren supermassive Schwarze Löcher mit Massen von etwa 10^9 Sonnenmassen vermutet werden. Sie bilden kollimierte relativistische Plasmaströme – genannt *Jets* – in den intergalaktischen Raum aus, deren Ausdehnung die der sie emittierenden Galaxien bei weitem übersteigt. Die Jetregionen gelten als Quelle der hochenergetischen Strahlung, die die Erde aus Richtung dieser Objekte erreicht.

Eine Unterklasse aktiver Galaxienkerne sind die sogenannten *Blazare*. Bei diesen Objekten ist die Blickrichtung des Beobachters nahezu zentral auf eine der Jetachsen gerichtet, womit sich die im Jet erzeugten Teilchen vorzugsweise in Richtung des Beobachters bewegen. Zwei Vertreter dieser Klasse sind die Galaxien Markarian 421 [Ahar 99] und Markarian 501 [Ahar 01]. Zwischen längeren Phasen der Ruhe weisen sie

[1] Pulsare weisen i. a. ein sehr langsames Anwachsen der Rotationsperiode auf. Da dieser Energieverlust etwa von der Größenordnung der abgestrahlten Synchrotronenergie ist, wird die Rotation als (hauptsächliche) Energiequelle der Pulsare angesehen. Zudem sind bei einigen Pulsaren sprunghafte Periodenverkürzungen beobachtet worden. Beide Effekte sind jedoch i. a. sehr klein.

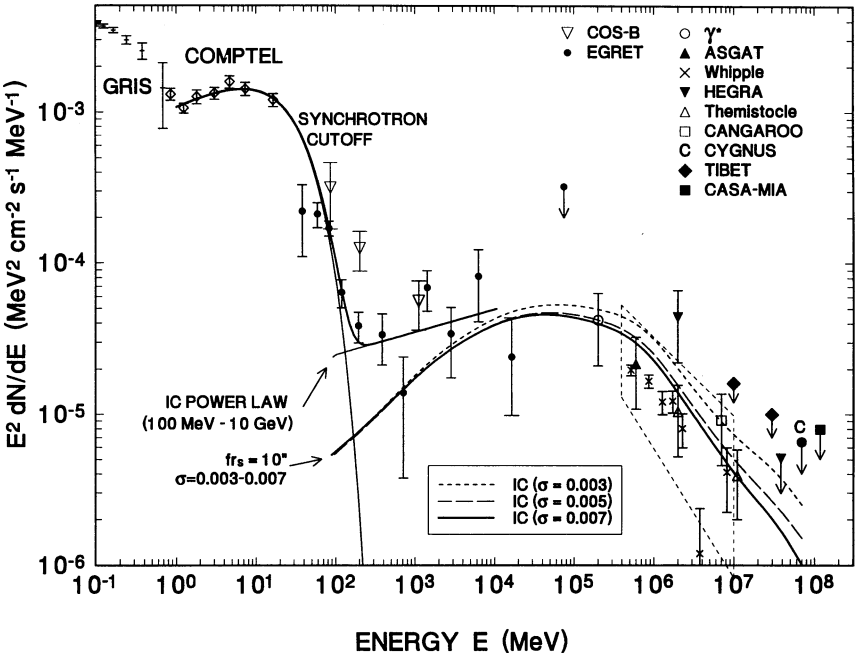


Abbildung 2.2: *Mit E^2 gewichtetes differentielles Photonspektrum (ungepulste Komponente) des Krebsnebels [Jage 96].*
Deutlich sind im Verlauf die in unterschiedlichen Energiebereichen zum Tragen kommenden Erzeugungsmechanismen auszumachen. Bis zu etwa 100 MeV wird das Spektrum von Synchrotronstrahlung dominiert, wobei der Verlauf durch die Energie der Elektronen und der Stärke des Magnetfelds bestimmt wird. Für die höherenergetischen Photonen wird als Erzeugungsprozeß das sogenannte Synchrotron–Self–Compton bzw. SSC–Modell diskutiert. Das SSC–Modell beschreibt die inverse Comptonstreuung von Elektronen an Photonen, wobei diese durch Synchrotronstrahlung derselben Elektronen–Population erzeugt wurden. Dabei können die Synchrotronphotonen zu noch höheren Energien gelangen.

immer wieder Zeiten sehr hoher Aktivität auf, während der sie trotz ihrer relativ hohen Distanz zur Erde zu den lichtstärksten Objekten im TeV–Energiebereich werden.

2.1.3.3 Gamma Ray Bursts

Mit *Gamma Ray Burst* wird ein Phänomen bezeichnet, bei dem für kurze Zeit ein sehr heller Ausbruch von Gammastrahlung am Himmel erscheint, der nach einer Abklingphase wieder vollständig verschwindet. Die Ausbrüche von *Gamma Ray Bursts*, während der sehr große Energiemengen in sehr kurzer Zeit freigesetzt werden, erscheinen isotrop am Himmel verteilt. Dies wird als Hinweis auf einen extragalaktischen Ursprung gewertet, da die Materie innerhalb unserer Galaxie stark anisotrop verteilt ist. Das Phänomen der *Gamma Ray Bursts* ist somit von kosmologischem Interesse.

Zwar konnte die Natur von *Gamma Ray Bursts* bislang nicht vollständig geklärt werden, in den letzten Jahren wurden jedoch enorme Fortschritte auf diesem Gebiet erzielt. Zudem konnte die Milagro–Kollaboration bei dem *Gamma Ray Burst* PSR 970417a einen ersten Hinweis auf Gammastrahlung im TeV–Energiebereich vermelden [Tota 00].

Mit Cherenkov–Teleskopen ist bisher kein Nachweis gelungen, da das nur kurzzeitige Andauern der Ausbrüche eine zeitnahe Beobachtung erforderlich macht, was jedoch nur selten gewährleistet werden kann.

2.2 Nachweis hochenergetischer kosmischer γ–Strahlung

Zum Nachweis hochenergetischer kosmischer γ–Strahlung kommen sowohl satellitengestützte als auch erdgebundene Detektoren zum Einsatz. Da die Teilchenflüsse mit zunehmender Energie der Teilchen stark abnehmen, müssen zum Nachweis sehr hoher Energien große Detektorflächen zum Einsatz kommen. Satellitengestützten Experimenten ist aus Kostengründen dadurch eine obere Nachweisschwelle von (derzeit) einigen 10^{10} eV (10 GeV) gesetzt; Energien oberhalb dieser Schwelle werden deshalb mit erdgebundenen Experimenten untersucht.

Ein direkter Nachweis der primären Photonen auf dem Erdboden ist jedoch nahezu unmöglich, da sie diesen i. a. nicht erreichen. Die Erdatmosphäre ist im betreffenden Energiebereich optisch dick, so daß die Primärteilchen bei ihrem Durchgang Kaskaden sekundärer Teilchen erzeugen, welche *ausgedehnte Luftschauer* genannt werden.

2.2.1 Ausgedehnte Luftschauer

Die kosmischen Photonen können in den Feldern der Elektronenhüllen und Kerne atmosphärischer Atome hochenergetische Elektron–Positron–Paare durch Paarbildung erzeugen, welche wiederum hochenergetische Photonen durch Bremsstrahlung erzeugen können. Sukzessive entwickelt sich so eine elektromagnetische Kaskade sekundärer Teilchen, deren Zahl in etwa exponentiell mit der Tiefe ansteigt.

Die für die Entwicklung eines elektromagnetischen Schauers charakteriktische Größe ist die Strahlungslänge X_0

$$\left(\frac{dE}{dx}\right)_{\text{rad}} = -\frac{E}{X_0} \tag{2.9}$$

die den Energieverlust von Elektronen durch Bremsstrahlung bei der Durchquerung eines Mediums beschreibt. Die Strahlungslänge ist somit ein Maß für die Schichtdicke, die die mittlere Energie der Elektronen durch Strahlungsverluste um den Faktor e reduziert. Die Atmosphäre weist auf Meeresniveau eine Dicke von ca. 28 Strahlungslängen auf.

Die maximale Anzahl der an einem Schauer beteiligten Teilchen wird etwa erreicht, wenn der Energieverlust der Elektronen durch Ionisation von der gleichen Größenordnung ist wie der durch Bremsstrahlung. Ab dieser Tiefe wird der Energieverlust der Elektronen durch Ionisation dominierend und der Schauer klingt langsam ab. Das Schauermaximum eines primären Photons mit einer Energie von 1 TeV liegt so typischerweise in einer Höhe von 10 km.

Hadronisch induzierte Luftschauer weisen neben hadronischen Kaskaden auch elektromagnetische Subkaskaden auf, weshalb die obenstehenden Betrachtungen für sie ebenfalls zum Tragen kommen.

Um die Teilchen der kosmischen γ–Strahlung trotz der Wandlung in ausgedehnte Luftschauer noch nachweisen zu können, wird sich der folgende dabei auftretende Effekt zunutze gemacht: Durchquert ein geladenes Teilchen ein Dielektrikum mit einer Geschwindigkeit, die über der Phasengeschwindigkeit des Lichts in diesem Medium liegt, wird das Medium angeregt, Photonen zu emittieren. Dieses Phänomen wird nach seinem Entdecker als *Cherenkov–Effekt* bezeichnet.

Der Winkel θ, unter dem die Cherenkov–Photonen von der Spur des das Medium durchquerenden geladenen Teilchens emittiert werden, ergibt sich zu

$$\cos\theta \quad = \quad \frac{c_0}{n \cdot v} \tag{2.10}$$

wobei c_0 für die Vakuumlichtgeschwindigkeit, n für den Brechungsindex des Mediums und v für die Geschwindigkeit des Teilchens stehen. Im Falle ausgedehnter Luftschauer wird unter Berücksichtigung der Vielfachstreuung der größte Teil der von den Sekundärteilchen hervorgerufenen Cherenkov–Photonen so innerhalb eines Winkels von bis zu ca. einem Grad gegenüber der ursprünglichen Einfallsrichtung des Primärteilchens emittiert. Dadurch leuchten die Cherenkov–Photonen bei senkrechtem Einfall des Primärteilchens einen Kreis auf dem Erdboden aus, der etwa einen Radius von 120 Metern aufweist.

Zur Veranschaulichung sind in Abbildung 2.3 simulierte Volumenhelligkeiten der in ausgedehnten Luftschauern erzeugten Cherenkov–Photonen jeweils für ein primäres Proton mit einer Energie von 6 TeV und ein primäres Photon einer Energie von 1 TeV gezeigt. Zu sehen ist, daß die laterale Verteilung der Cherenkov–Photonen von proton–induzierten Luftschauern i. a. deutlich breiter ausfällt als bei primären Photonen. Dieser Effekt erlaubt eine wirkungsvolle Trennung der hadronisch induzierten Ereignisse von denen, die durch primäre Photonen hervorgerufen werden.

Die mit einigen Nanosekunden Dauer sehr kurzen Cherenkov–Blitze sind also ein Indiz für auf die Erdatmosphäre treffende hochenergetische kosmische Teilchen. Aus der Zahl der auf dem Erdboden beobachteten Cherenkov–Photonen kann zudem auf die Energie des Primärteilchens geschlossen werden. Die Erdatmosphäre wird so zu einem aktiven Kalorimeter zum Nachweis kosmischer Teilchen, wobei die effektive Nachweisfläche für Cherenkov–Teleskope typischerweise 10^4 bis $10^5\,\mathrm{m}^2$ beträgt.

2.2.2 Abbildende Cherenkov–Teleskope

Cherenkov–Teleskope sind großflächige Spiegelteleskope, die zum Nachweis des in ausgedehnten Luftschauern erzeugten Cherenkov–Lichts verwendet werden. Zur Veranschaulichung sei bereits an dieser Stelle auf das in Abbildung 3.2 dargestellte H·E·S·S· Cherenkov–Teleskop verwiesen.

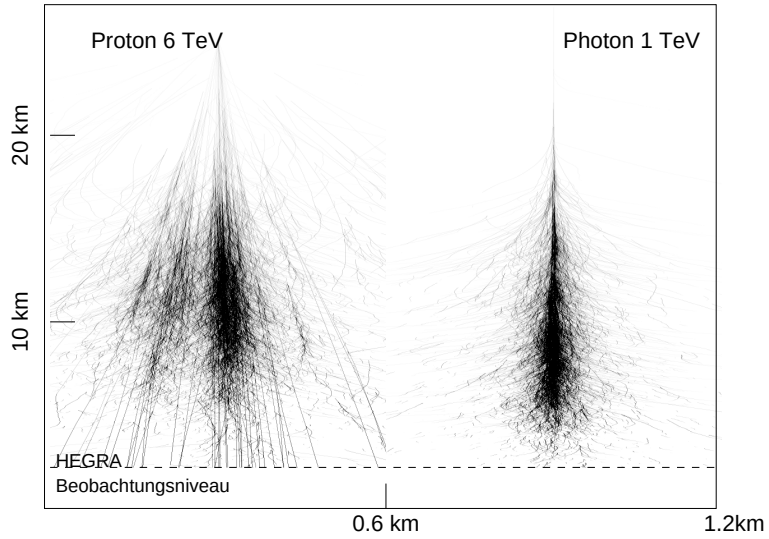


Abbildung 2.3: *Simulierte Volumenhelligkeit der in ausgedehnten Luftschauern erzeugten Cherenkov–Photonen. Links für ein primäres Proton einer Energie von 6 TeV, rechts für ein primäres Photon einer Energie von 1 TeV [Horn 00].*

Um möglichst große Lichtmengen sammeln zu können, sind Cherenkov–Teleskope mit großen Spiegelflächen ausgestattet, die aus Kostengründen in der Regel aus vielen kleinen Spiegelfacetten gebildet werden. Der Reflektor bildet das Cherenkov–Licht auf eine sich in der Fokalebene befindende Kamera ab. Da zum einen die Lichtmenge eines Luftschauers nur sehr klein ist und zum anderen der Cherenkov–Blitz nur eine Dauer von wenigen Nanosekunden aufweist, kommen zum Nachweis der Cherenkov–Photonen in der Regel hochempfindliche Photovervielfacher (engl. *photo–multiplier*, PMT) zum Einsatz.

Das Ziel bei der Vermessung der Cherenkov–Blitze ist eine möglichst genaue Rekonstruktion von Geometrie und Intensität der Schauer, welche letztlich die erwünschten Auskünfte über Energie, Einfallsrichtung und Spezies der Primärteilchen liefern. Dazu sind Abbilder der Luftschauer nötig, weshalb die Kameras von abbildenden Cherenkov–Teleskopen aus vielen matrixförmig angeordneten Photovervielfachern bestehen. Sie liefern so gerasterte zweidimensionale Abbilder der Schauer.

Das zweidimensionale Abbild eines Schauers vermag dessen räumlichen Verlauf jedoch nicht vollständig zu bestimmen, weshalb sich die Rekonstruktion der Schauergeometrie mit nur einem Teleskop recht schwierig gestaltet. Diesem Problem kann dadurch Rechnung getragen werden, mehrere Teleskope gleichzeitig einzusetzen, um stereoskopische Abbilder – d. h. zweidimensionale Abbilder aus unterschiedlichen Richtungen – des Schauers zu erhalten.

2.2.3 Stereoskopische Systeme abbildender Cherenkov–Teleskope

Pionierarbeit auf dem Gebiet der Stereoskopie hat die HEGRA–Kollaboration mit dem stereoskopischen System abbildender Cherenkov–Teleskope auf der Kanarischen Insel La Palma geleistet.

Das Ziel der Stereoskopie ist es, die Rekonstruktion der Schauergeometrie gegenüber Einzelteleskopen deutlich zu verbessern. Dazu werden die Objekte mit mehreren, geeignet plazierten Teleskopen simultan beobachtet, so daß Abbilder der Luftschauer aus verschiedenen Richtungen gewonnen werden. Kombiniert liefern diese anschließend eine wesentlich genauere räumliche Rekonstruktion des Schauers, was auch eine bessere Rekonstruktion der Energie des Primärteilchens zur Folge hat. In Abbildung 2.4 ist das Prinzip der Stereoskopie veranschaulicht.

Ein weiterer Vorteil, mehrere Teleskope zur Vermessung der Luftschauer einzusetzen, besteht in der wirkungsvolleren Unterdrückung von Untergrundereignissen. Dazu zählen Ereignisse, die von myon–induzierten Cherenkov–Photonen sowie Photonen des Nachthimmelhintergrundes (engl. *night–sky background*, NSB) verursacht werden. Durch den sogenannten *Multi–Teleskop–Trigger* können viele dieser Untergrundereignisse ausgeschlossen werden, da sie in der Mehrzahl nur in die Kamera eines Teleskops gelangen.

Ein einführender Übersichtsartikel über die Gamma–Astronomie unter besonderer Berücksichtigung des von der HEGRA–Kollaboration betriebenen Systems abbildender Cherenkov–Teleskope findet sich in [Kraw 00]. Zudem sei auf die Artikelserie [Völk 99a] und [Völk 99b] verwiesen, die ebenfalls eine leichtverständliche Einführung in dieses Thema bietet.

Eine detaillierte Darstellung des Prinzips der Stereoskopie sowie Berechnungen zur Leistungsfähigkeit stereoskopischer Systeme abbildender Cherenkov–Teleskope finden sich in [Ahar 97b] und [Ahar 97c]. Zudem sei hier auf die entsprechenden Betrachtungen für das HEGRA–System in [Kono 99] verwiesen.

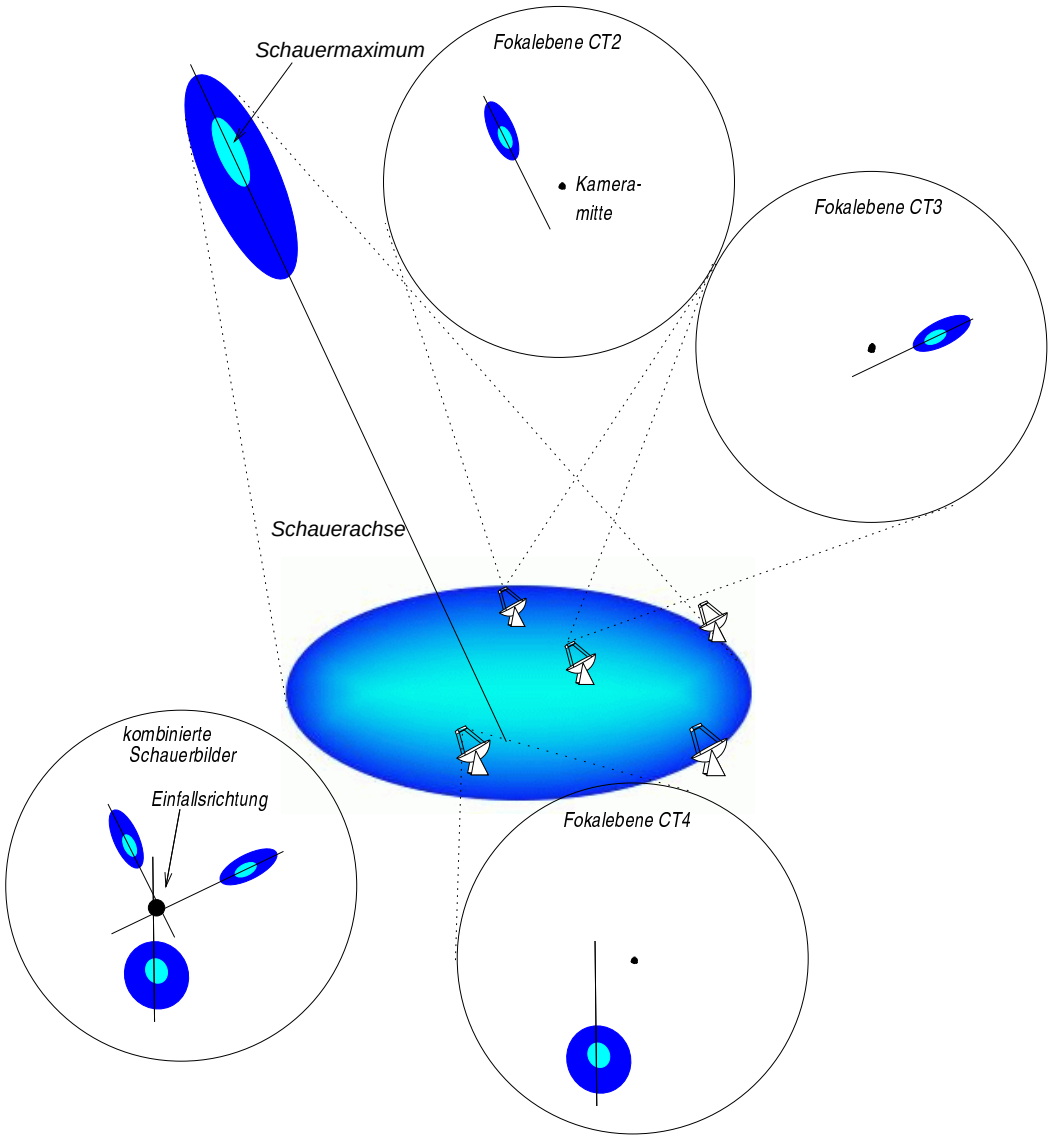


Abbildung 2.4: *Durch Überlagerung der Hauptachsen der aus unterschiedlichen Blickrichtungen gewonnenen Schauerbilder läßt sich die Schauerachse im Raum rekonstruieren. Dazu sind die Schauerbilder aller beteiligten Teleskope in ein gemeinsames Koordinatensystem zu transformieren [Horn 00].*

Kapitel 3

Das Victor H·E·S·S·–Projekt

Die H·E·S·S·–Kollaboration hat sich mit dem Ziel formiert, ein System abbildender Cherenkov–Teleskope einer neuen Generation für die Gamma–Astronomie im 100 GeV–Energiebereich zu errichten. Aufbauend auf den guten Erfahrungen mit dem System abbildender Cherenkov–Teleskope, das die HEGRA–Kollaboration auf der Kanarischen Insel La Palma betreibt, ist nun dieses Nachfolgeobservatorium auf der Südhalbkugel geplant. Zur H·E·S·S·–Kollaboration gehören u. a. einige der bereits am HEGRA–Experiment beteiligten Mitglieder, darunter auch das II. Institut für Experimentalphysik der Universität Hamburg.

In einer ersten Stufe wird ein stereoskopisches System aus vier Cherenkov–Teleskopen aufgebaut, das in einer Ausbaustufe später auf bis zu sechzehn Cherenkov–Teleskope erweitert werden soll. Als Standort für die H·E·S·S·–Teleskope wurde die Gegend um den Gamsberg in Namibia gewählt, für die ausgezeichnete optische Bedingungen über viele Jahre dokumentiert sind. Das erste Teleskop des Systems wird zur Zeit im den Gamsberg umgebenden Khomas–Hochland auf einer Höhe von 1800 m errichtet. In Abbildung 3.1 ist eine Photomontage der ersten vier H·E·S·S· Cherenkov–Teleskope im Khomas–Hochland von Namibia zu sehen.

In den folgenden Kapiteln soll auf die physikalischen Ziele, die Leistungsfähigkeit des Systems sowie den Aufbau der Teleskope eingegangen werden. Die Darstellungen beschränken sich dabei jeweils auf eine Auswahl an Fragestellungen; weitergehende Informationen zu diesen Themen finden sich z. B. in [Ahar 97a] und [HESS 00].

3.1 Physikalische Ziele

In Kapitel 2.1.3 wurde bereits erwähnt, daß der Ursprung der geladenen kosmischen Strahlung bislang nicht vollständig verstanden ist. Da die Beschleunigung geladener Teilchen jedoch in der Regel von der Erzeugung sehr hochenergetischer Photonen begleitet wird, können Cherenkov–Teleskope einen sehr wichtigen Beitrag zur Klärung leisten.

Ein weiterer Schwerpunkt wird die genauere Untersuchung der bereits etablierten

Abbildung 3.1: *Photomontage der ersten vier* H·E·S·S· *Cherenkov–Teleskope im Khomas–Hochland von Namibia [HESS 00].*

Quellen sehr hochenergetischer Photonen sein. Dazu gehören insbesondere die in Kapitel 2.1.3 vorgestellten Überreste von Supernovaexplosionen und aktive Galaxienkerne, die mit den H·E·S·S·–Teleskopen erstmalig räumlich aufgelöst werden sollen. Zudem soll der Versuch unternommen werden, die erwartete gepulste Komponente im hochenergetischen Gamma–Bereich eines Pulsars (PSR B1706-44) nachweisen zu können [Jage 00].

Einige als Quellen kosmischer Strahlung vorhergesagte Objektklassen konnten bisher nicht als solche nachgewiesen werden. Dazu zählen ausgedehnte Molekülwolken, Starburst–Galaxien und Galaxienhaufen. Das H·E·S·S·-Experiment wird sich daher mit der systematischen Untersuchung dieser Objektklassen befassen, wobei die Ergebnisse einen echten Test der Modellvorstellungen zulassen sollten.

Die H·E·S·S·–Teleskope eignen sich weiter zum Studium der intergalaktischen Hintergrundstrahlungsfelder. Die von weit entfernten Quellen – wie z. B. Blazaren – emittierten sehr hochenergetischen Photonen werden durch Wechselwirkung mit intergalaktischen Photonfeldern energieabhängig absorbiert, weshalb die Energiespektren dieser Quellen einen indirekten Zugang zu den intergalaktischen Photondichten liefern können.

Bei bereits bekannten Photondichten können die Messungen alternativ zur Verbesserung der Entfernungsbestimmung der Quellen herangezogen werden.

Abschließend sei noch die sogenannte Dunkle Materie angeführt. Nach heutiger Erkenntnis reicht die sichtbare Materie im Universum bei weitem nicht aus, dessen Dynamik erklären zu können. Es wird daher intensiv nach einer unsichtbaren Komponente der Materie geforscht. Nach supersymmetrischen Modellen könnten sogenannte Neutralinos einen Beitrag zur Dunklen Materie leisten, für die monoenergetische Gamma–Linien im Bereich um 100 GeV in Folge ihrer Anihilation vorhergesagt werden. Neutralinos werden gehäuft im Zentrum unserer Galaxie erwartet, weshalb sich zu ihrem Studium ein Standort auf der Südhalbkugel aufgrund der guten Sichtbarkeit des galaktischen Zentrums besonders eignet.

Für einen detaillierteren Überblick über die physikalischen Fragestellungen sei auf Anhang A von [Ahar 97a] verwiesen.

3.2 Leistungsfähigkeit des H·E·S·S·–Systems

Die erwarteten Eigenschaften des H·E·S·S·–Systems wurden durch detaillierte Monte–Carlo–Studien gewonnen, wobei sich die im folgenden beschriebenen Merkmale auf die erste Stufe mit vier Teleskopen beziehen.

Vornehmes Ziel bei der Auslegung der H·E·S·S·–Teleskope war die Senkung der Energieschwelle gegenüber bestehenden Cherenkov–Teleskopen. Das satellitengestützte EGRET–Experiment[1] konnte im Laufe seines Betriebs einen Katalog von 271 galaktischen und extragalaktischen Objekten zusammenstellen, die γ–Photonen mit Energien zwischen 100 MeV und einigen 10 GeV emittieren [Hart 99]. Davon sind lediglich sieben in dem bestehenden Cherenkov–Teleskopen zugänglichen Energiebereich sichtbar, weshalb der ihnen bisher nicht zugängliche Energiebereich von besonderem Interesse ist. Gegenüber den Systemteleskopen des HEGRA–Experimentes mit einer Energieschwelle von 500 GeV wird mit dem H·E·S·S·–System eine untere Energieschwelle von ca. 100 GeV erreicht, wobei ein bloßer Nachweis ohne spektroskopische Analysemöglichkeit ab etwa 40 GeV möglich sein soll.

Die Richtungsauflösung des H·E·S·S·–Systems wird besser als 0,1 Grad oberhalb von 100 GeV sein, womit es das in diesem Sinne beste Nachweisinstrument der Gamma–Astronomie wird. Der Schwerpunkt der Verteilung kann dabei auf wenige Bogensekunden genau bestimmt werden, so daß eine differenzierte Eingrenzung der Emissionsregionen von z. B. Pulsarnebeln möglich ist.

Bei der Rekonstruktion der Energie des primären Photons wird eine relative Genauigkeit von besser als 20 Prozent erreicht, was ein detailliertes Studium der Quellspektren erlaubt. Durch genaue Bestimmung der Abbruchenergie in den Spektren weit entfernter Quellen kann zudem auf die Beschaffenheit intergalaktischer Photonfelder geschlossen werden.

[1] Akronym für *Energetic Gamma Ray Experiment Telescope*

Die Sensitivität des H·E·S·S·–Systems wird im Vergleich zum HEGRA Teleskop–System um eine Größenordnung verbessert. Dies ermöglicht zum einen den Nachweis von Quellen, deren Flüsse bislang zu niedrig für einen gesicherten Nachweis waren. Zum anderen kann so die Photon–Statistik bereits bekannter Quellen deutlich erhöht werden.

3.3 Aufbau der H·E·S·S· Cherenkov–Teleskope

Abbildung 3.2 zeigt die Modellzeichnung eines H·E·S·S· Cherenkov–Teleskops.

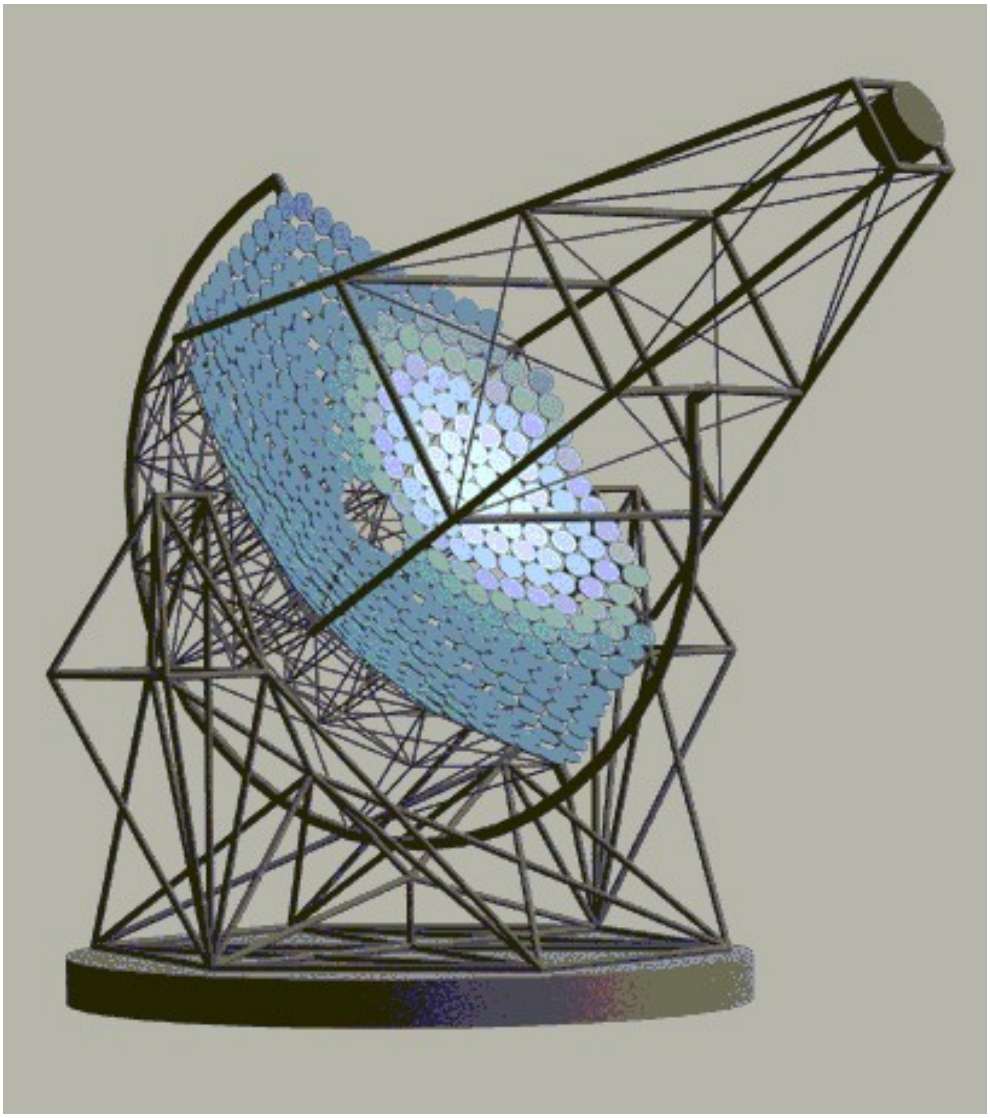


Abbildung 3.2: *Modellzeichnung eines* H·E·S·S· *Cherenkov–Teleskops [HESS 00].*

Der große Reflektor kann mit zwei Antriebsachsen auf die zu beobachtende Himmelsregion geführt werden. Dabei kommt eine azimutale Montierung zum Einsatz, bei der die eine Achse vertikal, die andere horizontal ausgerichtet ist. Die Nachführung azimutal montierter Teleskope ist recht aufwendig, da beide Achsen aufgrund der Erddrehung synchron zu verstellen sind. Zudem ist zu beachten, daß sich das Gesichtsfeld dabei (scheinbar) in der Kamera dreht.

Bei äquatorial montierten Teleskopen, bei denen die sogenannte Stundenachse parallel zur Erdachse eingerichtet ist, kann die Nachführung dagegen durch alleiniges Verstellen dieser Achse erfolgen. Die Ansprüche an die mechanische Realisierung sind bei äquatorial montierten Teleskopen jedoch ungleich höher, weshalb bei großen Teleskopen aus Kostengründen oft auf eine azimutale Montierung zurückgegriffen wird.

Um kostengünstig die große Spiegelfläche von 108 m^2 realisieren zu können, wird der Reflektor aus vielen einzelnen kreisförmigen Spiegelfacetten mit sphärischem Schliff gebildet, die jeweils einen Durchmesser von 60 cm aufweisen. Sie sind nach dem sogenannten Davies–Cotton–Prinzip [Davi 57] auf einer Kugelschale mit einer Brennweite von 15 Metern angeordnet, in deren Fokus sich die Kamera aus Photovervielfachern befindet. Die Vorteile dieser Auslegung sind zum einen, daß alle Spiegelfacetten so die gleiche Brennweite aufweisen, was eine kostengünstige Serienfertigung ermöglicht. Zum anderen bietet dieses Verfahren eine im Vergleich zu Parabolspiegeln bessere Abbildungsqualität bei achsenfernen Strahlen.

Sphärische Hohlspiegel weisen jedoch auch Nachteile auf. Achsenparallel einfallendes Licht wird von sphärisch geschliffenen Facetten nicht punktförmig in die Fokalebene abgebildet. Dieser mit sphährischer Aberration bezeichnete Effekt ist umso größer, je weiter die entsprechende Facette vom Zentrum des Reflektors entfernt ist. Bei der Auslegung der H·E·S·S·–Teleskope wurde deshalb darauf geachtet, diesen Effekt klein im Vergleich zum Auflösungsvermögen der Kamera zu halten.

Ein weiterer Nachteil sphährischer Spiegel ist die unterschiedliche Ankunftszeitverteilung der in die Kamera gelangenden Photonen. Aufgrund der Geometrie haben an zentraler Stelle im Spiegel reflektierte Photonen einen längeren Weg zur Kamera zurückzulegen als an den Spiegelrändern eintreffende Photonen. Im vorliegenden Fall ist diese Zeitdispersion jedoch klein im Vergleich zur Ansprechzeit der Photodetektoren.

Die Teleskopkamera in der Fokalebene hat ein Gesichtsfeld von fünf Grad. Sie wird aus 960 Pixeln gebildet, die jeweils 0,16 Grad des Gesichtsfeldes erfassen. Zum Einsatz kommen dabei hochempfindliche Photovervielfacher, die sich zusammen mit der gesamten Betriebs– und Ausleseelektronik sowie der Steuerungseinheit in einem Gehäuse befinden. So kann die Zahl der nötigen Kabelverbindungen zwischen der Kamera im Teleskop und der Datenerfassungszentrale klein gehalten werden.

3.3.1 Motorisch justierbare Spiegelfacetten

Der Reflektor eines jeden H·E·S·S·–Teleskops wird aus fast 400 Spiegelfacetten gebildet. Schon allein diese große Zahl läßt aufgrund des beträchtlichen Aufwandes und der vergleichsweise schlechten Zugänglichkeit der Spiegelfacetten eine manuelle Justierung nahezu unmöglich erscheinen. Aus diesem Grund ist eine motorische Verstellmöglichkeit

der Spiegelfacetten vorgesehen.

Dazu wird jede Spiegelfacette an drei Punkten, die sich in den Ecken eines gleichseitigen Dreiecks befinden, auf einer Grundplatte befestigt. Zwei der Befestigungen sind in der Höhe über der Grundplatte motorisch verstellbar – sie bilden jeweils einen sogenannten Aktuator –, so daß durch diese zwei Freiheitsgrade die gewünschte Richtung in gewissen Grenzen einstellbar ist[2]. Abbildung 3.3 zeigt die Modellzeichnung einer mit Aktuatormotoren bestückten Spiegelfacettenmechanik. Die Aktuatormechanik ist

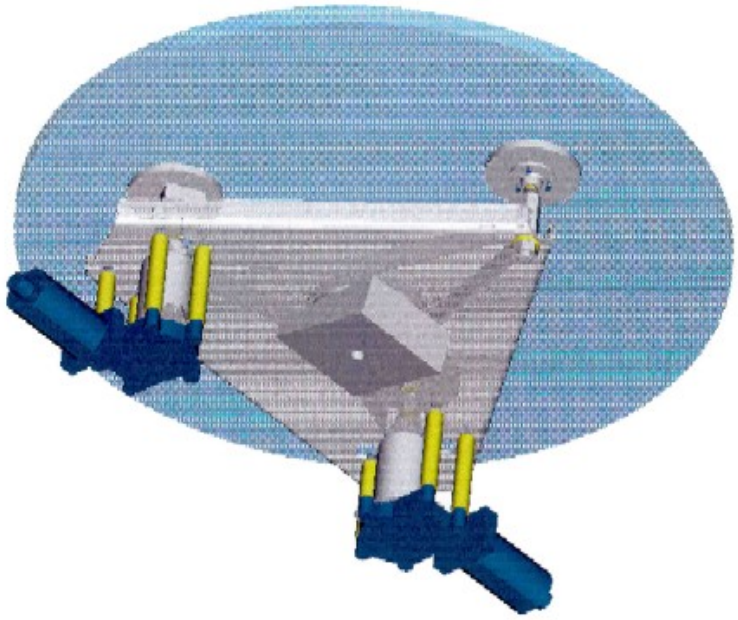


Abbildung 3.3: *Modellzeichnung einer mit Aktuatormotoren bestückten Spiegelfacettenmechanik [HESS 00].*

so ausgelegt, daß die Spiegelfacetten um ±1,5 Grad verstellbar sind.

Die Motoren zum Antrieb der Aktuatoren sind kostengünstige Getriebemotoren, die von der Firma Bosch in großen Stückzahlen für den Automobilbau gefertigt werden. Zwar verfügen sie nicht – wie Schrittmotoren – über eine selbständige Positioniereinheit; sie sind jedoch mit einem Drehgeber ausgestattet, der eine Kontrolle des Drehwinkels erlaubt. Die dabei erzielbare Justiergenauigkeit beträgt 0,001 Grad.

Sowohl von der Hamburger als auch von der Heidelberger Gruppe wurden Entwürfe für die Machanik zur Justierung der Spiegelfacetten erarbeitet. Es war u. a. Gegenstand der vorliegenden Arbeit, Prototypen beider Entwürfe im Zusammenhang mit der Elektronik zur Ansteuerung der Aktuatormotoren (s. u.) zu testen. Die Ergebnisse werden ausführlich in den Kapiteln 5.3 und 5.4 diskutiert.

Die Motoren sämtlicher Spiegelfacetten eines Teleskops sind jeweils einzeln von einer

[2]Eine *Fokussierung* der Spiegel ist mit dieser Methode nicht möglich.

zentralen Einheit sowohl per Hand als auch softwaregesteuert verstellbar. Zum Anschluß der Aktuatormotoren an die Steuerungselektronik sind zwölf Kabel pro Teleskop vorgesehen, die jeweils die Spiegelfacetten eines halben Segmentes des Spiegelträgers elektrisch versorgen. Der geplante Verlauf der Kabel zur Versorgung der Aktuatormotoren durch eines der Segmente ist in Abbildung 3.4 dargestellt.

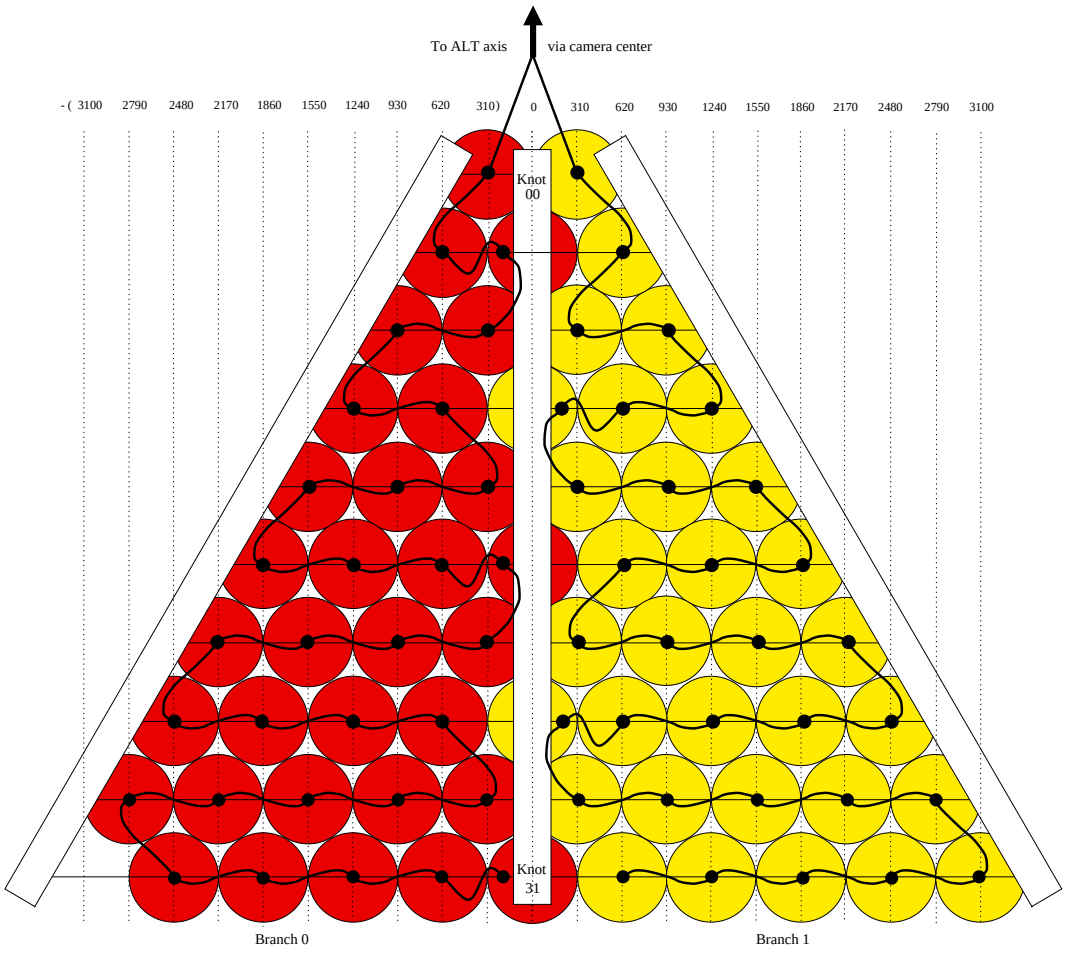


Abbildung 3.4: *Geplanter Verlauf der Kabel durch ein Spiegelsegment zur Versorgung der Aktuatormotoren. Sechs solcher Segmente bilden jeweils den Spiegel eines Teleskops [Tluc 01].*

Für die Entwicklung der Elektronik zur Ansteuerung der Aktuatormotoren zeichnet die Hamburger Gruppe verantwortlich. In Kapitel 4.1.1 werden die Komponenten der

Hardware einer eingehenderen Betrachtung unterzogen, wobei die für die vorliegende Arbeit wichtigen Besonderheiten detalliert zur Sprache kommen. Zudem wurden alle Komponenten der Elektronik verschiedenen Tests unterzogen, deren Ergebnisse in Kapitel 5.2 vorgestellt werden.

Kapitel 4

Das System zur automatisierten Justierung der Spiegelfacetten der H·E·S·S· Cherenkov–Teleskope (MACS)

Das grundsätzliche Verfahren zur Justierung der Spiegelfacetten der H·E·S·S·–Teleskope ist z. B. in [Hofm 98], [Kato 99] und [Gill 99] beschrieben.

Die hochempfindliche Pixelkamera aus Photovervielfachern wird tagsüber und in Mondnächten mittels eines Deckels vor Lichteinfall geschützt. Geeignet beschaffen bietet dieser Kameradeckel einen Diffusor in der Fokalebene[1], der die Abbilder aller Spiegelfacetten zugänglich macht. Ein menschlicher Operator wäre damit bereits prinzipiell in der Lage, anhand dieser Abbilder die Spiegelfacetten mittels der motorischen Handsteuerung auszurichten.

Um eine weitestgehend automatisierte Justierung zu realisieren, werden die Abbilder mit einer CCD–Kamera[2] aufgenommen und in digitaler Form einer Steuereinheit übermittelt. Die Steuereinheit analysiert die von der CCD–Kamera aufgenommenen Bilder und entscheidet, welcher Aktuatormotor welcher Spiegelfacette wie zu verfahren ist, damit letztlich alle Facetten im Rahmen der Vorgaben ausgerichtet sind.

Als Lichtquelle für die Abbilder der Spiegelfacetten sollen Sterne verwendet werden. In [Gill 99] wurde gezeigt, daß helle Sterne ab etwa der dritten Magnitude eine ausreichende Lichtmenge liefern, um den Lichtfleck einer einzelnen Spiegelfacette vor dem Hintergrund der restlichen Abbilder erfassen zu können. Hierbei ist zu beachten, daß während der Justierung die Effekte noch nicht optimaler Nachführung und des

[1] Der Kameradeckel wurde bisher nicht vollständig spezifiziert. Sollte er nicht so gestaltet sein, daß die Spiegelfacetten bei Überlagerung aller Abbilder im Zentrum ausgerichtet sind, ist dies zu kompensieren.

[2] Engl. *charge coupled device*; eine Technologie zur Aufnahme von Bildern, bei der ein aus matrixförmig angeordneten Lichtsensoren (Pixel) bestehender Chip verwendet wird, dessen Bildinformationen digitalisiert zur Verfügung gestellt werden.

Durchbiegens der Kameramasten[3] zu berücksichtigen bzw. zu kompensieren sind.

Das oben beschriebene Verfahren wurde mit Prototypen der Spiegelfacettenmechanik erfolgreich an einem der HEGRA–Teleskope getestet. Eine detaillierte Darstellung des Prinzips, Untersuchungen zur Justiergenauigkeit sowie eine Diskussion der Testergebnisse finden sich in [Gill 99].

Als Benennung des Systems zur Justierung der Spiegelfacetten der H·E·S·S· Cherenkov–Teleskope wurde H·E·S·S· *Mirror Alignment Control System* (MACS) gewählt. Dieses aus Hard– und Softwarekomponenten bestehende Gesamtsystem zur Justierung der Spiegelfacetten soll nun eingehender vorgestellt werden.

4.1 Hardwarekomponenten und ihr Aufbau

Die folgenden Kapitel widmen sich einer detaillierteren Beschreibung aller zur Justierung der Spiegelfacetten zum Einsatz kommenden Hardwarekomponenten.

Der größte Teil der Hardware, den es lediglich durch industrielle Standardkomponenten zu vervollständigen galt, wurde von der Universität Hamburg entwickelt.

4.1.1 Die Elektronik zur direkten Ansteuerung der Aktuatormotoren

Die Elektronik zur Ansteuerung der Aktuatormotoren ist eine von Mitarbeitern der Abteilung *Technische Entwicklung und Betrieb* (TEB) der Universität Hamburg speziell für diesen Zweck konzipierte Eigenentwicklung. Sie ist dafür ausgelegt, nur jeweils einen Motor zur Zeit zu betreiben, da mit mehrmonatigen Abständen zwischen einzelnen Gesamtjustierungen eines Teleskops gerechnet wird. So wird der Aufwand sowohl der Hardware als auch der Software gering gehalten.

Abbildung 4.1 zeigt den schematischen Aufbau der Elektronik zur Ansteuerung der Aktuatormotoren. Zum besseren Verständnis folgt die Terminologie im nachfolgenden Text den Benennungen in der Dokumentation der Hardware [TEB 00].

An einem sogenannten *Branch Cable* (engl. für Kabelzweig) sind die Aktuatormotoren von jeweils bis zu zweiunddreißig Spiegelfacetten angeschlossen; ein *Branch Cable* kann somit vierundsechzig Motoren mit allen nötigen Spannungs–, Steuer– und Signalleitungen versorgen. Jeder *Branch* wird an einem Modul, genannt *Branch Driver* (engl. für Treibereinheit), betrieben, an dem die Steuer– und Signalleitungen herausgeführt sind. Eine detaillierte Darstellung eines mit Spiegelfacetten bestückten *Branch Cables* findet sich in Abbildung 4.2.

Zentrale Einheit der Elektronik ist die *Branch Control Unit* (engl. für Kontrolleinheit). Sie setzt die Befehle zur Kommandierung der Aktuatormotoren um und steuert sämtliche *Branch Driver* eines Teleskopes an. Sie ist mit einer Frontplatte versehen, auf der Steuerelemente zur manuellen Kontrolle der Motoren angebracht sind.

[3] Engl. *bending*; die Masten bzw. Träger, die die Kamera (Masse ca. 600 kg) in 15 Metern Entfernung vom Spiegelträger halten, biegen sich (im wesentlichen) zenitwinkelabhängig aufgrund des ausgeübten Drehmoments durch.

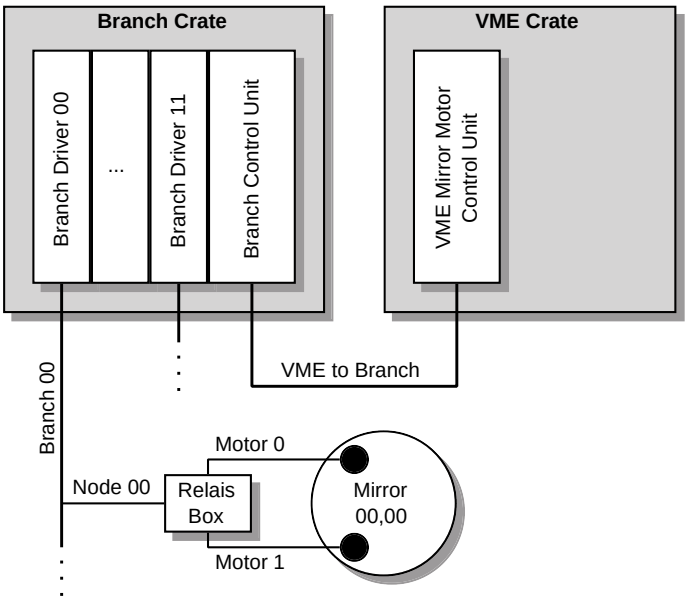


Abbildung 4.1: *Schematische Darstellung der Hardwarekomponenten zur Ansteuerung der Aktuatormotoren.*

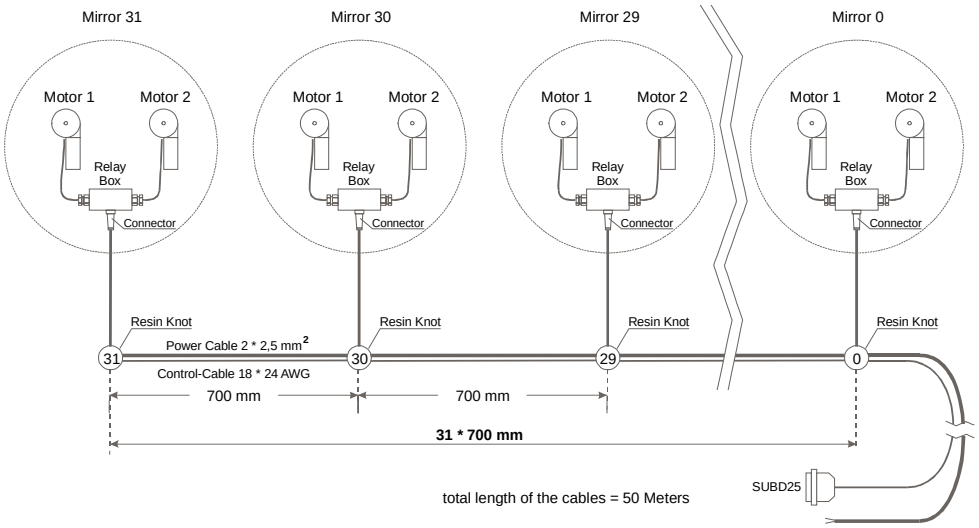


Abbildung 4.2: *Detaillierte Darstellung eines mit Spiegelfacetten bestückten* Branch Cables *[TEB 00].*

Um sämtliche Steuerfunktionen der Einheit zur automatisierten Justierung zur Verfügung zu stellen, ist die *Branch Control Unit* zudem über Steuerleitungen mit einem

VME–Board verbunden. Dieses stellt der Steuereinheit (VME–CPU) für die automatisierte Justierung sämtliche Steuerfunktionen der *Branch Control Unit* über ein Bussystem zur Verfügung. Hierbei kommt eine mit *VMEbus* [VME 93] bezeichnete Technologie zum Einsatz, die eine in Industrie und Forschung häufig genutzte Standardlösung zur Kommunikation unterschiedlicher Hardwarekomponenten darstellt.

Einige Merkmale der TEB–Elektronik zur Ansteuerung der Aktuatormotoren sind für später folgende Betrachtungen von besonderer Bedeutung. Auf sie soll deshalb detaillierter eingegangen werden.

4.1.1.1 Positionszähler

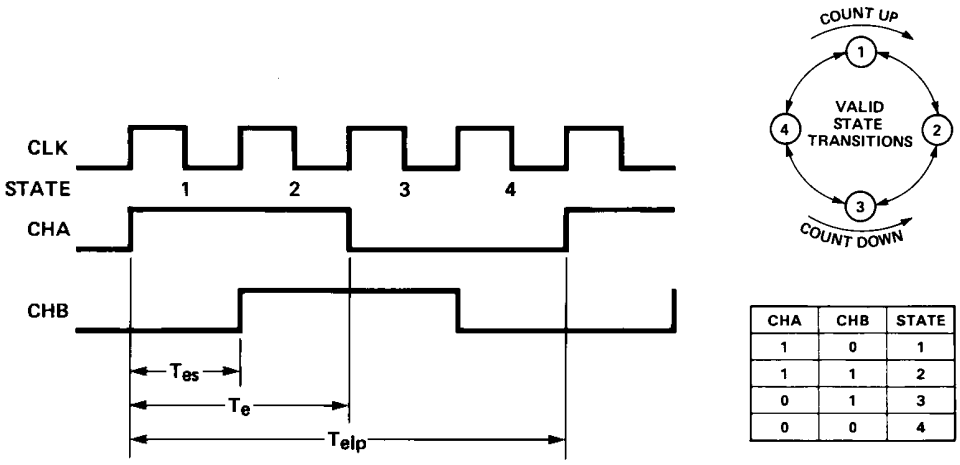
Aus Kostengründen kommen zur Justierung der Spiegelfacetten keine Schrittmotoren zum Einsatz, welche gewünschte Positionen selbständig anfahren können. Die hier verwendeten Gleichstrommotoren mit Getriebe sind dazu nicht in der Lage, sie liefern jedoch Informationen über Betrag und Richtung der Drehung, die die *Branch Control Unit* zur Implementierung eines Positionszählers nutzt.

Dazu befinden sich auf der Motorwelle der Motoren zwei halbringförmige Permanentmagnete mit entgegengesetzter Feldausrichtung, die von zwei Hallsensoren im Winkel von 90 Grad zueinander abgetastet werden. Zusammengefaßt ergeben die beiden Signale dieser Hallsensoren so vier mögliche Zustände (Winkelbereiche der Motorwelle von jeweils 90 Grad), wobei nur bestimmte Zustandsänderungen (Übergänge zwischen verschiedenen Winkelbereichen) vorkommen. Die Übergänge selbst werden dazu genutzt, Winkeländerungen in ganzen Vielfachen von 90 Grad zu bestimmen, während die Richtung der Änderung aus der Art des Übergangs ermittelt wird. Abbildung 4.3 zeigt den Verlauf beider Signale über eine volle Drehung der Motorwelle sowie die bei Motorbewegungen vorkommenden Zustandsänderungen. Die Motoren sind jeweils mit einem Getriebe mit einer Untersetzung von $^1/_{55}$ bestückt, so daß die Hallsensoren 220 Flankenwechsel pro Umdrehung der Antriebsschnecke liefern.

Um die Aktuatormotoren genau positionieren zu können, müssen die Signale der Hallsensoren korrekt übertragen und verarbeitet werden. Eine fehlerfreie Übertragung der Signale wird dabei durch die stets vorhandenen Störspannungen auf den Signalleitungen erschwert, die zu Mißinterpretationen von Flankenwechseln führen können.

Aus diesem Grund kommt zur Verarbeitung der Signale ein speziell für diesen Zweck ausgelegter Kontrollbaustein (Quadrature Decoder/Counter HCTL–2020 [Agil 99]) der Firma Agilent Technologies zum Einsatz. Er verfügt über ein mehrstufiges digitales Filtersystem, um Störsignale wirkungsvoll unterdrücken zu können. Die durch diese Filterstufe gelangenden Signale werden anschließend noch Konsistenzüberprüfungen unterzogen. Zum einen werden innerhalb einer gewissen Zeit auftretende Flankenwechsel gemeinsam betrachtet, um falsche Signale identifizieren zu können. Zum anderen wird überprüft, ob ein Flankenwechsel einer gültigen Zustandsänderung entspricht.

Der Mechanismus der Übertragung und Verarbeitung der Signale der Hallsensoren wurde ausführlich getestet, wobei die Ergebnisse in Kapitel 5.2.4 diskutiert werden.



CHA	CHB	STATE
1	0	1
1	1	2
0	1	3
0	0	4

Abbildung 4.3: *Zustandsdiagramm der Signale von den Hallsensoren der Aktuatormotoren [Agil 99]. Links ist der Verlauf der beiden Signale (*`CHA` *u.* `CHB`*) über eine volle Motordrehung (*T_{elp}*) abgebildet. Darüber sind die vier möglichen Zustände (*`STATE`*) durch Nummern unterschieden. Rechts sind die gültigen Zustandsübergänge (*`VALID STATE TRANSITIONS`*) veranschaulicht, aus denen die Richtungsinformation gewonnen wird.*

4.1.1.2 Motorgeschwindigkeiten

Die Aktuatormotoren sind Gleichspannungsmotoren, die zum Zweck der Justierung mit verschieden wählbaren Geschwindigkeiten fahrbar sein sollen. Die *Branch Control Unit* bietet deshalb die Möglichkeit, acht verschiedene Motorgeschwindigkeiten einzustellen.

Zur Realisierung unterschiedlicher Drehgeschwindigkeiten wird eine gepulste Gleichspannung verwendet. Dieses mit Pulsweitenmodulation bezeichnete Verfahren bedeutet ein wiederkehrendes kurzzeitiges Unterbrechen der Spannungsversorgung. Durch Wahl des sogenannten Puls–Pause–Verhältnisses, d. h. dem Verhältnis der Zeit mit voller Spannung zur spannungslosen Zeit, kann die Geschwindigkeit der Motoren variiert werden. Die Zykluszeit (Zeit für einen Puls und eine Pause) sollte dabei so klein gehalten werden, daß die Motoren keine ruckartigen Bewegungen ausführen.

4.1.1.3 Motorabschaltung

Die Spindeln der Aktuatoren haben einen begrenzten verfahrbaren Bereich, der im folgenden oft auch als absoluter Hub bezeichnet wird. Damit die Motoren und mechanischen Komponenten beim Fahren an die mit Anschlag bezeichneten Endbereiche nicht beschädigt bzw. unnötig belastet werden, ist die *Branch Control Unit* mit einer Motorabschaltung versehen.[4] Zudem soll mit ihr verhindert werden, daß die Spindeln

[4] Die Motoren verfügen über eine interne Abschaltautomatik, die jedoch für die hier gestellten Anforderungen zu spät greift.

derart weit in die Anschläge gefahren werden, daß sie aus diesen nicht mehr herausbewegt werden können. Ein solches Festfahren eines Aktuators hätte fatale Konsequenzen, denn die so unbrauchbar gewordene Spiegelfacette müßte ausgetauscht werden, was bei der Größe der Teleskope ein schwieriges Unterfangen darstellt.

Die Motorabschaltung wird durch Kontrollieren der Flankenwechsel der Hallsensoren bewerkstelligt. Bleiben beim Fahren eines Aktuatormotors (weitere) Signale der Hallsensoren für eine von der gewählten Geschwindigkeit abhängige Zeitspanne aus, wird dies als Indiz für ein baldiges Stehenbleiben des Motors gewertet und ihm die Betriebsspannung entzogen. Dabei ist jedoch zu vermeiden, daß die Motoren – z. B. aufgrund von Schwergängigkeit durch temperaturabhängige Gleitmittelviskosität – unkontrollierbar zwischen den Anschlägen stehenbleiben. Auch sollten die tatsächlichen Haltepositionen an den Anschlägen reproduzierbar innerhalb kleiner Bereiche liegen, damit ein Festfahren durch einmaliges Vermessen des verfahrbaren Bereiches dauerhaft vermieden werden kann.

Die aus der Motorabschaltung resultierenden Anschlagpositionen wurden auf ihre Reproduzierbarkeit hin untersucht. In Kapitel 5.3.1 werden die Ergebnisse für einige Hamburger Prototypen der Spiegelfacettenmechanik vorgestellt; in Kapitel 5.4.1 finden sich die Ergebnisse für einen Heidelberger Prototyp. Die daraus resultierenden absoluten Hübe der Aktuatoren werden jeweils in den Kapiteln 5.3.3 und 5.4.3 betrachtet.

4.1.1.4 Motorpositionierung

Da die verwendeten Motoren Positionen nicht selbständig anfahren können, enthält die Hardware einen Mechanismus, diese Funktionalität dennoch abbilden zu können.

Zusätzlich zu der Möglichkeit, Motoren an– bzw. abzuschalten bietet die *Branch Control Unit* dazu den sogenannten *Preset Mode* (engl. für Voreinstellungsmodus). Dieser erlaubt, den angewählten Motor eine bestimmbare Anzahl von Zählerimpulsen (Flankenwechseln der Hallsensoren) in die gewünschte Richtung zu fahren. Die *Branch Control Unit* sorgt in diesem Fall für eine zeitgerechte Abschaltung des Motors, um die Zielposition (d. h. die abzufahrenden Zählerimpulse) möglichst genau zu erreichen.

Allerdings ist dabei nicht gewährleistet, daß der Motor tatsächlich genau auf der Zielposition zum Stehen kommt, denn die Motoren laufen nach Abschaltung der Betriebsspannung noch etwas nach. Um dem entgegenzuwirken wird zwar die Geschwindigkeit kurz vor Erreichen der Zielposition auf einen geringeren Wert (frei wählbar) gesetzt, dies kann den Effekt jedoch nur teilweise kompensieren. Auch besteht immer die Möglichkeit, daß der Motor bereits vor Erreichen der Zielposition durch ausbleibende Flankenwechsel abgeschaltet wird (vgl. Kap. 4.1.1.3).

Die *Branch Control Unit* unternimmt für jede Fahrt im *Preset Mode* nur jeweils einen Versuch, den Motor die gewünschte Anzahl von Zählerimpulsen zu fahren. Durch Iteration der Anfahrtversuche kann bei einer bestehenden Abweichung von der Zielposition eine Verbesserung erreicht werden. Die dabei erzielbare Genauigkeit wurde getestet, wobei die Ergebnisse in Kapitel 5.2.5 vorgestellt werden.

4.1.2 Die VME–CPU zur Kommunikation mit der Elektronik

Eine Aufgabe der vorliegenden Arbeit bestand in der Auswahl einer geeigneten VME–CPU zur Kommunikation mit der *Branch Control Unit* über das *VME–Board*. Neben der tatsächlichen Beschaffenheit der Hardware ist dabei ebenso wichtig, unter welchen Betriebssystemen die VME–CPU betrieben werden kann. Generell kamen zwei Betriebssysteme für die VME–CPU in Frage, die die Kriterien hinsichtlich Stabilität, Flexibilität, Verbreitung und Robustheit erfüllen.

LynxOS ist ein kommerzielles Echtzeitbetriebssystem auf UNIX–Basis, das sich in der Industrie seit vielen Jahren etablieren konnte. Es ist für eine unüberschaubar große Auswahl an VME–CPUs mit unterschiedlichen Prozessoren (Motorola 68k, Motorola PPC, Intel ix86 etc.) verfügbar, was sich allerdings in einem recht hohen Preis niederschlägt.

Linux ist eine freie UNIX–Implementation, die vollständig im Quelltext zur Verfügung steht. Es wird erst seit kurzer Zeit in industriellen Projekten verwendet, kann sich dort aber einer stetig wachsenden Beliebtheit erfreuen. Für Linux gibt es derzeit nur einen Treiber (*universe* [Hann 00]), der die Kommunikation mit anderen VME–Komponenten über den VMEbus zur Verfügung stellt. Dieser unterstützt nur wenige VME–CPUs auf Intel ix86–Basis. Nur zwei Hersteller solcher CPUs haben einen Vertrieb in Deutschland, wovon einer die Firma ELTEC Elektronik ist, die zudem zu den günstigsten Anbietern von VME–CPUs gehört.

Wichtige Kriterien für die Beschaffenheit der VME–CPU waren Anschlußmöglichkeiten für gängige EIDE–Festplatten, Disketten– oder CDROM–Laufwerk, Tastatur und Monitor. Zur Kommunikation mit anderen Rechnern sollte sie zudem über einen Anschluß für den Vernetzungsstandard 100MBit–Ethernet (100Base-T) verfügen.

4.1.3 Die CCD–Kamera zur optischen Rückkoppelung

Die Auswahl der CCD–Kamera gestaltete sich überraschend schwierig. Zwar wurden eine ganze Reihe unterschiedlicher Modelle auf ihre Tauglichkeit für die vorliegende Aufgabe hin untersucht, jedoch erfüllte letztlich nur eine die Minimalanforderungen.

Die eingehender untersuchten Kameras waren:

1. Meade Pictor 416XT,
2. SBIG ST-7(E) und
3. Apogee KX1.

Zu allen finden sich in [Hofm 98] nähere Informationen samt einem Vergleich der jeweiligen technischen Ausführung.

Bei allen Kameras war die Problematik in der Verfügbarkeit von Hardwaretreibern für die in Frage kommenden Betriebssysteme begründet. Nur für die SBIG ST-7(E) liegt ein Treiber [Ashe 99] für eines der in Frage kommenden Betriebssysteme – in diesem Fall Linux – vor.

Zwar wurde bei der Meade Pixtor 416XT eine Offenlegung der Kommunikationsprotokolle zwischen Rechner und Kamera in Aussicht gestellt, eine feste Zusage konnte jedoch nicht erhalten werden. Zudem hätte dies die Eigenentwicklung eines Hardwaretreibers erfordert, was sich in der Regel recht aufwendig gestaltet.

Doch auch bei der SBIG ST-7(E) ist die Treibersituation nicht uneingeschränkt zufriedenstellend. Zum einen ist der Treiber nicht im Quelltext vorhanden und wird auch nur von dritter Seite zur Verfügung gestellt, zum anderen ist das Zeitverhalten der Signale auf der Schnittstelle zwischen Computer und Kamera so kritisch, daß – für UNIX–Systeme völlig untypisch und nicht unproblematisch – während der Bildauslese sämtliche Interrupts[5] des Rechners unterbunden werden.

Die Situation hat sich während des Verfassens dieser Arbeit geändert. Die Veröffentlichung von Linux–Treibern für die CCD–Kameras der Firma Apogee steht in Kürze bevor, weshalb diese wieder in die engere Wahl genommen worden sind. Dieser jüngsten Entwicklung konnte in der Darlegung allerdings nicht mehr Rechnung getragen werden.

4.1.4 Auswahl der Hardwarekomponenten

Mangels Alternativen fiel die Wahl der CCD–Kamera auf das Modell ST-7(E) der Firma SBIG. Diese erfordert aufgrund der Treibersituation eine VME–CPU auf Intel ix86–Basis, die unter dem Betriebssystem Linux läuft. Die Wahl fiel hier auf das sehr günstige Modell EUROCOM 138 [Elte 99] der Firma ELTEC Elektronik, das von dem freien und im Quelltext verfügbaren Linux–Treiber *universe* [Hann 00] zur Kommunikation mit anderen VME–Komponenten über den VMEbus unterstützt wird.

Abbildung 4.4 gibt einen Überblick über alle Hardwarekomponenten, die zur Justierung der Spiegelfacetten zum Einsatz kommen. Zur Ansteuerung der vom TEB entwickelten Komponenten kommuniziert die VME–CPU über den VMEbus mit dem *VME–Board*, das die Befehle an die *Branch Control Unit* weiterreicht. Die optische Verfolgung der Abbilder der Spiegelfacetten in der Fokalebene erfolgt mittels einer per Parallelport (IEEE 1284) an die VME–CPU angeschlossenen CCD–Kamera.[6]

Das Gesamtsystem ist schließlich über Ethernet–Netzwerkleitungen mit allen weiteren Komponenten des H·E·S·S·–Experiments verbunden und kann somit von zentraler Stelle aus gesteuert werden.

[5] Innerhalb von Computersystemen werden alle eintreffenden Ereignisse (Tastendruck, Mausbewegung, Netzwerkaktivität etc.) mittels sogenannter *Interrupts* den für ihre Verarbeitung zuständigen Komponenten signalisiert.

[6] Sollte die Länge der Parallelportverbindung den zulässigen Höchstwert übersteigen, müßte eine Signalaufbereitungseinheit zwischengeschaltet werden. Eine andere Möglichkeit wäre, die CCD–Kamera an einem eigenen kleinen Rechner im Teleskop zu betreiben.

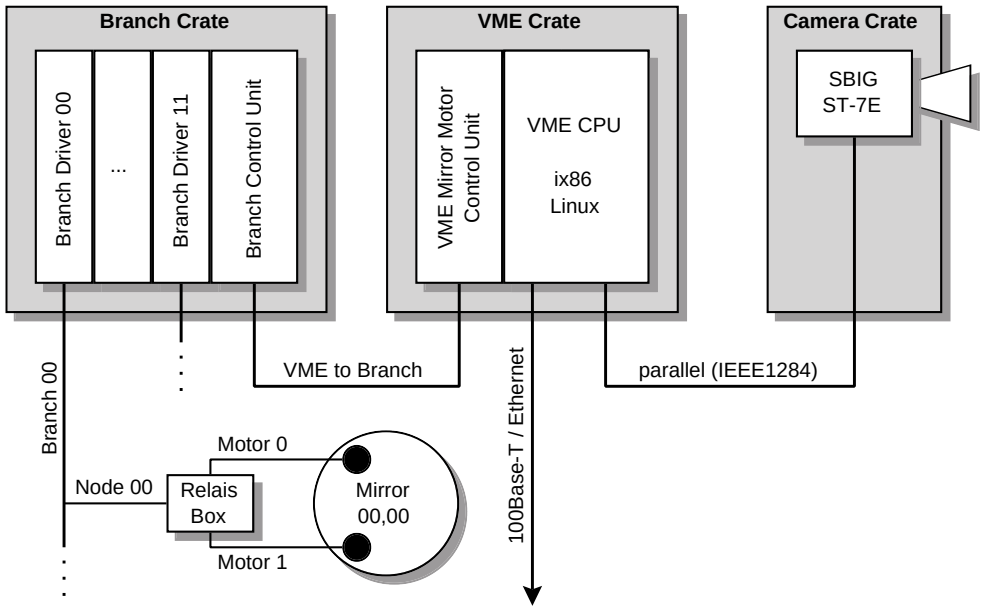


Abbildung 4.4: *Schematische Darstellung aller Hardwarekomponenten zur Ansteuerung der Aktuatormotoren.*

4.2 Grundstruktur der Software zur Justierung der Spiegelfacetten

Ein Programm vom hier zur Diskussion stehenden Umfang sollte zuerst auf unterschiedlichen Ebenen konzipiert und die Softwarekomponenten vor der eigentlichen Programmierung modelliert werden. Nur eine möglichst detaillierte Konzeption einer Software kann eine saubere Implementation und damit übersichtliche Strukturen garantieren. Diese Vorgehensweise gewährleistet zudem, daß ein Programm nach seiner Fertigstellung leicht gewartet und bei Bedarf komfortabel erweitert bzw. an neue Bedingungen angepaßt werden kann.

Der für die vorliegende Aufgabe gewählte dreistufige Ansatz zur Erstellung eines Konzeptes sei hier kurz skizziert:

1. Basis– bzw. Grundstruktur:
   Die Basisstruktur wird u. a. durch folgende Fragestellungen festgelegt:

   - In welcher Programmiersprache soll das Programm implementiert werden,
   - welche Technologien (Bibliotheken, Netzwerk, Benutzerschnittstelle, Datenbank etc.) sollen dabei zum Einsatz kommen,
   - welche Werkzeuge (Compiler, Debugger, Lexical Analyzer etc.) sollen genutzt werden,

- kann bzw. soll das Programm in mehrere Teilprogramme unterteilt werden, die u. U. auf verschiedenen Rechnern ablaufen und
- auf welche Systemplattformen soll die Software portiert werden können?

2. Innere Struktur bzw. Modellierung der Komponenten:
   Typische Fragen, die die innere Struktur einer Software betreffen, sind:
   - In welche Module bzw. Klassen kann das Programm sinnvoll unterteilt werden,
   - wie gestalten sich die Abhängigkeiten zwischen den Modulen bzw. Klassen,
   - welche Funktionalitäten sollen die einzelnen Module bzw. Klassen abdecken,
   - wie können diese Funktionalitäten möglichst vollständig gekapselt werden und
   - wie gestaltet sich der Fluß der Daten durch das Programm?
3. Algorithmen zur Darstellung der Funktionalitäten:
   In der Regel sind nicht alle Aufgaben, die eine Software erfüllen soll, trivialer Natur. Für diese Funktionalitäten müssen Algorithmen entwickelt werden. Einige Fragestellungen hierzu:
   - Welche gewünschten Funktionalitäten sind nicht trivial (wo bedarf es der Entwicklung einer Rechenvorschrift),
   - für welche Fälle soll der Algorithmus ausgelegt sein,
   - welche Fälle kann bzw. soll der Algorithmus nicht abdecken und wie werden diese erkannt und behandelt und
   - welche Informationen müssen dem Algorithmus zur Verfügung gestellt werden, damit das gewünschte Ergebnis erzielt werden kann?

4.2.1 Anforderungen an die Software

Die Anforderungen an die Grundstruktur der Software haben die Zahl der möglichen Varianten bereits im Vorfeld eingeschränkt.

Die Vorgaben für die Entwicklungsumgebung waren:

- Die Verwendung von Linux als Betriebssystem
  (Stabilität, Flexibilität, Verbreitung, Robustheit und Kosten) und
- die Verfügbarkeit möglichst aller Werkzeuge und Bibliotheken im Quelltext
  (Unabhängigkeit von Herstellern, Erweiterbarkeit, Wartbarkeit und Kosten).

Des weiteren muß bei jeder Grundstruktur der Software immer gewährleistet sein, daß die gewünschte Funktionalität des Programms auch darstellbar ist. Z. B. sind im vorliegenden Fall umfangreiche Bildbearbeitungs– und –analysefunktionen zu implementieren, welche in der Regel zeitintensive Berechnungen anstellen. Eine Programmiersprache, die zur Laufzeit interpretiert wird (Skript–Sprachen, Java etc.), und bei der Algorithmen im Gegensatz zu Compiler–Sprachen langsam ablaufen, sollte hierbei nicht zum Einsatz kommen.

4.2.2 Vorgestellte Konzepte

Der Autor hat sich in der Folge eingehend mit einem breiten Spektrum an unterschiedlichen Technologien und Werkzeugen auseinandergesetzt, um eine möglichst vielseitige Palette an Lösungen vorstellen zu können. Vier unterschiedliche Lösungen konnten so zusammengestellt werden, die im folgenden näher beschrieben sind.

4.2.2.1 Lokale Lösung

Die denkbar einfachste Möglichkeit ist ein lokal auf dem Steuerrechner ablaufendes Programm, welches in einer Standard–Programmiersprache (in der Regel C) geschrieben wird. Abbildung 4.5 zeigt schematisch den konzeptionellen Aufbau dieser Lösung.[7] Über das Netzwerk meldet sich der Operateur bei dem Steuercomputer an und startet

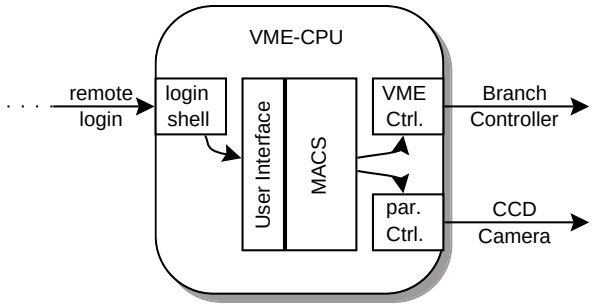


Abbildung 4.5: *Schematische Darstellung der Lokalen Lösung.*

lokal das Steuerprogramm.

Der offensichtliche Vorteil dieser Lösung ist die Verwendung allgemein bekannter Werkzeuge (z. B. der Programmiersprache C) ohne auf komplexe Technologien (z. B. Netzwerkprogrammierung) zurückzugreifen. Man kann davon ausgehen, daß nahezu jedes Mitglied der Kollaboration in der Lage ist, ein C–Programm zu warten bzw. zu erweitern. Diesem Vorteil stehen jedoch gravierende Nachteile gegenüber:

Sollte sich herausstellen, daß der Steuercomputer zu knapp dimensioniert wurde (Prozessorgeschwindigkeit, Speicherausbau usw.), kann das von der Programmierung kaum aufgefangen werden. Ein Auslagern von Teilen der Software auf andere Rechner wäre die geeignete Methode zur Entlastung des Steuercomputers, was allerdings die Aufgabe des lokalen Charakters zur Folge hätte. Eine Neukonzipierung wäre dann unumgänglich, wollte man eine saubere Strukturierung nicht aufgeben.

Zudem ist angedacht, die CCD–Kamera auch für andere Zwecke zu nutzen (z. B. zur Überwachung der Nachführungsgüte, der Abbildungsqualität und der Mißweisung der Teleskopkamera). Da diese Funktionalitäten ausschließlich von der Hamburger Gruppe

[7] In den Abbildungen zur Software finden Fachtermini Verwendung, für die es nach Meinung des Autors keine (kurzen) aussagekräftigen Entsprechungen im Deutschen gibt. Um dennoch Einheitlichkeit zu erlangen, wurden die Texte in diesen Abbildungen durchgehend in englischer Sprache gehalten, wofür der Autor um Verständnis bittet.

implementiert werden, muß der Steuercomputer die Kamera auch anderen Programmen zur Verfügung stellen. Es ist damit zu rechnen, daß ein solches Vorgehen zu Störanfälligkeit führt, da Kollisionen der um die CCD–Kamera konkurrierenden Programme kaum zu vermeiden wären.

Auch verhindert ein solches Vorgehen die saubere Integration des Justierprogrammes in die gesamte Teleskopsteuerung. Die zentrale Teleskopsteuerung müsste die Eingabe eines menschlichen Benutzers simulieren und die für menschliches Personal gedachten Antworten aufwendig interpretieren.

4.2.2.2 Client/Server–Lösung mit Sockets

Die oben geschilderte Problematik ist nicht neu; es gibt bereits ein Standardverfahren (als Client/Server–Modell bezeichnet), um solche Konstellationen programmtechnisch umzusetzen: Soll eine Ressource konkurrierenden Nutzern zur Verfügung gestellt werden, trennt man die Programmteile des Anfragers (engl. *client*) von denen des Anbieters (engl. *server*). Diese können dann auch auf unterschiedlichen Rechnern laufen, was den eigentlichen Vorteil dieses Konzeptes ausmacht.

Die Standardtechnologie der Netzwerkprogrammierung unter UNIX ist die Programmierung mit *Sockets* (ausführlich beschrieben z. B. in [Stev 98]). Sockets stellen dem Programmierer rudimentäre Anbindungen an das Netzwerk zur Verfügung. Ein wenig vergleichen läßt sich dies mit Ohr und Mund. Das Ohr meldet, wenn eine Nachricht angekommen ist. Der Mund ermöglicht es, Nachrichten in das Netzwerk zu schicken. Abbildung 4.6 versucht, das beschriebene Verfahren zu veranschaulichen.

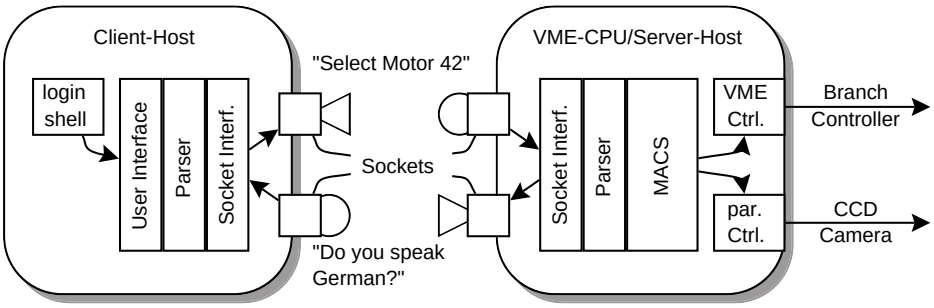


Abbildung 4.6: *Schematische Darstellung der Client/Server–Lösung mit Sockets.*

Die ankommenden Nachrichten zu interpretieren und entsprechende Aktionen einzuleiten bzw. gültige Nachrichten (in Abb. 4.6 durch „Select Motor 42“ symbolisiert) für andere zu formulieren, bleibt Aufgabe des Programmierers: Für jede Anwendung muß eine Sprache (das sogenannte Netzwerkprotokoll der höheren Schicht) definiert werden, mit der sich die Programmteile hinter Mund und Ohr verständigen können. Dies ist nicht ganz einfach, da ungültige Nachrichten (in Abb. 4.6 durch „Do you speak German?“ symbolisiert) erkannt und entsprechend behandelt werden müssen. Die Sprache unterliegt einer (strengen) Syntax, die bei jeder Nachricht zu überprüfen ist. Diese

Syntaxüberprüfung übernehmen sogenannte *Parser*, die üblicherweise mit speziellen Werkzeugen (auf UNIX–Systemen häufig lex & yacc) realisiert werden.

Durch die Verwendung dieser sehr allgemeinen Verfahren gewinnt man eine umfangreiche Flexibilität. Zum einen kann die Aufgabenverteilung zwischen Client und Server nahezu beliebig erfolgen; stellt sich z. B. heraus, daß der Speicher des Steuercomputers zu knapp bemessen wurde, können die speicherintensiven Programmteile einfach auf den Client ausgelagert werden. Zum anderen kann das Protokoll bei Bedarf dynamisch um Befehle erweitert werden; bisher nicht absehbare Aufgaben für die Steuerung können ohne großen Aufwand hinzugefügt werden. Der geringe Aufwand ergibt sich einerseits aus der strengen Syntax und andererseits aus dem Austausch der Nachrichten im ASCII–Format. Anhand einer neu festgelegten Nachricht können verschiedene Programmierer auf Client– und Server–Seite ohne weitere Absprachen die Funktionalität hinzufügen.

Auf der anderen Seite bedeutet diese Lösung eine extrem aufwendige Implementation. Die Entwicklung von Netzwerkanbindung und Parser ist nicht einfach und sehr fehleranfällig. Zudem würde der Programmcode äußerst komplex ausfallen, was zur Folge hätte, daß die Struktur eines nach dieser Methode umgesetzten Programms nicht leicht zu durchschauen wäre. Bereits einfache Änderungen können den Programmablauf an ganz anderen Stellen stören.

4.2.2.3 Client/Server–Lösung mit Remote Procedure Calls

Die Technologie der *Remote Procedure Calls* (RPC) wurde von der Firma Sun Microsystems spezifiziert und kommt in allen UNIX–Systemen zum Einsatz. Grundlegende Funktionalitäten, wie z. B. das *Network File System* (NFS), basieren auf RPC. Eine gute Einführung findet sich z. B. in [Stev 99].

RPC ermöglicht es, Prozeduren (Funktionen in C) des Server–Prozesses relativ einfach dem Client–Prozess nutzbar zu machen, ohne sich dabei um die zugrundeliegende Netzwerktechnologie kümmern zu müssen.

Alle Funktionen des Servers, die vom Client erreichbar sein sollen, werden entsprechend behandelt und so in den Client eingebunden, daß ein lokaler Aufruf im Client tatsächlich innerhalb des Servers abläuft. Um die technischen Details, wie Client und Server dabei miteinander kommunizieren, braucht sich der Programmierer nicht zu kümmern.

Anfragen des Client an den Server müssen nicht interpretiert werden, da auf der Client–Seite explizit die Funktion aufgerufen wird, die auf der Server–Seite ablaufen soll. Es müssen also weder die direkte Netzwerkanbindung, noch der Parser geschrieben werden, die den Einsatz der Lösung mit Sockets so aufwendig machen. Nachteilig ist jedoch, daß der Einsatz von RPC nicht immer reibungslos (insbesondere zwischen verschiedenen Betriebssystemen) funktioniert; z. B. bereitet Systemadministratoren die Anfälligkeit des *Network File Systems* (NFS), das wie erwähnt auf RPC aufsetzt, des öfteren Probleme.

Abbildung 4.7 zeigt den schematischen Aufbau dieser Lösung.

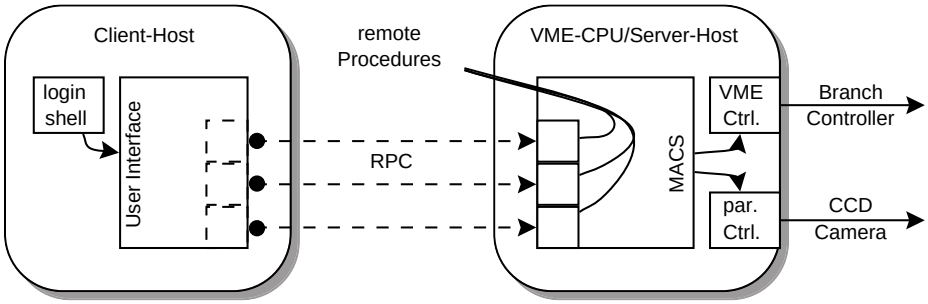


Abbildung 4.7: *Schematische Darstellung der Client/Server–Lösung mit Remote Procedure Calls.*

4.2.2.4 Client/Server–Lösung mit Distributed Objects

Gleich in mehreren Details unterscheidet sich die letzte vorgestellte Lösung von den vorherigen. Die Grundidee ist, sich von prozeduraler Programmierung zu lösen und moderne objektorientierte Verfahren zum Einsatz kommen zu lassen.

Konsequenterweise bedeutet dies nicht nur einen Wechsel der Programmiersprache; die sogenannten Klassenbibliotheken objektorientierter Programmiersprachen bieten zudem eine von der konkreten technischen Implementation abstrahierte Basis an Funktionalität, die meist weit über die Hälfte des sonst zu schreibenden Programmcodes als Baukasten zur Verfügung stellt. Auf die weiteren Vorteile objektorientierter gegenüber prozeduraler Programmierung soll hier nicht näher eingegangen werden. Hierzu sei z. B. auf die gelungene Einführung in [Appl 99] verwiesen.

Eng verknüpft mit der Programmiersprache Objective–C sind die *Frameworks* (Klassenbibliotheken) der *OpenStep*–Spezifikation [Next 94]. OpenStep ist ein Projekt der Firma NeXT Computer, das als Ziel eine einheitliche Umgebung für Programme und Benutzer hat, und zwar unabhängig davon, welches Betriebssystem dabei zum Einsatz kommt. Die Openstep–Spezifikation wird zur Zeit auch von einer Gruppe von Programmierern innerhalb des GNU–Projektes als freie Software – genannt GNUstep [GNUs 01] – umgesetzt. Alle Klassenbibliotheken sind im Quelltext verfügbar und laufen auf einer ganzen Reihe unterschiedlicher Betriebssysteme.

Ein kleiner aber fundamentaler Bestandteil von OpenStep ist das Foundation–Framework [Appl 01]. Dies ist eine Klassenbibliothek, in der sich etliche Klassen befinden, die ein breites Spektrum an häufig benötigter Grundfunktionalität abdecken und deren extensive Verwendung oft eine drastische Reduzierung des Programmieraufwandes ermöglicht. Das GNUstep–Projekt bietet neben dem Foundation–Framework noch weitere Komponenten, die aber bei den hier zur Diskussion stehenden Programmen nicht zum Einsatz kommen würden.

Das Foundation–Framework der GNUstep–Bibliotheken bietet einen hochentwickelten Mechanismus für die Client/Server–Programmierung. Die Technologie wird im Rah-

men von OpenStep mit *Portable Distributed Objects*[8] (PDO), bei GNUstep mit *GNU Distributed Objects* (GDO) bezeichnet. Ähnlich wie bei RPC kommunizieren Client und Server nicht direkt miteinander. Vielmehr werden auch hier lokal Funktionalitäten genutzt, die unter Umständen von Prozessen auf weit entfernten Rechnern zur Verfügung gestellt werden und dort ablaufen. Mit sehr wenigen zusätzlichen Zeilen Programmcode wird mittels GDO aus einem monolitischen ein verteiltes Client/Server–Programm. Bereits hierin zeigt sich die enorme Leistungsfähigkeit objektorientierter Programmierung. Abbildung 4.8 zeigt schematisch die Grundstruktur dieser Variante.

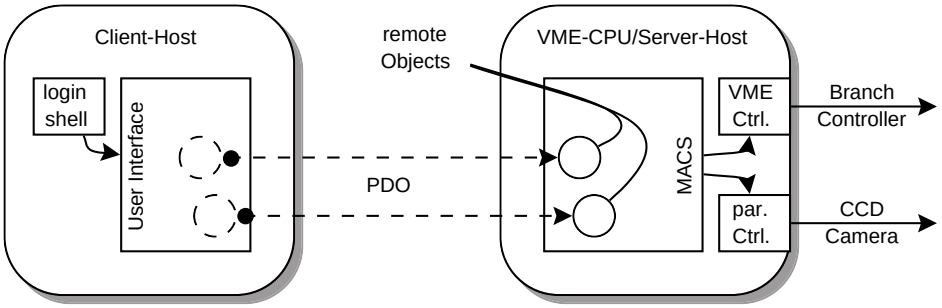


Abbildung 4.8: *Schematische Darstellung der Client/Server–Lösung mit Distributed Objects.*

Die vielfältigen Vorteile dieser Lösung erkauft man sich mit der Verwendung nicht allgemein bekannter Werkzeuge und Technologien. Wie hoch dieser Preis ist, läßt sich schwer abschätzen. Der Autor ist jedoch zu der Auffassung gelangt, daß der Aufwand, Objective–C zu lernen, in etwa dem gleichkommt, ein viel komplexeres C–Programm derart zu durchdringen, daß grundlegende Umstellungen bzw. Erweiterungen möglich sind.

4.2.3 Auswahl eines Softwarekonzeptes

Es wurden noch weitere Konzepte auf ihre Tauglichkeit für die vorliegende Aufgabe untersucht (z. B. C++ und Java), jedoch sämtlich bereits im Vorfeld verworfen. Gegen die Programmiersprache C++ spricht ihre Kompliziertheit und die Schwierigkeit, sie zu erlernen; gegen Java spricht die Interpretation zur Laufzeit in der Virtual–Machine, die lange Ausführzeiten komplexer Algorithmen zur Folge hat.

Zusammenfasssend sollen die verschiedenen Lösungen einander anhand der wichtigsten Kriterien gegenübergestellt werden. In Tabelle 4.1 findet sich dazu ein Überblick über die vorgestellten Konzepte. Es sei darauf hingewiesen, daß alle Angaben Einschätzungen des Autors sind und keinen Anspruch auf Allgemeingültigkeit erheben. Dennoch wurde dieser Versuch unternommen, um eine möglichst einfache Vergleichsmöglichkeit zu bieten.

Zur Verständlichkeit seien hier die verwendeten Bewertungskriterien kurz erläutert:

[8] engl. für portierbare verteilte Objekte

Tabelle 4.1: Vorgestellte Implementationsmöglichkeiten der Software

Lösung	Lokal	Client/Server		
Technologie		Sockets	RPC	PDO
Werkzeuge	C	C, lex & yacc	C, rpcgen	Objective–C, GNUstep
relativer Aufwand	100%	150%	110%	50%
Aufwand	0	− − −	−	+ + +
Flexibilität	− − −	+ +	+	+ + +
Kompatibilität	+ + +	+ +	+ +	−
Komplexität	0	− − −	−	+ +
Anfälligkeit	− − −	−	−	0

relativer Aufwand: Dieser gibt den grob geschätzten Aufwand an Entwicklungszeit relativ zur lokalen Lösung an. Implizit ist enthalten, wie gut der Autor mit den einzelnen Technologien vertraut ist.

Aufwand: Ein Versuch, den Aufwand an Entwicklungszeit zu bewerten.

Flexibilität: In die Bewertung der Flexibilität geht ein, wie leicht sich die jeweilige Lösung erweitern und neuen bzw. unvorhergesehenen Bedingungen anpassen läßt.

Kompatibilität: Die Kompatibilitätswertung gibt an, inwieweit die verwendeten Technologien verfügbar bzw. bereits auf gängigen Systemen vorinstalliert sind. Zudem ist enthalten, wie gut die Software auf andere Rechner– bzw. Betriebssysteme portierbar ist.

Komplexität: Unter Komplexität ist zusammengefaßt, wie viele Komponenten zu entwickeln sind und wie klar sich die resultierende Programmstruktur gestaltet. Auch fließt hier ein, wie gut die einzelnen Komponenten voneinander zu kapseln sind und wie verständlich die Programmstruktur für Dritte ist.

Anfälligkeit: Diese Bewertung gibt an, wie problemanfällig die Software ist und wie gut äußere Fehler bzw. Schwierigkeiten vom Programm erfaß– und entsprechend behandelbar sind.

Zur Übersichtlichkeit wurden die einzelnen Kriterien symbolisch von − − − (für sehr schlecht) bis + + + (für sehr gut) bewertet.

In der H·E·S·S·–Gruppe Hamburg wurden die vorgestellten Implementationsmöglichkeiten eingehend diskutiert. Es stellte sich – für den Autor überraschend – heraus, daß gerade die extremen Lösungen favorisiert wurden. Für die lokale Implementation sprechen ihre einfache Gestaltung und die Verwendung kanonischer Technologien und Werkzeuge; für die objektorientierte Client/Server–Version mit GDO, unter Verwendung von Objective–C und GNUstep, ihr klares Design und die Mächtigkeit der eingesetzten Technologien.

Die sich an die Diskussion anschließende Entscheidung fiel zugunsten des objektorientierten Ansatzes aus. Betont wurde jedoch, die Flexibilität objektorientierter Programmierung dahingehend zu nutzen, die hardwareabhängigen Teile derart zu kapseln, daß ein eventueller Wechsel von CCD–Kamera bzw. Steuerungshardware mit wenigen lokalen Eingriffen zu ermöglichen ist.

4.3 Softwarekomponenten und ihre Modellierung

Nachdem in Kapitel 4.2 das Grundkonzept der Software zur Justierung der Spiegelfacetten der H·E·S·S·–Teleskope – unter Einbeziehung der zum Einsatz kommenden Technologien – ausführlich dargelegt wurde, soll im folgenden genauer auf die innere Modellierung der Komponenten eingegangen werden.

Sämtliche Teile der Software wurden in der objektorientierten Programmiersprache Objective–C geschrieben. Die im folgenden zur Beschreibung der Programme verwendeten Begriffe entstammen daher vielfach dem Sprachgebrauch der objektorientierten Programmierung (OOP). Es würde den Rahmen dieser Arbeit sprengen, sie hier in allen Einzelheiten aufzuführen; es sei daher auf die sehr gelungene Einführung in [Appl 99] verwiesen.

4.3.1 Die Softwarekomponenten im Überblick

Der erste Schritt auf dem Weg zu einem Programm vom Umfang der hier zur Diskussion stehenden Aufgabe, besteht in der möglichst detailreichen Konzipierung des Gesamtsystems. Dazu sollte das System zunächst in natürliche, d. h. logisch zusammengehörige, funktionale Einheiten untergliedert werden. Jede dieser Komponenten sollte dabei nach Möglichkeit soweit abgeschlossen sein, daß eine nahezu unabhängige Entwicklung betrieben werden kann. Dazu ist es wichtig, die Schnittstellen zwischen den Komponenten bereits frühzeitig festzulegen. Diese Vorgehensweise macht auch ein umfangreiches Projekt überschaubar und zeigt bereits im Vorfeld mögliche Schwächen auf.

So wurde auch bei der Entwicklung des Systems zur Justierung der Spiegelfacetten verfahren. Abbildung 4.9 zeigt eine schematische Darstellung des Gesamtsystems. Die eigentliche MACS–Software wurde in drei Hauptkomponenten untergliedert. An die VME–CPU werden die beiden zu steuernden Geräte, *Branch Control Unit* und CCD–Kamera, direkt angeschlossen. Für jedes dieser Geräte ist jeweils ein eigener Prozeß zur Kommunikation vorgesehen, der unabhängig vom Rest der Software bestehen soll. Beide Prozesse sind zur direkten Ansteuerung der Hardware gedacht, wobei sie diese Möglichkeit allerdings nur anderen Teilen der Software zur Verfügung stellen, ohne – bis auf die Initialisierung der Hardware – selbst aktiv zu werden. Damit fungieren sie als sogenannte Server–Prozesse; sie vermitteln den Zugriff auf die Hardwarekomponenten über das Netzwerk an die Softwarekomponenten, die diesen Dienst in Anspruch nehmen.

Die zentrale Komponente der MACS–Software beinhaltet die gesamte Logik zur Justierung der Spiegelfacetten. In ihr werden alle Verfahren zur Analyse der Daten und zur Ableitung der zur Justierung nötigen Steuerbefehle implementiert. Zur Ansteuerung

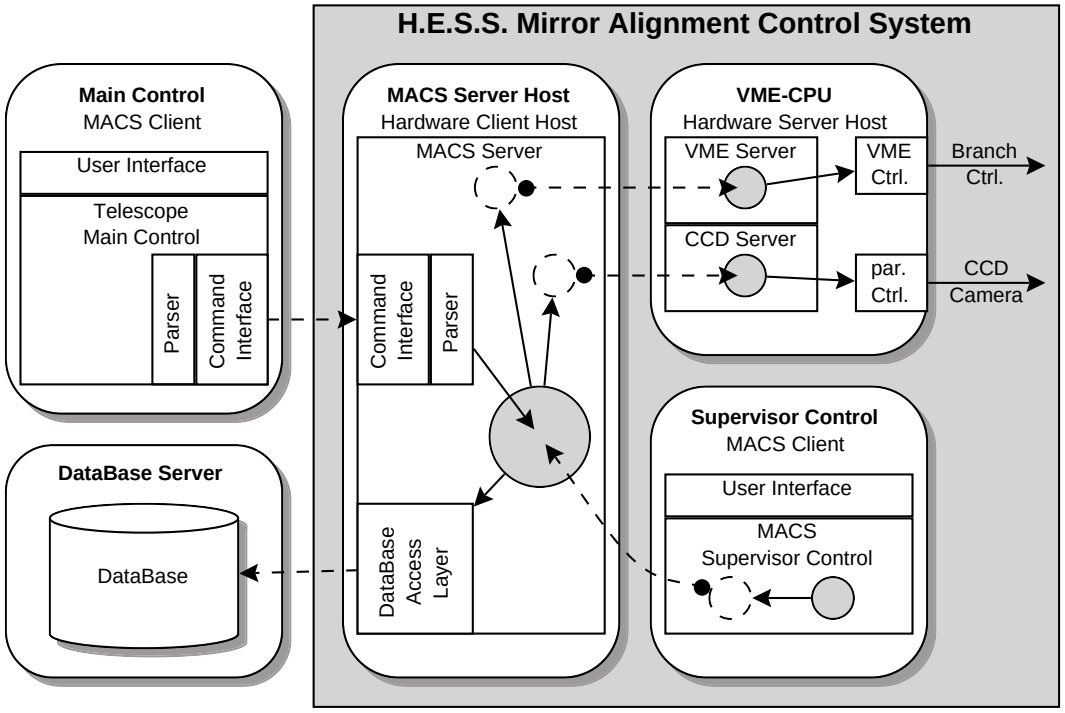


Abbildung 4.9: *Überblick über die Softwarekomponenten des* H·E·S·S· *Mirror Alignment Control Systems (MACS).*
*Das System besteht aus drei Hauptkomponenten. Auf der VME–CPU laufen zwei Server–Prozesse (VME Server und CCD Server, rechts oben im Bild), die jeweils den Zugriff auf die zu betreibenden Geräte (*Branch Control Unit *und CCD–Kamera) über das Netzwerk exportieren. Die gesamte Logik zur Justierung befindet sich in der zentralen Komponente (MACS Server, in der Bildmitte), die die beiden Server auf der VME–CPU zur Ansteuerung der Geräte nutzt. Dieser Prozeß agiert ebenfalls als Server, um die Justierung verschiedenen Clients zur Verfügung stellen zu können. Als mögliche Clients kommen dabei die zentrale Steuerung für das gesamte* H·E·S·S·*–System (Main Control, links oben) sowie ein spezielles Programm zur unabhängigen Interaktion mit dem MACS Server (Supervisor Control, rechts unten) in Frage. Zusätzlich ist geplant, wichtige bzw. interessante Parameter in einer* H·E·S·S·*–weiten Datenbank (DataBase Server, links unten) abzulegen, um sie durch einfachen Zugriff für alle Benutzer zugänglich zu machen.*

der Geräte nutzt sie dabei die beiden Hardware–Server über das Netzwerk. Das hierbei zum Einsatz kommende Verfahren zur Kommunikation über das Netzwerk ist – wie in Kapitel 4.2.2.4 erläutert – *GNU Distributed Objects* (GDO).

Bei der Konzipierung des Systems wurde darauf geachtet, es möglichst flexibel zu gestalten. So ist z. B. vorgesehen, die Justierung selbst als Dienst anderen Nutzern zur Verfügung zu stellen. Deshalb agiert auch die zentrale Komponente als Server–Prozeß;

sie exportiert ihre Funktionalität an andere Programme über das Netzwerk, wobei zur Zeit zwei unterschiedliche Client–Prozesse vorgesehen sind. Zum einen ist dies die zentrale Steuerung des gesamten H·E·S·S·–Systems, von der aus sämtliche Funktionalitäten der Teleskope dem Bedienpersonal zugänglich sein sollen. Zum anderen erschien es dem Autor als sinnvoll, eine von dieser Zentrale unabhängige Möglichkeit zu haben, die Justierung nutzen zu können. Dazu ist ein Programm zur direkten Ansteuerung der Justiersoftware vorgesehen. Zusätzlich zur Unabhängigkeit gewinnt man so den Zugriff auf interne Funktionalitäten, die dem normalen Bedienpersonal nicht zugänglich gemacht werden sollen, die sich jedoch bei der Erstinbetriebnahme bzw. während der Fehlerbereinigungsphase als sehr hilfreich erweisen können.

Die Technologie zur Kommunikation aller H·E·S·S·–Dienste über das Netzwerk ist bisher noch nicht spezifiziert worden. Zur Diskussion stehen zwei Verfahren (*hcomm* [Bern 00] und *CORBA* [OMG 00]), auf die hier nicht näher eingegangen werden soll. Beide Verfahren wurden jedoch mit Teilen der Software getestet und erwiesen sich als relativ leicht in die MACS–Software integrierbar.

Abschließend ist vorgesehen, Parameter von allgemeinem Interesse in einer H·E·S·S·–weiten Datenbank ablegen zu können. Dies könnten z. B. Informationen über defekte oder ungewöhnlich schwergängige Motoren bzw. Aktuatoren sein. Durch den i. a. einfachen Zugriff auf Datenbanken wären diese Informationen so allen Interessierten leicht zugänglich.

Die beschriebene Untergliederung in funktionale Einheiten erwies sich als sinnvoll und hilfreich. Nicht nur konnten die Komponenten unabhängig voneinander entwickelt werden, sie konnten zudem auch vollständig ausgetestet werden, ohne andere Teile der Software fertiggestellt haben zu müssen.

In den folgenden Kapiteln soll nun eingehender auf die innere Struktur der Softwarekomponenten eingegangen werden. Anzumerken ist, daß während der Durchführung dieser Arbeit nicht das gesamte Softwaresystem erstellt werden konnte. Die erbrachten Leistungen sollten jedoch eine gute Grundlage zur Vervollständigung bieten. Näheres dazu findet sich in Kapitel 6.2.

4.3.2 Basisklassen

Um nicht jedes zu erstellende Programm wieder von Grund auf neu schreiben zu müssen, wurden immer wieder gebrauchte Funktionalitäten in sogenannte Basisklassen zusammengefaßt. Diese Basisklassen wurden so konzipiert, daß sie für möglichst viele der allgemeinen Anforderungen an ein Programm eine Lösung bieten. Ziel bei der Entwicklung von Basisklassen ist, sich bei der Entwicklung darauf aufbauender Programme auf die eigentlich abzubildende Funktionalität konzentrieren zu können.

Programme nutzen die durch die Basisklassen zur Verfügung gestellten Funktionalitäten, indem sie entweder diese Basisklassen direkt einbinden oder abgeleitete Klassen von ihnen bilden, welche in der Folge um applikationsspezifische Dinge erweitert werden können.

4.3.2.1 HESS–Application

Die fundamentale Basisklasse *HESSApplication* bildet die Grundlage für alle im Rahmen der Software zur Justierung der Spiegelfacetten zu entwickelnden Programme. Sie wurde so ausgelegt, daß möglichst viele der immer wieder benötigten Funktionalitäten enthalten sind, um sich bei der Entwicklung aller darauf aufbauender Programme auf die zu implementierende Aufgabe konzentrieren zu können. Dabei wurde darauf geachtet, die Funktionalitäten möglichst so abstrakt und allgemeingültig zu gestalten, daß die Umsetzung der darauf aufbauenden Programme keiner Einschränkung unterliegt.

Die Basisklasse *HESSApplication* implementiert die folgenden Funktionalitäten:

1. Ausgabe von Programminformationen.
   Es ist guter Stil, Programmen die Fähigkeit mitzugeben, Auskunft über sich selbst geben zu können. Dazu gehören Informationen wie Name, Versionsnummer, Hersteller bzw. Programmierer des Programms sowie die Angabe der erlaubten Kommandozeilenoptionen (s. u.).
   Der Klasse *HESSApplication* wird durch den Aufruf ‚*Programmname* `-help`‘ mitgeteilt, diese Informationen auszugeben. Sollten Programme eine detailliertere oder gänzlich anders geartete Ausgabe benötigen, kann der Standardmechanismus leicht durch eine spezielle Version ersetzt werden. Zudem wurde die Option ‚*Programmname* `-c`‘ geschaffen, die das Programm veranlasst, die *Default–Konfiguration* (s. u.) auf dem Bildschirm auszugeben. Diese gibt nicht nur Auskunft über die Voreinstellungen des Programms, sie kann zudem als Basis für eine angepaßte Konfigurationsdatei dienen.

2. Parametrisierung zur dynamischen Festlegung des Verhaltens.
   Im Allgemeinen ist es unerwünscht oder unmöglich, das gesamte Verhalten eines Programms bereits im Quelltext festzulegen. Nahezu jedes Programm wird deshalb so entwickelt, daß es dynamisch an die jeweiligen Bedingungen anpaßbar ist. Dies erreicht man dadurch, daß gewisse Einstellungen erst beim Start des Programms vorgenommen werden, wobei dies durchaus auf unterschiedlichen Wegen geschehen kann.
   Dieser Mechanismus wurde in der Basisklasse *HESSApplication* implementiert, wobei darauf geachtet wurde, ihn möglichst flexibel zu gestalten. Nachfolgend eine kurze Beschreibung der umgesetzten Möglichkeiten, Programmverhalten per Optionen beieinflussen zu können:

   - Festlegung einer sogenannten *Default–Konfiguration*.
     Selbst wenn ein Programm ohne explizite Angabe von Optionen gestartet wird, sollten die einzelnen Parameter – soweit möglich – mit vernünftigen Werten besetzt sein. Einen solchen Satz von Standardwerten nennt man *Default–Konfiguration*.
     Die Basisklasse *HESSApplication* bietet die Möglichkeit, solch eine Default–Konfiguration bereits im Quelltext festzulegen. Zudem wurde ein Verfahren implementiert, die Default–Konfiguration strukturiert ausgeben zu können,

um sie sich z. B. als Basis für eine Konfigurationsdatei (s. u.) nutzbar zu machen.

- Interpretation der Kommandozeile zur Beeinflussung einzelner Programmoptionen.
  Eine häufig genutzte Möglichkeit, Programmoptionen zu spezifizieren, ist die Mitgabe der Einstellungen beim Programmstart per Komandozeile. Allerdings ist nicht immer erwünscht (z. B. aus Sicherheitsgründen), daß alle Einstellungen von der Kommandozeile aus veränderbar sind.
  Die Klasse *HESSApplication* ist in der Lage, die Kommandozeile zu interpretieren und beliebige Optionen mit angegebenen Werten zu belegen. Dies folgt dem Schema *Optionname*=*Wert*, wobei überprüft wird, ob *Optionname* eine gültige Option ist und ob diese von der Kommandozeile aus verändert werden darf.

- Auslesen einer spezifischen Konfiguration aus einer Konfigurationsdatei.
  Es ist mitunter sehr mühsam, einem Programm vom Normalwert abweichende Parameter bei jedem Start neu mitgeben zu müssen. Dazu nutzt man Konfigurationsdateien, welche erlauben, Programmeinstellungen auch über die Laufzeit eines Programmes hinaus zu erhalten.
  Die Möglichkeit, eine Konfigurationsdatei festzulegen, diese auszulesen und deren Inhalt zu interpretieren wurde der Klasse *HESSApplication* mitgegeben.

- Unkomplizierte Verwaltung aller Programmoptionen.
  Selbstverständlich ist, daß der Parametrisierungsmechanismus von den Programmen einfach zu nutzen ist. Dies umfaßt einfache Verfahrensweisen, die Werte für bestimmte Optionen zu erfahren und zu verändern, sowie Gültigkeitsbereiche der angegebenen Werte überprüfen zu können.

- Festlegung von Erklärungstexten zu einzelnen Programmoptionen.
  Aus dem Namen einer Option läßt sich in den seltensten Fällen die vollständige Bedeutung ablesen. Zudem ist aus dem Namen i. a. allein nicht ersichtlich, wie der gültige Wertebereich beschaffen ist, d. h. welche Werte zulässig sind und welche nicht.
  Deshalb implementiert die fundamentale Basisklasse *HESSApplication* die Möglichkeit, einen Erklärungstext zu jeder Option anzugeben. Dieser wird beim Aufruf von *Programmname* `-help` (s. o.) auf dem Bildschirm ausgegeben.

Der gesamte Parametrisierungsmechanismus wurde fehlertolerant gestaltet. Es wird überprüft, ob die spezifizierten Optionen überhaupt existieren bzw. ob sie von z. B. der Kommandozeile aus verändert werden dürfen.

3. Verwaltung von Laufzeitinformationen.
   Programmausgaben, die nicht unmittelbar an den Benutzer gehen, nennt man Laufzeitinformationen.

Die Klasse *HESSApplication* bietet den auf sie aufbauenden Programmen ein sechsstufiges System zur Ausgabe von Laufzeitinformationen. Die Stufe, ab der Informationen tatsächlich ausgegeben werden (mit Prioritäten größer oder gleich der gewählten), ist dabei frei konfigurierbar. Die nach aufsteigender Priorität geordneten Stufen sind die folgenden:

- Ausführliche Ausgaben zum Zwecke der Fehlerbeseitigung (Stufe 5: `DEBUG`). Der Prozeß der Programmfehlerbeseitigung wird im Fachjargon *Debugging* (von engl. *to debug*, etwa entwanzen) genannt. Während der Entwicklung eines Programms ist es zur Erkennung von Fehlern sehr hilfreich, genaue Informationen darüber zu haben, welcher Teil des Programms gerade zur Ausführung kommt. Auch kann es sinnvoll sein, ausführliche Informationen über den Zustand wichtiger Programmvariablen etc. zur Verfügung zu haben. Im Normalbetrieb wird man dagegen von der Ausgabe solcher Informationen absehen, da sie in der Regel bereits nach kurzer Laufzeit sehr umfangreich werden.
- Informationsmeldungen (Stufe 4: `INFO`).
  Bei der Abarbeitung eines Programms kann es mitunter sinnvoll sein, Informationen von allgemeinerem Charakter auszugeben. Dazu zu zählen wären z. B. der Zeitpunkt von Programmstart und –ende sowie Angaben darüber, welche Geräte zur Benutzung geöffnet werden.
- Auskunft über eingeleitete Aktionen (Stufe 3: `CONTROL`).
  Ein durchaus interessantes Ereignis ist, wenn ein Programm von einem Benutzer oder einem Client–Programm mit der Ausführung einer Aktion beauftragt wird. Anhand genauer Information über alle eingeleiteten Aktionen kann ein komplexer Ablauf später sehr ganau analysiert werden.
- Warnungen (Stufe 2: `WARNING`).
  Warnungsmeldungen folgen Ereignissen, die zwar noch nicht als Fehler zu werten sind, deren Eintreffen jedoch von großem Interesse zur Erkennung bzw. Vermeidung von Problemen sind. Es kann z. B. unerwünscht sein, wenn ein Client einen Server–Prozeß zu kontaktieren versucht, welcher bereits mit einem anderen Client kommuniziert. Wird der zweite Client vom Server–Prozeß abgelehnt, ist dies zwar aus Sicht des Servers nicht als Fehler zu werten, eine Warnung an den Operateur ob dieses Umstands ist jedoch sehr sinnvoll.
- Fehlermeldungen (Stufe 1: `ERROR`).
  Kommt es in einem Programm zu einem unspezifizierten Ereignis (Fehler), so muß diesem Rechnung getragen werden. Oft kann ein Programm die Ursache des Fehlers bereinigen und ungestört mit der Bearbeitung fortfahren. Wird z. B. beim Programmstart ein Parameter mit einem zu hohen Wert belegt, kann es sinnvoll sein, diesen durch den spezifizierten Höchstwert zu ersetzen, ohne das Programm zu beenden. Eine Meldung darüber sollte aber in jedem Fall erfolgen.

- Meldungen über Fehler, die zu Programmabbrüchen führen (Stufe 0: `FATAL`). Tritt in einem Programm ein nicht zu behebender Fehler auf, sollte es genaue Auskunft über die Ursache geben und sich so sauber wie möglich beenden. Ist ein Programm z. B. darauf angewiesen, daß ein bestimmtes Gerät angeschlossen und funktionsbereit ist, kann eine vernünftige Bearbeitung nicht erfolgen, wenn dies nicht der Fall ist.

Die Ausgabe der Laufzeitinformationen kann wahlweise auf den Bildschirm, in eine sogenannte Log–Datei oder durch Übergabe an den Systeminformationsdienst *syslog* erfolgen.

Durch Einbeziehung der Klasse *HESSApplication* kann ein zu entwickelndes Programm – unter voller Nutzung der dargestellten Funktionalitäten – bereits sehr kurz und effizient implementiert werden.

4.3.2.2 HESS–Daemon

Programme, die ihren Dienst ohne direkte Interaktion mit den Benutzern verrichten, nennt man *Daemons*[9]. In UNIX–basierten Systemen laufen viele solcher Programme (z. B. `inetd`, `syslogd` und `nfsd`) nahezu unbemerkt im Hintergrund. Sie offerieren anderen Programmen in der Regel System– und sogenannte Server–Dienste und werden meist nur zur Reinitialisierung kurzzeitig beendet. Eine Einführung in die Programmierung von Daemons findet sich z. B. in [Stev 93].

Einige der Programme zur Justierung der Spiegelfacetten sind solcherart Dienstanbieter, weshalb eine Klasse zur Abbildung dieser Eigenschaft implementiert wurde. Die Klasse *HESSDaemon* ist eine von *HESSApplication* abgeleitete Klasse, die diese um jene spezielle Funktionalität erweitert. Zusätzlich zu der Möglichkeit, ein darauf aufbauendes Programm als Deamon ablaufen zu lassen, bietet sie somit die gesamte Funktionalität der Klasse *HESSApplication*.

Da ein Daemon kurz nach dem Programmstart in den Hintergrund gebracht wird, besteht fortan keine Möglichkeit zur direkten Interaktion mit dem Benutzer; Programmausgaben auf dem Bildschirm sind also nicht möglich. Die Klasse *HESSDaemon* überprüft deshalb, ob spezifiziert wurde, die Laufzeitinformationen auf dem Bildschirm auszugeben. Da dies nicht zulässig ist, beendet sich das Programm in diesem Fall mit einer Fehlermeldung. Laufzeitmeldungen müssen daher entweder in eine Datei geschrieben oder aber dem Systeminformationsdienst *syslog* übergeben werden.

4.3.3 Controller–Klassen für die Hardware

Klassen, die zur Verwaltung von Objekten anderer Klassen (sogenannte Instanzen, von engl. *instance object*) oder Geräten (*devices*) bestimmt sind, nennt man *Controller–Klassen* (engl. *controller classes*). Controller–Klassen dienen dazu, komplexe – meist

[9] Engl. Begriff, der in diesem Kontext etwa für Helfergeist steht; nicht zu verwechseln mit engl. *demon*, das eher mit dem Begriff Dämon zu übersetzen ist und meist böse Absichten impliziert.

aus vielen primitiven Anweisungen zusammengesetzte – Aktionen an den zu verwaltenden Objekten durchzuführen, wobei sie den Zustand dieser Objekte ändern. Sie selbst bleiben dabei in der Regel zustandslos und reichen Zustandsinformationen der zu verwaltenden Objekte nur an die den Controller nutzenden Programmteile weiter.

Typisches Beispiel für einen Controller ist eine Klasse, die für die korrekte Ansteuerung eines Gerätes verantwortlich ist. Viele Geräte verfügen nur über relativ primitive Befehle; erst eine geeignete Kombination dieser Befehle ergibt einen sinnvollen Bearbeitungsschritt, wobei die Zulässigkeit meist eng an den Zustand des Gerätes gekoppelt ist. So macht es z. B. wenig Sinn, ein Bild mit einer Kamera aufzunehmen, deren Objektiv nicht geöffnet ist. Der Befehl zum Aufnehmen eines Bildes hat also dafür zu sorgen, daß die Kamera eingeschaltet und das Objektiv geöffnet wird, bevor es zur eigentlichen Aufnahme kommt.

Für die Ansteuerung der Geräte zur Justierung der Spiegelfacetten wurden entsprechende Controller–Klassen entwickelt, die im folgenden näher vogestellt werden.

4.3.3.1 MACS–VME–Controller

Für die Verwaltung des VMEbus wurde die Klasse *MACSVMEController* geschaffen. Sie dient zur Initialisierung des VMEbus und soll den sie nutzenden Programmen einen einfachen Zugriff auf an den VMEbus angeschlossene Geräte bieten. Zur tatsächlichen Übertragung der Daten über den VMEbus kommt dabei der frei verfügbare Linux–Gerätetreiber *universe* [Hann 00] zum Einsatz.

Der VMEbus selbst verfügt über eine Vielzahl unterschiedlicher Kommunikationsmethoden. Das *VME–Board* zur Kommunikation mit der *Branch Control Unit* nutzt davon jedoch nur den A16/D16–Modus, bei dem mit 16 Bit kodierte Werte an bzw. von Adressen übertragen werden, die ebenfalls mit 16 Bit kodiert sind. Die Klasse *MACSVMEController* implementiert deshalb zur Zeit nur diesen Modus; sie könnte bei Bedarf jeoch einfach um weitere Modi erweitert werden.

Zur Kommunikation mit Geräten über den VMEbus stellt die Klasse zwei sogenannte Methoden (Fachausdruck, von engl. *method*) zur Verfügung:

1. Lesen eines Wortes von einer zu spezifizierenden Adresse.
   Anzugeben ist die gewünschte Adresse auf dem VMEbus. Die Methode liefert dann das von dieser Adresse gelesene Datenwort als Ergebnis an den aufrufenden Programmteil zurück.

2. Schreiben eines anzugebenden Wortes an eine zu spezifizierende Adresse.
   Diese Methode schreibt das gewünschte Datenwort an die angegebene Adresse, wobei keine Werte an den aufrufenden Programmteil zurückzugeben sind.

Mittels dieser beiden Methoden ist es Programmen möglich, die an den VMEbus angeschlossenen Geräte – insbesondere also die *Branch Control Unit* – vollständig über den VMEbus zu kontrollieren, ohne dabei von den konkreten Übertragungsmechanismen Kenntnis haben zu müssen.

4.3.3.2 MACS–Branch–Controller

Die *Branch Control Unit* verfügt über einen definierten Satz von Kommandos, mit dem alle ihre Aktionen kontrolliert werden können. Zwar wäre es möglich, die Kommandierung der *Branch Control Unit* allein mit den von der Klasse *MACSVMEController* zur Verfügung gestellten Methoden zu betreiben, allerdings geben die dabei zum Einsatz kommenden Bitmuster nur bedingt Auskunft über die Art des jeweiligen Befehls.

Zur Abstrahierung der Befehle von ihrer technischen Umsetzung wurde deshalb die Klasse *MACSBranchController* geschaffen. Sie bietet für jeden Befehl der *Branch Control Unit* eine entsprechende Methode, deren Namen bereits die Art des Befehls verdeutlicht. Somit ist aus dem Quelltext unmittelbar ersichtlich, welcher Befehl an die *Branch Control Unit* übertragen wird; es tauchen nicht die schwer interpretierbaren Bitmuster der Methoden der Klasse *MACSVMEController* auf. Zudem vermeidet diese Vorgehensweise, daß ungültige Kommandos (nicht spezifizierte Bitmuster) an die *Branch Control Unit* übermittelt werden, was bei direkter Verwendung von Bitmustern sehr leicht geschehen kann.

Aber selbst die Nutzung dieser von der technischen Umsetzung abstrahierten Methoden beinhaltet drei Nachteile:

1. Einige Befehle kodieren mehr als eine Information.
   Aus Effizienzgründen kodieren einige der Befehle der *Branch Control Unit* mehrere Informationen zugleich. So müssen z. B. Normal– und Kriechgeschwindigkeit immer zusammen gesetzt werden.
   In der Steuerungssoftware wird man dagegen oft nur einen bestimmten Parameter setzen wollen oder ist an einer bestimmten Information – z. B. ob ein Motor läuft – interessiert.

2. Die Methoden sind nicht an den Zustand der *Branch Control Unit* gekoppelt.
   Zwar ist es durchaus möglich, jeden Befehl zu beliebiger Zeit an die *Branch Control Unit* zu senden. Dies macht in vielen Fällen jedoch keinen Sinn oder kann gar zu unvorhersehbaren Effekten führen.
   Man wird daher versuchen, einen Befehl nur dann zur Ausführung kommen zu lassen, wenn er in dem Zustand, in dem sich die *Branch Control Unit* befindet, zulässig und sinnvoll ist. Es ist z. B. wenig sinnvoll – und im Sinne der Steuerungssoftware sogar gefährlich –, den Befehl zum Starten eines Motors an die *Branch Control Unit* zu übermitteln, wenn noch kein Motor selektiert wurde.

3. Die Methoden selbst bieten keine Möglichkeit, Fehlkommandierungen zu erkennen.
   Es ist nie auszuschließen, daß die Kommunikation zwischen Steuerungssoftware und *Branch Control Unit* etwa durch defekte oder falsch gesteckte Verbindungen gestört ist. Auch besteht immer die Möglichkeit, daß die *Branch Control Unit* einen Defekt aufweist und die übermittelten Befehle nicht korrekt bearbeitet.
   Daher wird man bemüht sein, nach jedem Befehl zu überprüfen, ob dieser auch korrekt ausgeführt wurde. Andernfalls droht beim Nichterkennen solcher Proble-

me unvorhersagbares Programmverhalten, was eine gezielte Fehleranalyse nahezu unmöglich macht.

Bei Befehlen, die mehrere Informationen kodieren, wäre es aufwendig und unübersichtlich, die interessierende Information jedesmal extrahieren bzw. alle nötigen Parameter zusammenstellen zu müssen. Zudem wäre es mühsam und sehr fehleranfällig, vor jedem zu übermittelnden Befehl zu überprüfen, ob die Ausführung zulässig und sinnvoll ist. Dies gilt auch für die Kontrolle der korrekten Abarbeitung nach der Übermittlung.

Aus diesen Gründen wurde zusätzlich zu den die Befehle abbildenden Methoden eine zweite, weiter abstrahierte Ebene geschaffen. Die Methoden dieser höheren Ebene (engl. *high–level interface*) bieten der Steuerung eine äußerst flexible Kommandierung der *Branch Control Unit*. Sie sind vollständig von der tatsächlichen technischen Umsetzung der einzelnen Kommandos entkoppelt, stellen also Schreib– und Lesebefehle für einzelne Parameter bzw. Informationen zur Verfügung ohne die genaue Kodierung der entsprechenden Kommandos kennen zu müssen. Mit ihnen kann z. B. die Normalgeschwindigkeit einzeln gesetzt werden, ohne daß die Notwendigkeit besteht, auch die Kriechgeschwindigkeit angeben zu müssen. Zudem überprüfen sämtliche Methoden dieser Ebene, ob der gewünschte Befehl zulässig, d. h. ob er zum Zustand der *Branch Control Unit* kompatibel ist. Sollte dies nicht der Fall sein, wird ein sogenannter Ausnahmefehler (engl. *exception*) generiert, der dem aufrufenden Programmteil genaue Information über den Grund des Fehlschlags gibt. Abschließend wird nach jedem Befehl ermittelt, ob dieser von der *Branch Control Unit* auch korrekt abgearbeitet wurde. Auch hier wird ein aussagekräftiger Ausnahmefehler generiert, falls es zu Fehlkommandierungen kommt.

Die Einführung der höheren Ebene zur Kommandierung der *Branch Control Unit* erwies sich als äußerst erfolgreich. So konnten mit ihr die in Kapitel 5.2.3 beschriebenen Fehler in der Kommandierung leicht lokalisiert und komfortabel bereinigt werden. Denn die Probleme hatten so kein unerwartetes Programmverhalten zur Folge, sondern führten sofort zu Ausnahmefehlern, die genaue Auskunft über das Fehlverhalten gaben.

4.3.4 Der VME–Server–Daemon (macsVMEd)

Eine zentrale Komponente der Software zur Justierug der Spiegelfacetten der H·E·S·S·–Teleskope ist das Programm zur Kommunikation mit der *Branch Control Unit* über den VMEbus.

Aufgabe dieses Programms ist es, anderen Softwarekomponenten den Zugriff auf den VMEbus – und damit der *Branch Control Unit* – zur Verfügung zu stellen. Wie in Kapitel 4.2.3 beschrieben, wurde aus Gründen der Flexibilität entschieden, dieses Programm als Server–Prozeß auszulegen, welcher den Zugriff auf den VMEbus über das Netzwerk an die anderen Teile des Systems exportiert. Da bei diesem Prozeß somit keine direkte Kommunikation mit Benutzern auftritt, gehört er zur Klasse der *Daemons*.

Aufbauend auf die Klassen *HESSDaemon* (vgl. Kap. 4.3.2.2; beinhaltet *HESSApplication*, vgl. Kap. 4.3.2.1) und *MACSVMEController* (vgl. Kap. 4.3.3.1) konnte das

Programm recht zügig entwickelt werden. Im wesentlichen kreiert der VME–Server–Daemon nur ein Instanzobjekt (engl. *instance object*, etwa konkretes Exemplar) der Klasse *MACSVMEController*, welches dann mittels *Distributed Objects* am Netzwerk anderen Programmen zur Verfügung gestellt wird. Ein Client–Prozeß, der Zugriff auf den VMEbus der VME–CPU erhalten möchte, kontaktiert den VME–Server–Daemon, wobei automatisch ein lokales Stellvertreterobjekt (engl. *proxy object*) für das zu importierende Objekt erzeugt wird. Fortan kann das Client–Programm mit diesem so importierten Objekt interagieren, als wäre dies ein lokales – im Client ansässiges – Objekt, ohne sich dabei um die tatsächlich stattfindende Kommunikation über das Netzwerk kümmern zu müssen.

Zur eigentlichen Kommunikation mit der *Branch Control Unit* nutzen die Client–Programme allerdings die speziell für diesen Zweck entwickelte Klasse *MACSBranch-Controller* (vgl. Kap. 4.3.3.2), die den Datentransfer über den VMEbus mittels dem importierten Instanzobjekt der Klasse *MACSVMEController* transparent abwickelt.

Der VME–Server–Daemon wird durch die Eingabe von

```
macsVMEd
```

auf der Kommandozeile gestartet. Sofort nach dem Start schaltet er sich selbst in den Hintergrund und verrichtet fortan als Daemon–Prozeß seinen Dienst. Andere Programme können den VME–Server–Daemon nun über das Netzwerk kontaktieren, um Zugriff auf den VMEbus zu erhalten.

Ausführliche Informationen zum Programm erhält man durch Angabe der Option ‚`-help`‘:

```
macsVMEd -help
```

Das Programm gibt in diesem Fall die folgenden Informationen auf dem Bildschirm aus, ohne sich anschließend in den Hintergrund zu schalten (weiter auf der nächsten Seite).

```
macsVMEd V1.0, R000509
MACS VME server daemon
This program is part of the H.E.S.S. Mirror Alignment Control System (MACS)
Usage:
    macsVMEd -help              to get this message
    macsVMEd -c                 to print out a default config file
    macsVMEd [option=value ...] to start normally
Possible options:
    ConfFile    'filename':             use configuration file <filename>
    LogFile     '' or '>stderr':        log to standard error
                'filename':             log to file <filename>
                '>syslog':              log to syslog mechanism
    LogLevel    0:                      log messages up to level FATAL
                1:                      log messages up to level ERROR
                2:                      log messages up to level WARNING
                3:                      log messages up to level CONTROL
                4:                      log messages up to level INFO
                5:                      log messages up to level DEBUG
Interacting with the running daemon:
    kill -HUP  <pid>    re-initialize VME server daemon (not working yet)
    kill -TERM <pid>    safely stop daemon (do not use kill -KILL)
Copyright (c) 2000 II. Institut fuer Experimentalphysik, Universitaet Hamburg
Contact: Rene Cornils <cornils@mail.desy.de>
```

Da sich der Prozeß nach Ausgabe der Informationen beendet, kann dies jederzeit erfolgen, ohne daß dabei die Gefahr besteht, eine eventuell bereits laufende Version des Programms zu stören.

Die Default–Konfiguration (vgl. Kap. 4.3.2.1) kann durch die Eingabe von

```
macsVMEd -c
```

auf der Kommandozeile erhalten werden, wobei sich das Programm nach der Ausgabe ebenfalls sofort beendet. Die Ausgabe gestaltet sich wie folgt:

```
{
    CTident = 00;
    ConfFile = /usr/local/macs/etc/macsVMEd.conf;
    LogFile = /usr/local/macs/log/macsVMEd.log;
    LogLevel = 4;
    MaxClients = 1;
}
```

Die einzelnen Optionen haben dabei die folgenden Bedeutungen:

CTident: Mit diesem Parameter wird festgelegt, unter welchem Namen der VME–Server seinen Dienst am Netzwerk anmeldet. Der Name setzt sich dabei aus `hessMacsVme` und dem Text dieser Option zusammen, wobei geplant ist, als Wert die jeweilige Nummer des Teleskops zu wählen, für das der entsprechende Prozeß zuständig ist.

ConfFile: Diese Option spezifiziert den Pfad für die Default–Konfigurationsdatei.

LogFile: Gibt an, wohin die Laufzeitinformationen geschrieben werden.

LogLevel: Hiermit wird festgelegt, ab welcher Priorität die Laufzeitinformationen ausgegeben werden.

MaxClients: Legt die maximale Anzahl an Clients fest, die den VME–Server–Daemon gleichzeitig kontaktieren dürfen. Zur Zeit ist nur jeweils ein Client vorgesehen. Als zukünftige Erweiterung ist jedoch denkbar, einen zentralen Prozeß zu haben, der als Client regelmäßig die VME–Server aller Teleskope zu Überprüfungszwecken kontaktiert.

Eine direkte Interaktion mit dem VME–Server–Daemon ist nicht möglich. Auf UNIX–basierten Systemen gibt es jedoch ein Verfahren, Nachrichten (sogenannte Signale, engl. *signals*) an laufende Prozesse zu schicken. Dies geschieht mittels des Systemkommandos

```
kill -SIGNAL PID
```

wobei `SIGNAL` durch das gewünschte Signal und `PID` durch die Prozeßnummer zu ersetzen sind. Die einzige zur Zeit implementierte Nachricht an den VME–Server–Daemon wird durch Senden des Signals `TERM` (für engl. *terminate*) vermittelt. Ein Erhalt dieses Signals veranlasst das Programm dazu, sich sauber zu beenden.

Für die Zukunft ist zudem vorgesehen, das Programm benachrichtigen zu können, sich neu zu initialisieren. So können dann z. B. Änderungen an der Konfigurationsdatei zur Geltung gebracht werden, ohne das Programm neu starten zu müssen.

Kapitel 5

Tests von Soft– und Hardwarekomponenten

Ein Schwerpunkt der vorliegenden Arbeit bestand darin, sämtliche Hardwarekomponenten zur Ansteuerung der Aktuatormotoren ausgiebigen Tests zu unterziehen. Besonderes Augenmerk wurde dabei auf die Eigenentwicklungen aus Hamburg (TEB–Hardware u. Aktuatorentwurf) und Heidelberg (Aktuatorentwurf) gelegt, denn bei jeder Neuentwicklung sind Fehler zu erwarten, die es zu bereinigen gilt. Auch muß das Hardwarekonzept seine grundsätzliche Leistungsfähigkeit im Einsatz erst unter Beweis stellen, ist doch allein vom Entwurf her nicht zwingend klar, daß die geforderte Funktionalität auch geliefert werden kann.

Die ersten Tests galten jedoch der Software, denn diese wurde schließlich dazu benutzt, alle Hardwarekomponenten zu testen. Wie für die Hardware, so gilt auch für die Software, daß das Konzept im Einsatz unter Beweis stellen muß, daß es die geforderte Leistung erbringen kann. Eine Fehlerbereinigungsphase ist ebenso unabdingbar.

Anschließend wurden Prototypen der Spiegelfacettenmechanik getestet. Untersucht wurden sowohl das Zusammenspiel mit der Steuerelektronik als auch die Zuverlässigkeit der Aktuatoren selbst.

Bereits vor den Tests wurde die Weiterentwicklung des Hamburger Entwurfs für die Spiegelfacettenmechanik aufgegeben, da eine Entscheidung aus finanziellen Gründen zugunsten des Heidelberger Entwurfs getroffen worden war. Über lange Zeit stand der Hamburger Gruppe jedoch kein Heidelberger Prototyp zur Verfügung, weshalb die ersten Tests mit Hamburger Prototypen vorgenommen worden sind. Später ist der Hamburger Gruppe ein Heidelberger Prototyp der Spiegelfacettenmechanik zur Verfügung gestellt worden, der daraufhin mit der nunmehr ausgereiften Software umfangreichen Tests unterzogen wurde.

Den Tests lagen folgende Zielsetzungen zugrunde:

1. Grundsätzlicher Funktionsbeweis des Softwarekonzeptes.
   Es sollte geprüft werden, ob das Softwarekonzept grundsätzlich dazu in der Lage ist, die an die Steuerung gestellten Anforderungen abbilden zu können. Dem

Konzept inhärente Unzulänglichkeiten, die eine Realisierung über diesen Weg in Frage gestellt hätten, sollten hier schnell zutage treten.

2. Erkennung und Behebung von Fehlern in den Quelltexten der Programme.
   Viele tausend Zeilen Quelltext waren geschrieben worden, die nun von Fehlern bereinigt werden sollten.

3. Erkennung und Behebung von Fehlern in der Steuerungshardware.
   Es galt, die Komponenten auf verschiedenen Ebenen zu untersuchen:

   (a) Test der Kommunikation zwischen dem *VME–Board* und der VME–CPU.
   Insbesondere weil das *VME–Board* eine Hamburger Eigenentwicklung darstellt, sollte überprüft werden, ob der Austausch von Befehlen und Daten über den VMEbus fehlerfrei funktioniert.

   (b) Test der Kommandierung der Branch Control Unit.
   Die *Branch Control Unit* verfügt über einen Satz von Befehlen zur Steuerung der Motoren. Die Tests sollten überprüfen, ob die einzelnen Befehle den gewünschten Effekt erzielen und ob der bestehende Satz an Befehlen ausreicht, die Steuerung realisieren zu können.

   (c) Test der Genauigkeit des Positionszählers und der Motorpositionierung.
   Die Zuverlässigkeit der gesamten Steuerungshardware ist im wesentlichen abhängig von der Genauigkeit und Zuverlässigkeit des Positionszählers, der die Flankenwechsel der Hallsensoren auswertet und akkumuliert. Dieser Mechanismus sollte getestet werden. Zudem ist die Steuerungshardware für das genaue Anfahren einer bestimmten Position der Motoren – und damit der Aktuatoren – zuständig. Auch dies galt es zu überprüfen.

4. Langzeitstudien über das Verhalten der Spiegelfacettenmechanik unter Dauerbelastung.
   Diese Tests sollten Unzulänglichkeiten und Fehler in den Entwürfen der Spiegelfacettenmechanik aufzeigen und Hinweise für Verbesserungen liefern.

Sämtliche Tests wurden in enger Zusammenarbeit mit den Mitarbeitern des TEB durchgeführt.

Des weiteren wurde ein Programm entwickelt, um die Verlötung der ca. achthundert Motoren mit den Relaisboxen für jedes der H·E·S·S·–Teleskope weitgehend automatisiert testen zu können.[1] Die Tests sollen vom Personal, das mit der Aufgabe der Verlötung betraut wird, selbständig durchgeführt werden, weshalb eine eigenständige Dokumentation zu diesem Testprogramm erstellt wurde. Diese findet sich in Anhang A.

[1] Mit der Verlötung der Motoren ist bis zum Zeitpunkt der Niederschrift dieser Arbeit nicht begonnen worden, so daß hier keine Ergebnisse vorgestellt werden können.

5.1 Test des VME–Server–Daemon

Wie bereits in Kapitel 4.3.4 (s. auch Abb. 4.9) beschrieben, ist der VME–Server–Daemon eine zentrale Komponente des Gesamtsystems. Dieses Programm soll anderen Programmen (den sog. Clients) den Zugriff auf den VMEbus – und damit die *Branch Control Unit* – ermöglichen.

Das Programm wurde entsprechend ausführlich getestet. Die Tests sollten in erster Linie zeigen, ob die eingesetzten Technologien tatsächlich eine adäquate Grundlage für die Realisierung bieten. Zudem galt es, den Entwurf daraufhin zu untersuchen, ob er die an das Programm gestellten Anforderungen abzubilden vermag. Nicht besonders hervorgehoben werden muß, daß ein weiterer Hauptzweck von Softwaretests darin besteht, Programmfehler aufzudecken und Hinweise zur Bereinigung zu liefern.

5.1.1 Transaktionsrate

Die erste Untersuchung des VME–Server–Daemons befaßte sich mit der Transaktionsrate, d. h. mit der Rate, mit der Kommandos von einem Client–Programm über das Netzwerk an den VME–Server übermittelt werden können. Selbstverständlich hatte der VME–Server die Kommandos dabei an die *Branch Control Unit* weiterzuleiten und eventuelle Rückgabewerte über das Netzwerk an den Client–Prozeß zurückzugeben.

Da die Steuerungselektronik dazu in der Lage ist, Aktuatorpositionen selbständig anzufahren, sind die Anforderungen an die Transaktionsrate für den Einsatz in Namibia moderat. Werte von einer Transaktion pro Sekunde dürften hierfür genügen. Allerdings sollte die Software auch für diverse Hardwaretests – wie z. B. Langzeittest der Spiegelfacettenmechanik, Qualitätstests der Motorverlötungen etc. – geeignet sein. Für diese Art von Tests ist eine Rate von etwa 20 Transaktionen pro Sekunde wünschenswert, so daß Motorpositionen zehntelsekundenweise protokolliert werden können.

Zur Ermittlung der Transaktionsrate wurde ein kleines Testprogramm entwickelt, das über das 10MBit Ethernet–Netzwerk des Instituts mit dem auf der VME–CPU ablaufendem VME–Server–Daemon kommuniziert. Dieses Programm schickt abwechselnd *Stop Motor* und *Read Position Counter* Kommandos an die *Branch Control Unit*; ersteres ist ein reiner Schreibbefehl, letzteres überträgt einen Wert (den des Positionszählers) zurück an das Testprogramm. Gemessen wurde jeweils die Zeit, die für 10 000 Transaktionen benötigt wurde.

Bei einer Testzeit von ca. 24 Stunden, was ein gutes Abbild unter verschiedenen Netzbelastungen geben sollte, ergaben sich dabei folgende Werte:

Transaktionen pro Sekunde
jeweils gemittelt über 10 000 Transaktionen

Minimum	Mittelwert	Maximum
116	320	370

Die erreichte Transaktionsrate ist also weit höher als gefordert und kann im übrigen als sehr gut im Vergleich zu anderen Verfahren für verteile Objekte gelten. Anzumerken ist, daß allein bei diesem Test 27 Millionen Transaktionen fehlerfrei abgearbeitet wurden.

Die Auskunft über Zeitverzögerungen für einzelne Transaktionen wurde auf anderem Wege gewonnen: Bei den Tests der Spiegelfacettenmechanik (s. Kap. 5.3 u. 5.4) wurde der Positionszähler etwa jede Zehntelsekunde ausgelesen und zusammen mit der aktuellen Zeit protokolliert. Dies erlaubte eine gezielte Analyse der Verzögerung einzelner Transaktionen. Abbildung 5.1 zeigt die Zeitverzögerungen für einen typischen

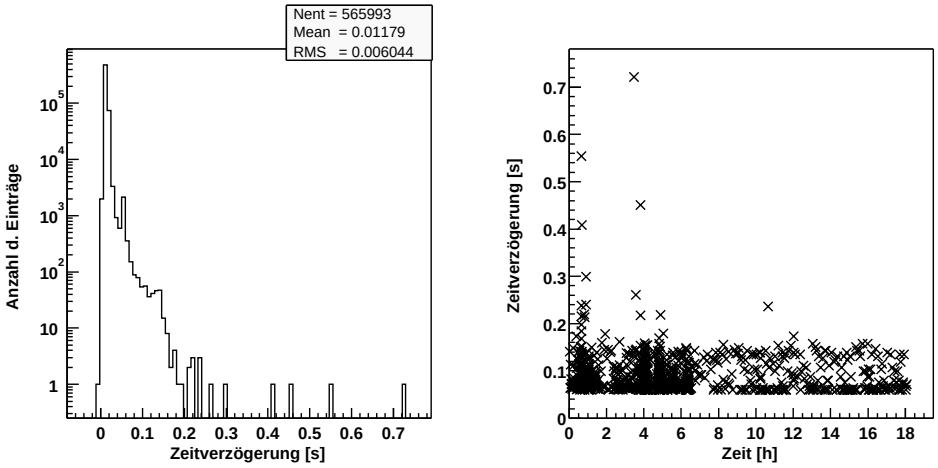


Abbildung 5.1: *Zeitverzögerung* (timeout) *für einen typischen Testlauf eines Aktuators. Links: Histogramm der Zeitverzögerung in logarithmischer Darstellung. Rechts: Zeitverzögerungen mit Werten über 0,05 Sekunden aufgetragen über die gesamte Testzeit.*

Testlauf eines Aktuators. Da für das Auslesen des Positionszählers drei Befehle (zwei Statusabfragen zusätzlich zur Abfrage des Positionszählers) über das Netzwerk an den VME–Server zu übertragen waren, liegt die Erwartung für die mittlere Verzögerung bei

$$\frac{3}{\langle \text{Transaktionsrate} \rangle} \quad = \quad \frac{3}{320}\,\text{s} \quad = \quad 0,0094\,\text{s} \tag{5.1}$$

Darin sind Zeiten für Berechnungen und Programmausgaben nicht berücksichtigt, so daß der gemessene Wert der mittleren Zeitverzögerung noch etwas höher ausfallen sollte. Die tatsächlich gemessene mittlere Zeitverzögerung von 0,012 Sekunden ist in guter Übereinstimmung mit diesem Wert.

Der dominierende Effekt für Zeitverzögerungen deutlich über dem Mittelwert ist die Belastung des Netzwerkes. Dies ist gut im rechten Diagramm von Abbildung 5.1 zu sehen, in dem die Zeitverzögerungen gegen die Testdauer aufgetragen sind[2]. Deutlich sind in einigen Bereichen Häufungen auszumachen, wohingegen in anderen Bereichen nur wenige Einträge erscheinen. Verzögerungen über etwa 0,15 Sekunden traten nur äußerst selten auf; der höchste Wert dieses Testlaufs betrug ca. 0,72 Sekunden, der höchste in allen Testläufen gemessene Wert lag bei etwa zwei Sekunden.

[2] Um nicht alle 500 000 Einträge durch Punkte darstellen zu müssen, wurden nur Verzögerungen über 0,05 Sekunden in das Diagramm aufgenommen. Die weggelassenen Einträge hätten den Bereich unterhalb dieser Schwelle vollständig geschwärzt.

Mit verschiedenen Client–Programmen kam es in den zusammengenommen sehr umfangreichen Tests nie zu einem sogenannten *Timeout* Ausnahmefehler (engl. *timeout exception*), der auftritt, wenn die Quittierung auf eine Anfrage eines Client–Programms an den VME–Server–Daemon für eine gewisse Zeit ausbleibt.

5.1.2 Speicherlecks

Wird ein Programm als sogenannter *Daemon* (vgl. Kap. 4.3.2.2) konzipiert, ist besonders auf die Freigabe von nicht mehr benötigtem Speicher zu achten. Solche Programme laufen komplett im Hintergrund (d. h. ohne direkte Interaktion mit Benutzern) und werden in der Regel nur zu Wartungs– und Reinitialisierungszwecken beendet. Nicht freigegebener Speicher akkumuliert sich also und kann über lange Zeiträume beträchtliche Ausmaße erreichen. In Extremfällen kann dies zum Absturz des Systems führen, wenn der Systemkernel nicht dazu in der Lage ist, diesen Prozeß zu beenden und aus dem Speicher zu entfernen. Dieses Phänomen des sich akkumulierenden nicht freigegebenen Speichers wird als Speicherleck (engl. *memory leak*) bezeichnet.

Der VME–Server–Daemon wurde auf Speicherlecks hin untersucht. Dazu gibt es Systemkommandos wie z. B. *top*, die den von Programmen belegten Speicher anzeigen. Mittels eines Test–Clients wurde nun der VME–Server mit vielen Millionen Anfragen belegt, wobei erneute Kontaktierungen durch immer wiederkehrende Verbindungsabbrüche erzwungen worden sind.

Tatsächlich fand sich ein Speicherleck in der ersten Version des Programms. Nachdem dieser Fehler jedoch bereinigt wurde, ist in dem nunmehr mehrmonatigen Betrieb kein weiteres Speicherproblem zutage getreten.

5.1.3 Zuverlässigkeit

Nach einer sehr kurzen Phase der Fehlerbereinigung wurde die Software dazu benutzt, die Hardwarekomponenten und andere Softwareteile zu testen.

Im Verlauf der mehrmonatigen Tests von Soft– und Hardware hatte der VME–Server–Daemon viele hundert Millionen über das Netzwerk vermittelte Kommandos abzuarbeiten, wobei eine Beendigung des Programms nur selten (z. B. wegen Ein– bzw. Ausbau von Hardwarekomponenten, Einspielungen neuer Funktionsbibliotheken etc.) stattfand. Dennoch traten während aller Tests keinerlei Probleme auf, die sich in Programmabstürzen, Fehlkommandierungen der *Branch Control Unit*, Speicherlecks oder Fehlverhalten in der Kommunikation mit den Client–Programmen geäußert hätten.

Nach Abschluß aller vorgestellten Tests darf davon ausgegangen werden, daß der VME–Server–Daemon für den Dauereinsatz in Namibia bereit ist. Eventuell noch zutage tretende Fehler sollten eher unbedeutender Natur sein und schnell behoben werden können.

5.2 Test der Hardware zur Ansteuerung der Aktuatormotoren

Die Hardware zur direkten Ansteuerung der Aktuatormotoren (vgl. Kap. 4.1.1) ist eine Eigenentwicklung der Universität Hamburg, die speziell für diesen Zweck konzipiert wurde. Dies macht es unumgänglich, die gesamte Hardware ausgiebig zu testen, da sich bei Neuentwicklungen Fehler nie vollständig vermeiden lassen.

Die Tests sollten dabei zweierlei sicherstellen:

1. das korrekte Funktionieren der Elektronik sowie
2. die Abbildbarkeit der gewünschten Funktionalität.

5.2.1 VME–Board

Die *Branch Control Unit* verfügt über einen Satz von Befehlen, mittels derer die Steuerungssoftware die *Branch Control Unit* kommandieren kann. Für die Übermittlung der Kommandos an die *Branch Control Unit* ist das *VME–Board* zuständig, das über den VMEbus mit dem Steuerungscomputer (VME–CPU) verbunden ist. Es muß nicht besonders betont werden, daß eine fehlerfreie und reibungslose Kommunikation über den VMEbus gewährleistet sein muß.

Es stellte sich bei einigen Tests jedoch heraus, daß unter gewissen Umständen einige Bits nicht korrekt von dem *VME–Board* an den Steuerungsrechner übermittelt wurden. Der Fehler trat nur bei Lesebefehlen auf, bei Kommandos also, die Daten vom *VME–Board* an den Steuerungscomputer übertragen. Dieser Fehlkommunikation wurde auf den Grund gegangen. Wie sich zeigte, mußte dazu der Datenübertragungsprozeß auf unterer Ebene (d. h. Signalisierung auf dem VMEbus) untersucht werden. Nachfolgende Darlegungen der Protokolle folgen dem VMEbus Standard [VME 93].

In Abbildung 5.2 ist der Handshaking–Mechanismus[3] für die Datenauslese dargestellt. Der die Daten lesende Master ist in diesem Fall der Steuerungsrechner, der der die Daten sendende Slave das *VME–Board*. Der Master teilt dem Slave über den Wegfall des Signals `DTACK*` (*data transfer acknowledge*) mit, daß dieser Datenübermittlung einzuleiten hat. Die Spezifikation fordert, daß die Datenleitungen 25 ns nach diesem Signal mit gültigen Werten belegt sind. Durch Hochsetzen des Signals `DS0*` (*data strobe*) signalisiert der Master dem Slave schließlich, daß das Auslesen der Datenleitungen abgeschlossen ist. Im vorliegenden Fall benötigt der Master 17,5 ns zur reinen Auslese der Datenleitungen.

Die Verläufe der relevanten Signale bei der fehlerhaften Datenübertragung sind in Abbildung 5.3 aufgezeigt. Der dargestellte Bereich ist mit dem in Abbildung 5.2 identisch. Zwar ist der Verlauf des Signals `DTACK*` noch enthalten, der Verlauf des Signals `DS0*` wurde der Übersichtlichkeit halber jedoch weggelassen. Der tatsächliche Auslesevorgang der Datenleitungen wird aber durch die weiterhin angegebenen vertikalen Linien begrenzt.

[3] Signalisierungsprotokoll zur Sicherstellung einer asynchronen Datenübertragung

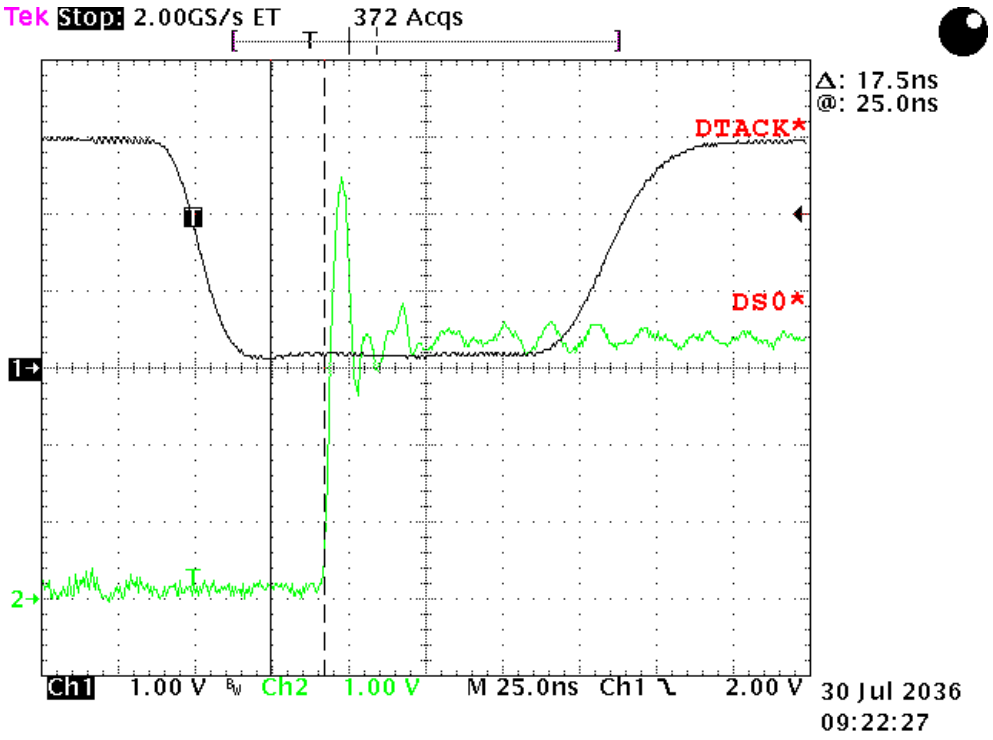


Abbildung 5.2: *Handshaking–Mechanismus der Datenauslese über den VMEbus. Dargestellt ist eine Oszilloskopaufnahme, die das Protokoll der Datenauslese verdeutlicht. Die Zeitauflösung der Aufnahme beträgt 25 ns pro horizontaler Unterteilung (Information:* `M 25.0ns`*, unten im Bild), die vertikale Auflösung beträgt 1 V pro Unterteilung (Information:* `Ch1 1.00 V` *und* `Ch2 1.00 V`*, unten) für die beiden Kanäle. Auf Kanal 1 (*`1→`*, Signalverlauf oben) ist der Verlauf des Signals* `DTACK*`*, auf Kanal 2 (*`2→`*, Signalverlauf unten) der Verlauf des Signals* `DS0*` *wiedergegeben. Der Trigger (Position bei Symbol* `T`*, invers dargestellt) ist auf das Unterschreiten von 2 V des Signals* `DTACK*` *eingestellt (Information:* `Ch1 ↘ 2.00 V`*, rechts unten), womit nach der VMEbus Spezifikation der Master dem Slave signalisiert, die Datenauslese einzuleiten. 25 ns (Information:* `@: 25.0ns`*, rechts oben) nach diesem Signal müssen die Datenleitungen vom Slave mit gültigen Werten belegt sein (Position bei durchgezogener vertikaler Linie). Mit dem Hochsetzen des Signals* `DS0*` *(Position bei gestrichelter vertikaler Linie) signalisiert der Master dem Slave, daß er die Daten vollständig gelesen hat. Dies dauert im vorliegenden Fall 17,5 ns (Information:* `Δ: 17.5ns`*, rechts oben). Während dieser Zeit, also zwischen den beiden vertikalen Linien, müssen die Datenleitungen stabil mit gültigen Niveaus belegt sein.*

Deutlich ist zu sehen, daß das Signal der Datenleitung `D03` zwischen den beiden vertikalen Linien einbricht. Die eigentlich vom Slave zu übermittelnde logische `Eins` wird somit vom Master als logische `Null` interpretiert. Dies konnte auf den Umstand

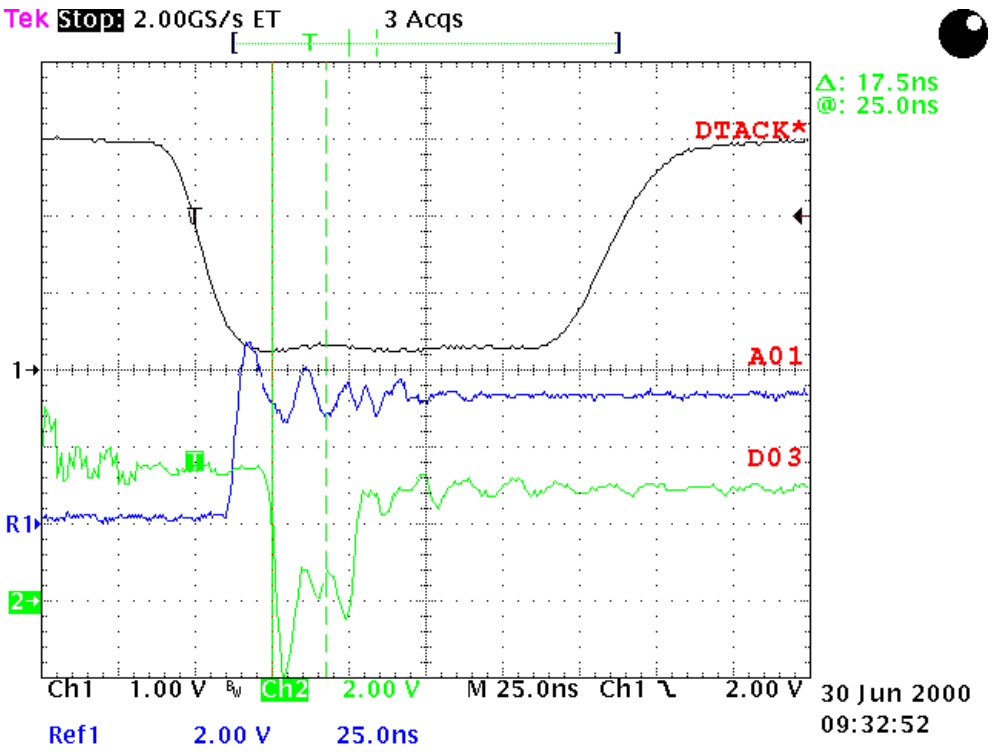


Abbildung 5.3: *Adressierung mit fehlerhafter Datenübertragung über den VMEbus. Dargestellt ist eine Oszilloskopaufnahme, die den Handshaking–Mechanismus im Zusammenspiel mit einem Adress– und einem Datensignal verdeutlicht.*
Die horizontale Zeitauflösung, die Auflösung des weiterhin mit dem Signal `DTACK*` *belegten Kanals 1 und die Bedingung für den Trigger entsprechen Abb. 5.2. Der Kanal 2 (*`2→`*, Verlauf: mittleres Signal links, unteres Signal rechts), mit einer Auflösung von 2 V pro vertikaler Unterteilung (Information:* `Ch2 2.00 V`*, unten im Bild), ist mit dem Signal der Datenleitung* `D03` *belegt. Das Referenzsignal 1 (*`R1→`*, Verlauf: unteres Signal links, mittleres Signal rechts), ebenfalls mit einer Auflösung von 2 V pro vertikaler Unterteilung (Information:* `Ref1 2.00V`*, links unten), ist mit dem Signal der Adressleitung* `A01` *belegt.*
Kurz vor der tatsächlichen Datenauslese (durchgezogene vertikale Linie) bricht das eine logische `Eins` *repräsentierende Datensignal* `D03` *ein und behält diesen Zustand bis weit nach Abschluß der Auslese (gestrichelte vertikale Linie). Kurz zuvor ist als Ursache dafür das Ungültigwerden der Adressleitung* `A01` *zu verzeichnen, was jedoch im Rahmen der VMEbus Spezifikation liegt.*

zurückgeführt werden, daß die vom Master angelegte Adressleitung `A01` kurz nach dem Wegfall von `DTACK*` ihren gültigen Wert von logisch `Null` verliert und fortan eine logische `Eins` repräsentiert. Dies bewegt sich jedoch im zulässigen Rahmen der Spezifikation des VMEbus. Hierin wird nur eine Gültigkeit der Adressleitungen bis zum Wegfall des

Signals `DTACK*` gefordert; während der tatsächlichen Datenauslese dürfen die Adressleitungen also undefiniert sein.

Es ist daher für den Slave unumgänglich, die Information der Adressleitungen nach deren Gültigkeit auf dem VMEbus für die eigentliche Datenübermittlung zwischenzuspeichern. Dies wurde in der ersten Version des *VME–Boards* versäumt, da bei den bisher vom TEB verwendeten VME–CPUs die Adressleitungen – kulanterweise – bis zum Abschluß der Datenauslese ihre korrekten Werte behielten. Die im Rahmen des H·E·S·S·–Projektes zum Einsatz kommende VME–CPU legt dagegen die Zeiten für die Signalgültigkeiten recht eng aus, ohne sich jedoch außerhalb der Spezifikation zu bewegen.

Die Behebung dieses Problems erwies sich als relativ einfach; es mußte lediglich eine entsprechend modifizierte Programmlogik in den Kontrollbaustein des *VME–Boards* eingespielt werden.

Nachdem die Entwickler der Steuerungshardware eine Zwischenspeicherung der Signale der Adressleitungen implementiert hatten, konnten keine weiteren Kommunikationsprobleme festgestellt werden.

5.2.2 Kommandostruktur

Der Satz an Befehlen, mittels der die *Branch Control Unit* beauftragt werden kann, legt fest, was mit einer Steuerung zu erreichen möglich ist. Eine detaillierte Übersicht über die Kommandostruktur mit einer Definition aller weiter unten verwendeten Bezeichnungen findet sich in [TEB 00].

Zu untersuchen war, ob dieser Satz an Befehlen, den eine Softwaresteuerung zur Kommandierung der *Branch Control Unit* nutzen kann, das gewünschte Verhalten der Motoren abzubilden vermag, d. h. ob der Befehlssatz in diesem Sinne vollständig war.

Tatsächlich stellte sich anhand vieler Tests heraus, daß die Kommandostruktur in drei Punkten unvollkommen war. In Absprache mit den Entwicklern wurde der Befehlssatz erweitert bzw. modifiziert, um diesen Unzulänglichkeiten Rechnung zu tragen. Im einzelnen betraf dies folgende Befehle:

1. Prüfung, ob ein Knoten tatsächlich mit Motoren (d. h. einer Spiegelfacette) bestückt ist.
   Mit der ursprünglichen Form der Kommandierung konnte die Prüfung, ob an einem Knoten Motoren angeschlossen sind, nur dadurch erfolgen, die Motoren jeweils zu selektieren und anschließend zu starten. Bei diesem Verfahren werden angeschlossene Motoren jedoch bewegt und müssen nach der Überprüfung an ihre ursprüngliche Position zurückgefahren werden. Dies wäre eine unangemessen aufwendige Prozedur für einen solchen einfachen aber wichtigen Test. Das Kommando *Write Motor Selection Register* wurde deshalb so erweitert, daß bei der Selektion eines Motors sogleich überprüft wird, ob dieser Motor auch angeschlossen ist. Dies erfolgt über das Schalten des entsprechenden Relais in der Relaisbox. Anschließend kann das Ergebnis über den Zustand des Bits *SEL* (für *selected*) erfahren werden, welches mit dem Befehl *Read Status Register* abgefragt werden

kann.
Mit dieser Modifikation ist es nun möglich, durch sukzessives Selektieren aller Motoren festzustellen, welche Knoten nicht bestückt sind bzw. welche Kabelversorgungen einen Defekt aufweisen. Die Motoren müssen dabei nicht gefahren werden.

2. Einstellbarkeit der Kriechgeschwindigkeit.
Die *Branch Control Unit* verfügt über einen Modus, den angewählten Motor über eine genau festlegbare Anzahl von Zählerinkrementen zu fahren (vgl. Kap. 4.1.1.4). Dieser sogenannte *Preset Mode* verlegt das zeitkritische Kontrollieren der aktuellen Motorposition von der Softwaresteuerung in die *Branch Control Unit*, die darüber wacht, den Motor rechtzeitig abzuschalten, um die Zielposition möglichst genau zu erreichen. Mittels des Befehls *Write Speed Register* kann der *Branch Control Unit* mitgeteilt werden, mit welcher Geschwindigkeit (*Motor Rotation Speed*, Werte von 0 bis 7) die Motoren zu fahren sind. Wird ein Motor allerdings im *Preset Mode* gefahren, wird die Motorgeschwindigkeit kurz vor Erreichen der Zielposition verringert, um den Motor möglichst genau stoppen zu können. In der ersten Version der Kommandostruktur konnte diese sogenannte Kriechgeschwindigkeit (*Motor Creep Speed*) nicht verändert werden. Es erwies sich jedoch als schwierig, einen für alle Fälle optimalen Wert zu finden. Wird die Kriechgeschwindigkeit zu hoch gewählt, so fahren die Motoren weit über die Zielposition hinaus; ist sie zu niedrig, kann dies dazu führen, daß der Motor in schwergängigen Bereichen (z. B. kurz vor den Anschlägen) aufgrund der Motorabschaltung (vgl. Kap. 4.1.1.3) zu früh anhält. Der Befehl *Write Speed Register* wurde deshalb so erweitert, daß die Kriechgeschwindigkeit zusätzlich zur normalen Fahrgeschwindigkeit der Motoren gesetzt werden kann. Entsprechend wurde der Befehl *Read Speed Register* modifiziert, um die in der *Branch Control Unit* eingestellte Kriechgeschwindigkeit auslesen zu können.
Mit diesen Modifikationen der Befehle zum Setzen und Auslesen der Geschwindigkeiten ist es nun möglich, die Kriechgeschwindigkeit den jeweiligen Bedingungen dynamisch anzupassen.

3. Löschen des Fehlerstatus ohne Rücksetzen der *Branch Control Unit*.
Bleibt ein Motor beim Fahren stehen, wird in der *Branch Control Unit* das Statusbit *NOM* (für *no motion*) gesetzt, welches über den Befehl *Read Status Register* abgefragt werden kann. Ein erneutes Fahren eines Motors ist nun erst nach Löschen dieses Statusbits möglich. In der ersten Version der Kommandostruktur war dies nur über ein generelles Zurücksetzen der *Branch Control Unit* zu erreichen. Zusätzlich zum Löschen der meisten Statusbits[4], deselektiert der Befehl *General Clear* allerdings auch einen eventuell selektierten Motor. Dabei werden alle Spannungen vom Motor genommen, insbesondere auch die Versorgungsspannung der Hallsensoren. Unweigerlich bekommt dabei der Positionszähler unvorhersehbare Flankenwechsel auf seine Signaleingänge; der Wert stimmt nicht mehr mit

[4] einige der Statusbits sind hardwarebestimmt und können nicht gelöscht bzw. zurückgesetzt werden

der Motorposition überein. Da das Stehenbleiben eines Motors keinesfalls immer einen Fehler bedeutet[5], ist es unbefriedigend, vor einer Weiterfahrt die *Branch Control Unit* komplett zurücksetzen zu müssen. Wie beschrieben wird dabei der Positionszähler nämlich ungültig und kann für die Positionsbestimmung des Motors nicht mehr verwendet werden. Um diesen Umstand zu beheben, wurde der Befehlssatz um ein Kommando zum alleinigen Zurücksetzen der Statusbits erweitert. Im Gegensatz zu *General Clear* ändert der Befehl *Clear Error Flags* nicht den Selektionsstatus der Motoren. Ein eventuell selektierter Motor bleibt also selektiert (d. h. mit allen Spannungen versorgt), womit der Positionszähler seine Gültigkeit behält.
Erst diese Befehlssatzerweiterung ermöglichte die in den Kapiteln 5.3 und 5.4 vorgestellten genauen Tests der Aktuatoren ohne aufwendige Manipulation des Positionszählers nach jeder Motorabschaltung an den Anschlägen.

Nach diesen Erweiterungen bzw. Modifikationen der Kommandostruktur gab es bei allen weiteren durchgeführten Tests keine Veranlassung, noch etwas zu ändern. Alle von den Programmen auszuführenden Aufgaben konnten mit dem bestehenden Befehlssatz abgebildet werden.

5.2.3 Kommandierung

Die Kommunikation mit dem *VME–Board* findet mittels 16 Bit breiter Werte (sogenannte *Words* oder Worte) statt. Schreibbefehle senden solch einen Wert an das Board, Lesebefehle empfangen einen Wert. Die einzelnen Bits des zu übertragenden Wertes sind nun mit – für jeden Befehl anderen – Bedeutungen belegt; sie kodieren eine bestimmte Information.

Getestet wurde mittels einiger Testprogramme, ob die per Schreibbefehl an das *VME–Board* übertragenen Bits korrekt interpretiert und ob die per Lesebefehl vom *VME–Board* übertragenen Bits korrekt gesetzt wurden.

In dieser sogenannten Kommandierung wurden einige kleine Fehler entdeckt, wobei sich alle in eine der folgenden Kategorien einordnen ließen:

- Nichtübertragung einzelner Bits,
- Negierung der Bedeutung einzelner Bits,
- Vertauschungen zweier Bits oder
- Unklarheiten in der Dokumentation.

Da keiner dieser Fehler schwerwiegend war und alle innerhalb kürzester Zeit behoben wurden, seien hier nur zwei beispielhaft aufgelistet.

1. Kommando *Read Position Counter*
   Das Vorzeichen des Positionszählers wird in Bit 14 des übertragenen Registers

[5] an den Anschlägen ist dies sogar ein gewünschter Effekt

kodiert. Hat das Bit den Wert 0, ist der Zählerwert positiv, hat es den Wert 1, ist der Zählerwert negativ. Der Zählerwert selber belegt die Bits 00 bis 12.
Die Wertigkeit des Vorzeichen–Bits wurde nun anfangs falsch gesetzt, so daß das Vorzeichen des Zählerwertes immer vertauscht war.

2. Kommando *Write Interval Register*
Der zu setzende Wert des Intervall-Registers wird in den Bits 00 bis 09 kodiert. Allerdings beeinflussen die höherwertigen Bits den zu übertragenden Wert; sie müssen sämtlich mit dem Wert 0 belegt sein, damit das Intervallregister korrekt gesetzt wird.
Dies war in der ersten Version der Dokumentation nicht explizit gefordert.

5.2.4 Positionszähler

Wie in Kapitel 4.1.1.1 dargestellt, entspricht ein Zählerinkrement des Positionszählers einem Flankenwechsel eines der Hallsensoren der Motoren. Der Kürze halber wird als Einheit dafür im folgenden der – zugegebenermaßen in diesem Kontext etwas unschöne – aus dem Englischen stammende Begriff *Count* verwendet.

Die korrekte Verarbeitung der Signale der Hallsensoren ist Voraussetzung für eine genaue Positionierung der Aktuatoren und damit der Spiegelfacetten. Dies ist nicht trivial, denn die Signale müssen über lange Kabelwege von den Motoren zu den *Branch Driver Boards* übertragen werden, wobei stets vorhandene Störspannungen für eine Verfälschung der Signale sorgen können. Wie bereits in Kapitel 4.1.1.1 erwähnt kam aus diesem Grund ein Elektronikbaustein der Firma Agilent zum Einsatz, der speziell für diese Art Anwendung ausgelegt ist. Damit ist jedoch nicht ausgeschlossen, daß es zu Falschzählungen kommt; es galt also, die Steuerungshardware auf Zählerungenauigkeiten hin zu untersuchen.

Zu diesem Zweck wurde ein Motor mit der in Abbildung 5.4 dargestellten Meßvorrichtung versehen. Eine mit einer Markierungsnut versehene Kreisscheibe wurde an der Antriebsschnecke eines Motors angebracht, um die Bewegung des Motors bequem ablesen zu können. Sie hatte einen Umfang von 220 mm, womit die Strecke in Millimetern am Rand der Kreisscheibe genau den Counts des Positionszählers entsprach, da eine Umdrehung des Motors 220 Flankenwechsel der Hallsensoren liefert. Am nicht mitrotierenden Motorgehäuse wurde ein kleiner Metallblock mit einer Nut befestigt, die als Referenzmarkierung diente.

Ein zur Untersuchung der Ungenauigkeit der Positionszählung entwickeltes Programm fuhr den so präparierten Motor nun eine per Pseudozufallsverfahren[6] ermittelte Anzahl[7] von Counts in eine – ebenfalls pseudozufällig – bestimmte Richtung. Nach je-

[6] Die meisten Verfahren zur rechnergestützten Erzeugung gleichverteilter Zufallszahlen beruhen auf zahlentheoretischen Betrachtungsweisen. Die in der Regel rekursiven Algorithmen sind deterministischer Natur, weshalb die einzelnen Zahlen einer Reihe zwar (mehr oder weniger) gleichverteilt, jedoch nicht im eigentlichen Sinne zufällig sind. Zudem können Rechner Zahlen (bislang) nur mit endlicher Genauigkeit darstellen. Aufgrund dieser Umstände spricht man bei so erzeugten Zahlen von *Pseudozufallszahlen*.

[7] zwischen 1 und 999

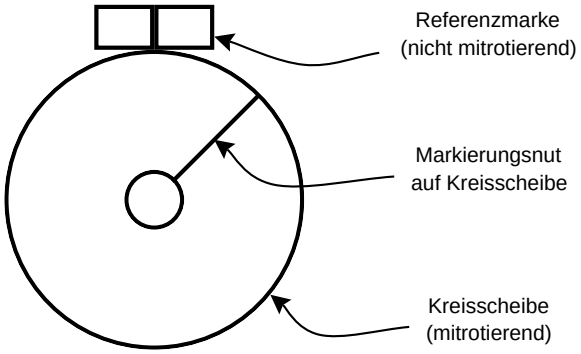


Abbildung 5.4: *Meßvorrichtung zur Bestimmung der Ungenauigkeit der Positionszählung. Die Kreisscheibe mit der Markierungsnut wurde mit der Antriebsschnecke des Motors verbunden, so daß die Bewegung des Motors bequem abzulesen war. Sie hatte einen Umfang von 220 mm, was genau der Anzahl der Counts pro voller Umdrehung des Motors entsprach. Somit konnte die Deplazierung des Motors in Counts durch Ablesen der Deplazierung der Markierungsnut in Millimetern bestimmt werden. Am nicht mitrotierenden Motorgehäuse war ein kleiner Metallblock mit einer Nut montiert, die als Referenzmarke fungierte.*

weils eintausend dieser Fahrten fuhr das Programm den Motor zur Ausgangsposition zurück und legte eine kleine Pause ein, um dem Operateur die Möglichkeit zu geben, die Deplazierung zu bestimmen. Die angefahrene Ausgangsposition wurde ausschließlich durch den Positionszähler bestimmt, wohingegen die Markierungsnut die tatsächliche Bewegung des Motors wiedergab. Verzählungen hätten also eine Deplazierung gegenüber der Referenzmarke zur Folge gehabt.[8]

Dieser Vorgang wurde fünfzigmal wiederholt, so daß der Motor insgesamt 50 000 Fahrten zu absolvieren hatte. Zu beachten ist, daß dieses Verfahren eine Deplazierung nur modulo 220 festzulegen vermag, da ganze Drehungen des Motors die Position der Markierungsnut relativ zur Referenzmarke nicht verändern.

Während des gesamten Tests summierten sich die gezählten Counts auf fast 25 Millionen[9]. Eine Abweichung von der Referenzposition konnte dabei nicht festgestellt wer-

[8] Streng genommen ist dies so nicht richtig, denn Verzählungen können sich kompensieren. Daher handelt es sich hier vielmehr um eine Art diskreten *random walk* mit unterschiedlichen mittleren Schrittlängen in negativer und positiver Richtung, da *a priori* nicht klar ist, daß Unter– bzw. Überzählungen gleich häufig auftreten. Erschwerend kommt hinzu, daß Störsignale auch dann zu Falschzählungen führen können, wenn der Motor nicht bewegt wird. Aus der Deplazierung der Markierungsnut gegenüber der Referenzmarke ließe sich also nur ein Höchstwert für die Ungenauigkeit des Positionszählers bei einem bestimmten Konfidenzniveau ermitteln. Dazu wäre aber eine Modellierung des Branchkabels zusammen mit dem Zählerbaustein notwendig, was allerdings einen unverhältnismäßig hohen Aufwand für diesen einen Test bedeutet hätte. Auf eine Ableitung dieser Beziehung wurde deshalb verzichtet, denn der exakte Wert der Ungenauigkeit ist unbedeutend für diesen Test.

[9] genau 24. 421. 464

den. Von einer korrekten Berechnung der Ungenauigkeit wurde hier abgesehen, es ist jedoch evident, daß sie vernachlässigbar gering ist.

5.2.5 Motorpositionierung

Im *Preset Mode* fährt die *Branch Control Unit* den angewählten Motor eine per *Interval Register* festzulegende Anzahl von Counts. Da die verwendeten Motoren keine Schrittmotoren sind, muß die Motorspannung dazu so geschickt abgeschaltet werden, daß der Motor weder deutlich vor der Zielposition stehenbleibt, noch weit über diese hinausfährt (vgl. Kap. 4.1.1.4).

Das Programm zur Untersuchung des Positionszählers (vgl. Kap. 5.2.4) wurde zugleich zum Test der Motorpositionierung im *Preset Mode* der *Branch Control Unit* verwendet. Der Motor war dabei nur mit der Kreisscheibe zur Untersuchung des Positionszählers bestückt, hatte also weit weniger zu leisten, als wenn er an einem Aktuator montiert wäre. Dieser nahezu freilaufende Motor sollte somit dazu neigen, die Zielposition deutlich zu überfahren, da die natürlichen Bremswirkungen durch die Reibung in der Spindelführung und die Federkräfte in den Anschlagbereichen fehlten. Im Gegenzug war allerdings zu erwarten, daß ein Anhalten vor der Zielposition aufgrund der fehlenden Bremswirkungen kaum auftreten sollte.[10]

Das Testprogramm verzichtete auf eine dynamisch an die Verhältnisse angepaßte Kriechgeschwindigkeit; sie wurde fest auf den Wert 3 (möglicher Wertebereich 0 bis 7) eingestellt. Die im folgenden dargelegten Ergebnisse sind deshalb sicherlich schlechter als mit einer entsprechenden Anpassung der Kriechgeschwindigkeit möglich wäre. Die Normalgeschwindigkeit des Motors wurde ebenfalls fest gewählt, und zwar auf den Wert 6.

Abbildung 5.5 zeigt die Ergebnisse des Tests zur Untersuchung der Motorpositionierung. Die mittlere Abweichung von der Zielposition für die jeweils erste Anfahrt betrug etwa fünf Counts (s. Abb. 5.5, links), wobei die maximale Abweichung bei sechs Counts lag. Dabei stoppte der Motor niemals vor der Zielposition sondern erreichte sie entweder genau oder fuhr darüber hinaus. Dies entsprach der Erwartung; im Falle eines an einem Aktuator montierten Motors würden sich die Abweichungen wegen der zusätzlichen Reibungskräfte hin zu kleineren – aufgrund der Motorabschaltung mitunter negativen – Werten verschieben.

Die Zahl der benötigten Versuche zur Erreichung der Zielposition betrug im Mittel ca. drei Fahrten (d. h. zwei nötige Iterationen). Für eine nicht weiter optimierte Kriechgeschwindigkeit ist das ein sehr guter Wert, der auch in dieser Form vollkommen genügen würde. Allerdings müßte die Kriechgeschwindigkeit zumindest so weit anpaßbar sein, daß diese bei einem Halt vor der Zielposition zu erhöhen wäre. Das zu diesem Test verwendete Programm hatte eine solche Logik allerdings nicht implementiert.

Die Ergebnisse dieses Tests haben gezeigt, daß die Motorpositionierung der *Branch*

[10] Ein solcher Halt vor der Zielposition tritt in der Regel nicht durch eine zu frühe Motorabschaltung, sondern durch eine zu geringe Motorgeschwindigkeit auf. Der Motor schafft es in diesem Fall nicht, Reibungs– und Federkräfte zu überwinden.

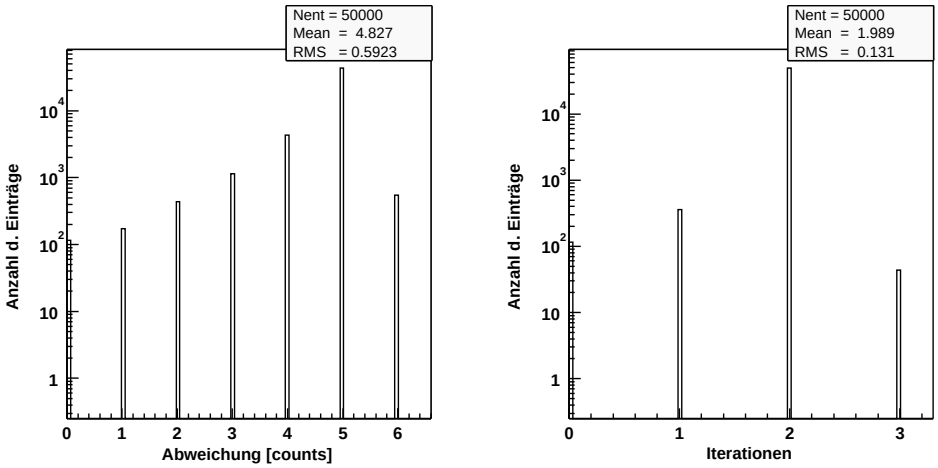


Abbildung 5.5: *Genauigkeit der Motorpositionierung. Links ist das Histogramm der Abweichung von der Zielposition bei der jeweils ersten Anfahrt dargestellt. Eine positive Abweichung bedeutet dabei, daß der Motor die entsprechende Anzahl von Counts über die Zielposition hinausgefahren ist. Rechts ist das Histogramm für die zur Erreichung der Zielposition nötigen Iterationen (sukzessive Anfahrtversuche) abgebildet.*

Control Unit im *Preset Mode* das gewünschte Verhalten abzubilden vermag. Es darf davon ausgegangen werden, daß sich für eine dynamisch an die Verhältnisse angepaßte Kriechgeschwindigkeit die Zahl der im Mittel zur Erreichung der Zielposition nötigen Iteration noch verringern ließe.

5.3 Test einiger Hamburger Prototypen der Spiegelfacettenmechanik

Nachdem in ausgiebigen Tests keine weiteren Fehler sowohl in der Soft– als auch in der Hardware gefunden werden konnten, wurden einige der verfügbaren Prototypen der Spiegelfacettenmechanik Langzeittests unterzogen.

Da der Hamburger Gruppe zu diesem Zeitpunkt kein Prototyp des Heidelberger Entwurfes zur Verfügung stand, wurden die ersten Tests ausschließlich mit Hamburger Prototypen durchgeführt.

Für die Tests wurde jeweils ein Aktuator eintausendmal mit einer festgelegten Geschwindigkeit vom unteren Anschlag in den oberen und zurück gefahren. Während der gesamten Testdauer wurde dabei die aktuelle Position des Motors zusammen mit der verstrichenen Zeit ca. zehnmal pro Sekunde protokolliert. Abhängig von der gewählten Geschwindigkeit und vom Typ des untersuchten Aktuators dauerten diese Tests zwischen vierzehn und zwanzig Stunden pro Aktuator.

Für die Positionsbestimmung wurde der Positionszähler der *Branch Control Unit* ausgelesen und ausgegeben, wobei der Nullpunkt willkürlich gewählt wurde. Die Er-

gebnisse werden im folgenden anhand einiger Diagramme exemplarisch für einen der getesteten Aktuatoren vorgestellt (Prototyp Nr. 59, Motor Nr. 1).

5.3.1 Reproduzierbarkeit der Anschlagpositionen

Da das Anhalten der Motoren über das Ausbleiben von Flankenwechseln der Hallsensoren an der Motorwelle innerhalb vorgegebener – von der gewählten Geschwindigkeit abhängiger – Zeitintervalle bestimmt wird, gibt es keinen definierten Anschlag (vgl. Kap. 4.1.1.3). Es ist jedoch unabdingbar, daß die tatsächlichen Halteposition reproduzierbar innerhalb eines vernünftigen Bereiches liegen, da andernfalls der verfahrbare Bereich zu stark eingeschränkt werden müsste. Deshalb ist die Reproduzierbarkeit der Endpositionen ein wichtiger Untersuchungsgegenstand.

Abbildung 5.6 zeigt die Ergebnisse für die Endpositionen bei eingefahrener Spindel (unterer Anschlag). Mit einer Bandbreite von fünf Counts und einer Standardabwei-

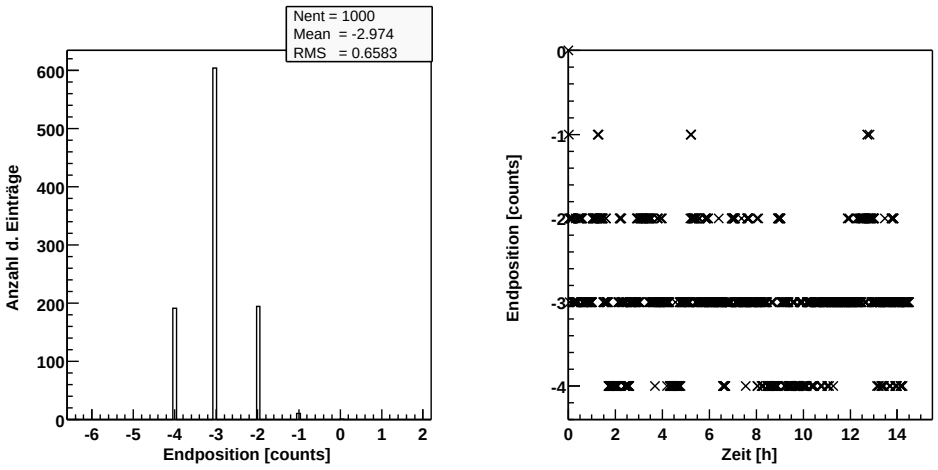


Abbildung 5.6: *Reproduzierbarkeit des Anschlags bei eingefahrener Spindel (unterer Anschlag). Links: Histogramm der unteren Endposition über eintausend Anfahrten, wobei der Nullpunkt der Position willkürlich gewählt wurde. Rechts: Verteilung der unteren Endpositionen über die gesamte Testzeit zur Veranschaulichung des Langzeitverhaltens.*

chung (RMS) von 0,66 Counts ist der gezeigte Aktuator der – in diesem Sinne – Beste von allen untersuchten Aktuatoren. Im Vergleich zum gesamten verfahrbaren Bereich von ca. 5800 Counts (vgl. Abb. 5.10) ist diese Unsicherheit vernachlässigbar gering. So ist denn auch keine von der Testzeit abhängige Variation der Endposition (s. Abb. 5.6, rechts) feststellbar.

Die Messungen für die Endpositionen bei ausgefahrener Spindel (oberer Anschlag) zeigen jedoch ein gänzlich anderes Verhalten. Wie in Abbildung 5.7 zu sehen, beträgt die Standardabweichung 63 Counts bei einer Bandbreite von über 250 Counts. Die Abhängigkeit der erreichten Position von der Testdauer (s. Abb. 5.7, rechts) offenbart,

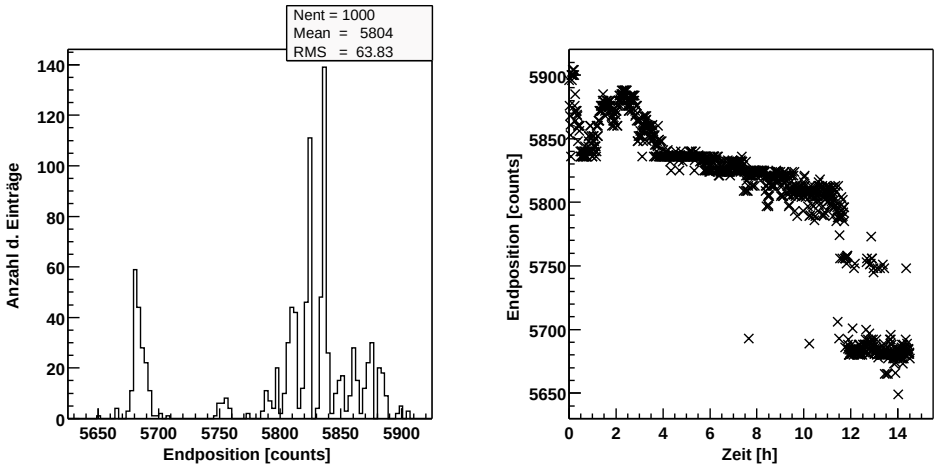


Abbildung 5.7: *Reproduzierbarkeit des Anschlags bei ausgefahrener Spindel (oberer Anschlag). Links: Histogramm der oberen Endposition über eintausend Anfahrten, wobei der Nullpunkt der Position willkürlich gewählt wurde. Rechts: Verteilung der oberen Endpositionen über die gesamte Testzeit zur Veranschaulichung des Langzeitverhaltens.*

daß dies kein statistisches Phänomen ist. Nur über längere Zeiträume zeigen sich große Variabilitäten, während sie sich innerhalb kleiner Zeitabschnitte durchaus in akzeptablen Bereichen bewegen. Auffällig ist auch die sprunghafte Verkürzung gegen Ende der Testzeit.

Diese Unzulänglichkeit im oberen Anschlag war bei allen untersuchten Hamburger Prototypen gegeben, so daß auf einen Fehler im Hamburger Entwurf geschlossen werden konnte. Eingehende Diskussionen mit dem Konstrukteur ergaben, daß dieses Problem wohl auf eine mangelnde Führung der Spindel im oberen Anschlagbereich zurückzuführen sei. Der bei ausgefahrener Spindel im Gewinde verbleibende Teil war sehr knapp bemessen, so daß die Spindel gegenüber der Rotationsachse offensichtlich ein wenig verkippen konnte. Dieses hatte nichtlineare, d. h. stark von Temperatur und Gleitmittelbeschaffenheit abhängige Reibungseffekte zur Folge, die für eine große Variabilität der Motorgeschwindigkeit über längere Zeiträume im oberen Anschlag sorgten.

Die Diskontinuität am Ende der Testzeit konnte allerdings nicht geklärt werden, zeigte sich jedoch in dieser Form auch nur bei diesem einen Aktuator. Öfter gab es solche Sprünge bei anderen Aktuatoren allerdings zu Beginn der jeweiligen Testzeit, wo sich Temperatur und Gleitmittelbeschaffenheit noch nicht stabilisiert hatten.

Eine Verlängerung des im Gewinde verbleibenden Teils der Spindel im ausgefahrenen Zustand schien dem Konstrukteur aufgrund dieser Messungen angezeigt. Da die Weiterentwicklung des Hamburger Entwurfs jedoch bereits eingestellt worden war, fanden die vorangestellten Resultate keine weitere Berücksichtigung. Dennoch konnten hiermit die Leistungsfähigkeit und Zuverlässigkeit der gesamten Steuerung eindrucksvoll gezeigt werden, denn auf andere Weise durchgeführte Tests hatten diese Systematik

nicht aufgedeckt.

5.3.2 Benötigte Fahrzeit zwischen den Anschlägen

Zum näheren Verständnis der obigen Resultate wurde die benötigte Zeit für die Fahrten zwischen den Anschlägen untersucht. Die Aktuatoren werden mit Federn unter Zug gesetzt, um ihr Spiel möglichst gering zu halten, weshalb für die jeweiligen Richtungen unterschiedliche Fahrzeiten zu erwarten sind. Die Auswertung fand deshalb auch für beide Richtungen getrennt statt.

In Abbildung 5.8 ist die benötigte Zeit für die Fahrt vom unteren in den oberen Anschlag über alle eintausend Fahrten dargestellt. Auffallend sind die zwei weit aus-

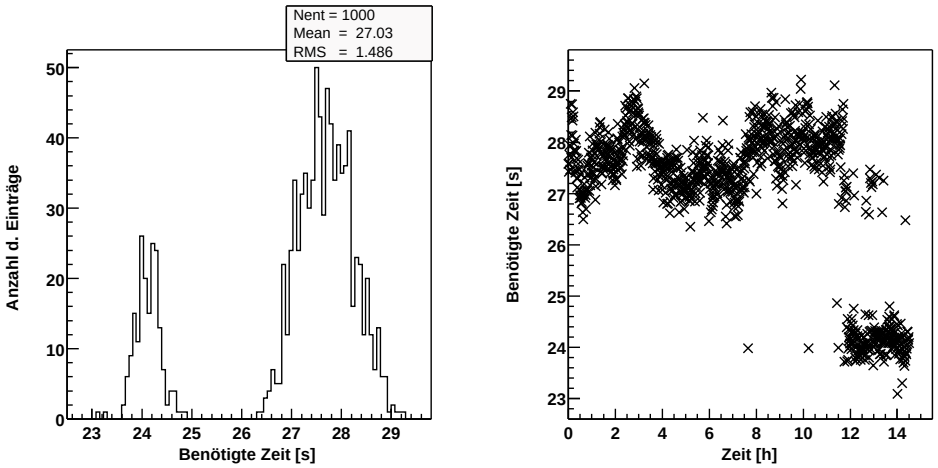


Abbildung 5.8: *Benötigte Zeit für die Fahrt vom Anschlag mit eingefahrener Spindel (unterer Anschlag) zum Anschlag bei ausgefahrener Spindel (oberer Anschlag). Links: Histogramm der benötigten Zeiten über eintausend Fahrten. Rechts: Zur Veranschaulichung des Langzeitverhaltens ist die benötigte Zeit über die gesamte Testzeit aufgetragen.*

einanderliegenden Populationen, zwischen denen sich keine Einträge befinden. Schaut man auf den Verlauf der benötigten Zeit über die Testdauer (s. Abb. 5.8, rechts), zeigt sich, daß die Population mit der geringeren Fahrzeit zeitlich mit dem Einbruch in der Anschlagposition (vgl. Abb. 5.7) korreliert. Wie bereits erwähnt, konnte für dieses Verhalten keine Erklärung gefunden werden.

Die stetig abnehmende Endposition im oberen Anschlag (vgl. Abb. 5.7, rechts) geht dagegen nicht einher mit einer Abnahme der benötigten Fahrzeit. Es läßt sich sogar eine leichte Zunahme vor dem Einbruch ausmachen. Dies stützt die Annahme, daß Reibungseffekte für eine frühzeitige Abnahme der Geschwindigkeit gesorgt haben, so daß die Motorabschaltung an immer kleineren Positionen stattfand.

Wie der Abbildung 5.9 zu entnehmen ist, fällt die benötigte Zeit für die Fahrt vom oberen zum unteren Anschlag aufgund der Spannfedern deutlich kürzer aus. Wie zu

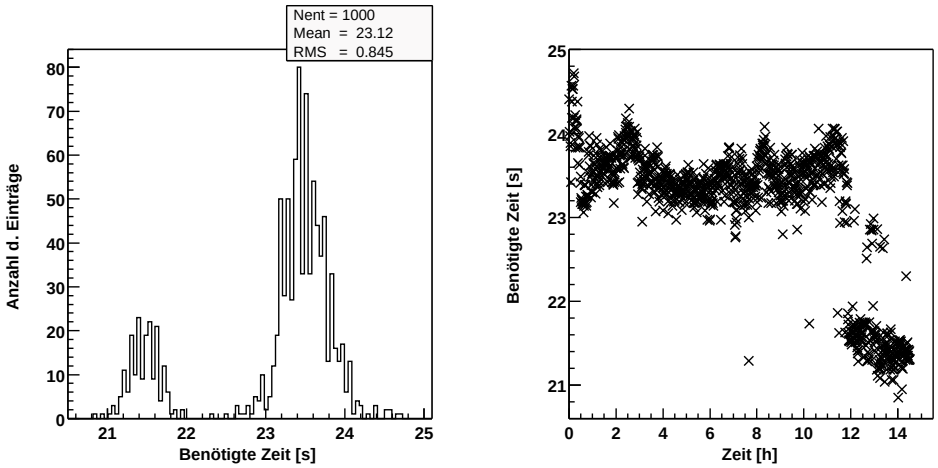


Abbildung 5.9: *Benötigte Zeit für die Fahrt vom Anschlag mit ausgefahrener Spindel (oberer Anschlag) zum Anschlag bei eingefahrener Spindel (unterer Anschlag). Links: Histogramm der benötigten Zeiten über eintausend Fahrten. Rechts: Zur Veranschaulichung des Langzeitverhaltens ist die benötigte Zeit über die gesamte Testzeit aufgetragen.*

erwarten war, findet sich die bereits besprochene Diskontinuität auch hier. Verglichen mit der Fahrt in den oberen Anschlag ist der Sprung jedoch etwas geringer, da es sich hierbei um die Richtung handelt, bei der der Motor durch die Spannfeder unterstützt wird.

5.3.3 Absoluter Hub

Die abschließende Untersuchung widmet sich dem absoluten Hub, d. h. der tatsächlich verfahrbaren Strecke der Aktuatoren, die aus der Differenz der unteren und oberen Anschlagpositionen resultiert.

Die Unsicherheiten in den Anschlagpositionen allein vermögen nur Unzulänglichkeiten im Entwurf der Mechanik aufzuzeigen. Erst das Verhältnis von Variation zum gesamten verfahrbaren Bereich liefert ein Kriterium zur Leistungsfähigkeit des Gesamtsystems, bestehend aus Steuerungshardware und Aktuatorentwurf. Als Maß für die Güte der Kombination aus Steuerungshardware und Aktuator wurde die relative Unsicherheit des absoluten Hubs

$$\frac{\mathrm{Hub}_{\max} - \mathrm{Hub}_{\min}}{\langle \mathrm{Hub} \rangle} \tag{5.2}$$

gewählt, die sich aus der Differenz von Maximalwert und Minimalwert im Verhältnis zum Mittelwert berechnet.

Aus Sicht der Steuerungssoftware wäre ein definierter – eventuell für alle Aktuatoren gleichermaßen gültiger – Verfahrbereich wünschenswert. Abgesehen von einer einmali-

gen Referenzierung, könnten so Fahrten in den Anschlag – und damit hohe Belastungen von Motoren und Aktuatormechanik sowie die Gefahr des Festfahrens – dauerhaft vermieden werden. Um dies zu erreichen, müßte ein Sicherheitsfaktor auf die relative Unsicherheit aufgeschlagen werden, damit sichergestellt werden kann, daß jede gültige Zielposition immer erreichbar ist.

Ein Wert der relativen Unsicherheit von einem Prozent scheint hierfür akzeptabel; Werte deutlich über zwei Prozent würden den verfahrbaren Bereich mit diesem Verfahren jedoch bereits deutlich einschränken.

Auf den Test, ob der verfahrbare Bereich ausreichend dafür ist, den geforderten Winkelbereich für die Bildpunkte der Spiegelfacetten auf der Fokalebene vollständig abfahren zu können, wurde verzichtet. Dies begründet sich durch die bereits eingestellte Weiterentwicklung des Hamburger Entwurfs.

Abbildung 5.10 zeigt die statistische Auswertung des absoluten Hubs für alle zweitausend Fahrten von Anschlag zu Anschlag. Der Verlauf wird dominiert durch die relativ

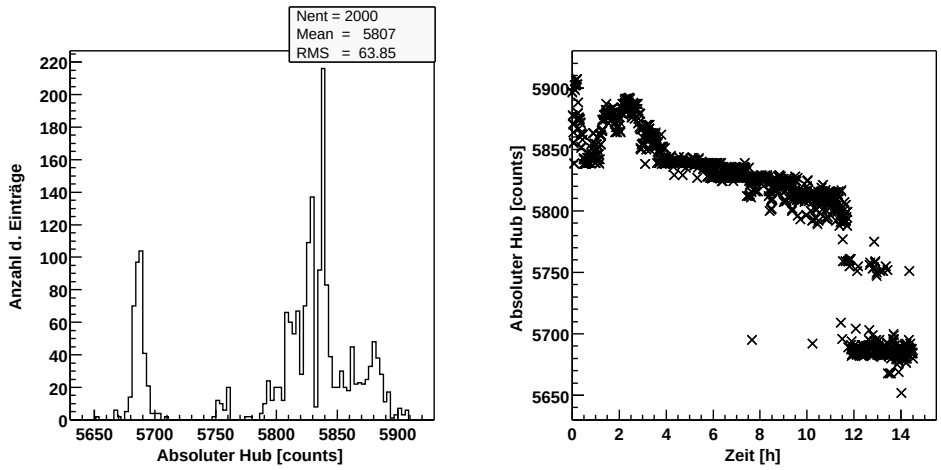


Abbildung 5.10: *Absoluter Hub des Aktuators. Links: Histogramm für alle zweitausend Fahrten von Anschlag zu Anschlag. Rechts: Verhalten des absoluten Hubs über die gesamte Testzeit zur Veranschaulichung des Langzeitverhaltens.*

große Unsicherheit der Endposition im oberen Anschlag. Die Variation des absoluten Hubs von ca. 250 Counts beträgt etwa $4,3\,\%$ der mittleren verfahrbaren Strecke von 5807 Counts. Bei anderen Hamburger Aktuatoren ergaben sich sogar Werte von bis zu $7\,\%$.

Hier zeigt sich deutlich, daß der getestete Hamburger Entwurf überarbeitet werden müßte, um den Anforderungen an einen verläßlichen Einsatz in Namibia gerecht werden zu können.

Trägt man den absoluten Hub gegen die benötigte Fahrzeit von Anschlag zu Anschlag auf (s. Abb. 5.11), so läßt sich abseits der bereits diskutierten Diskontinuität kein ungewöhnliches Verhalten ausmachen.

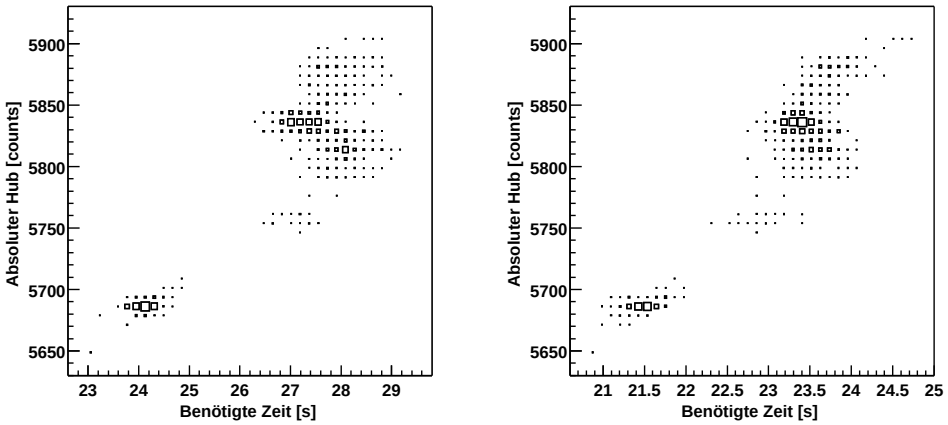


Abbildung 5.11: *Absoluter Hub des Aktuators gegen die benötigte Zeit. Links: Fahrten vom unteren in den oberen Anschlag (gegen die Federkraft). Rechts: Fahrten vom oberen in den unteren Anschlag (mit der Federkraft).*

Da die Spannfeder je nach Fahrtrichtung deutlich unterschiedliche Fahrzeiten zur Folge hat, muß die Abhängigkeit für jede Richtung getrennt aufgetragen werden.

5.4 Test eines Heidelberger Prototypen der Spiegelfacettenmechanik

Schließlich stand der Hamburger Gruppe auch ein Prototyp der Heidelberger Spiegelfacettenmechanik zur Verfügung. Dieser wurde ebenfalls den in Kapitel 5.3 beschriebenen Tests unterzogen. Untersucht wurden nur Eigenschaften, die im Zusammenhang mit der Hamburger Elektronik zur Ansteuerung der Aktuatormotoren stehen. Die Tests, ob die Auslegung der mechanischen Komponenten die Spezifikationen erfüllt, blieben der Heidelberger Gruppe vorbehalten.

Die Ergebnisse werden im Folgenden beispielhaft für einen der beiden (Motor Nr. 0) Aktuatoren vorgestellt. Es wurde hierzu der nach den Meßergebnissen schlechtere gewählt, um ein härteres Kriterium dafür zu haben, ob der Heidelberger Entwurf in Zusammenarbeit mit der Hamburger Ansteuerungselektronik die Anforderungen erfüllen kann.

5.4.1 Reproduzierbarkeit der Anschlagpositionen

In Abbildung 5.12 ist die Endposition bei Fahrten in den unteren Anschlag bei eingefahrener Spindel dargestellt. Mit einer Standardabweichung (RMS) von 4,4 Counts bei absoluter Variation von 23 Counts ist dies der in diesem Sinne am schlechtesten definierte Anschlag der Heidelberger Aktuatoren.

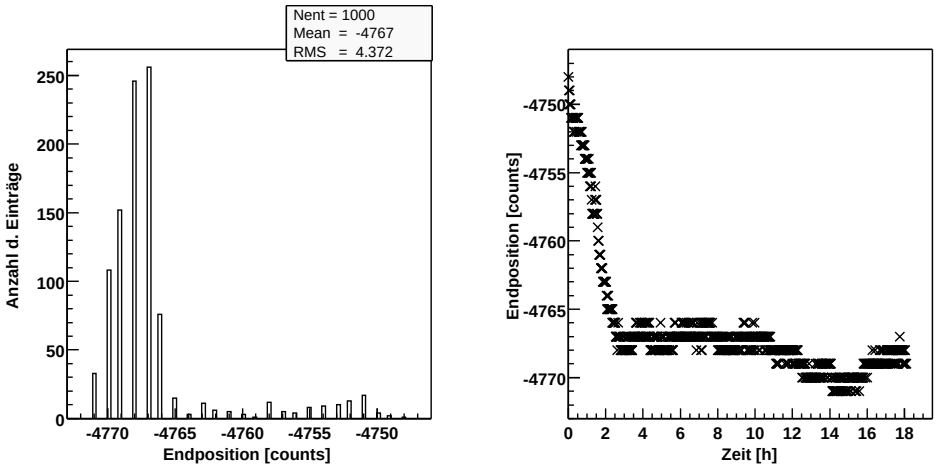


Abbildung 5.12: *Reproduzierbarkeit des Anschlags bei eingefahrener Spindel (unterer Anschlag). Links: Histogramm der unteren Endposition über eintausend Anfahrten, wobei der Nullpunkt der Position willkürlich gewählt wurde. Rechts: Verteilung der unteren Endpositionen über die gesamte Testzeit zur Veranschaulichung des Langzeitverhaltens.*

Blickt man auf den Verlauf der erreichten Endpositionen über die gesamte Testzeit (s. Abb. 5.12, rechts), so fällt auf, daß sie sich zu Beginn des Tests kontinuierlich verkleinern, um sich nach etwa zwei Stunden bei einem kaum noch variierenden Bereich einzupendeln. Dies dürfte darauf zurückzuführen sein, daß sich Temperatur, Gleitmittelbeschaffenheit (Viskosität) und andere Bedingungen erst nach einer gewissen Zeit stabilisieren, um dann annähernd konstant zu bleiben. Dieses Verhalten zeigte sich bei allen Anschlägen der beiden untersuchten Aktuatoren.

Wie aus Abbildung 5.13 ersichtlich, ist das Verhalten im oberen Anschlag, d. h. bei ausgefahrener Spindel, ähnlich. Zwar sind die Werte für Standardabweichung (3,5 Counts) und absolute Variation (10 Counts) besser als im unteren Anschlag, jedoch ist der Hauptteil der Einträge in einem nicht derart ausgeprägten Bereich konzentriert, wie dies im unteren Anschlag der Fall ist.

Die Werte für den hier nicht dargestellten Aktuator betrugen 2,8 bzw. 0,98 Counts für die Standardabweichung und 13 bzw. 7 Counts für die absolute Variation im jeweils unteren bzw. oberen Anschlag. Alle Werte können als hervorragend gelten und bestätigen, daß eine Motorabschaltung über ausbleibende Flankenwechsel der Hallsensoren dazu geeignet ist, die Anschläge zu definieren.

5.4.2 Benötigte Fahrzeit zwischen den Anschlägen

Die benötigte Fahrzeit vom unteren zum oberen Anschlag ist in Abbildung 5.14 dargestellt. Auffällig ist, daß sich zwei Populationen ausmachen lassen. Blickt man auf den Verlauf der benötigten Fahrzeit über die gesamte Testdauer, so zeigt sich, daß gegen

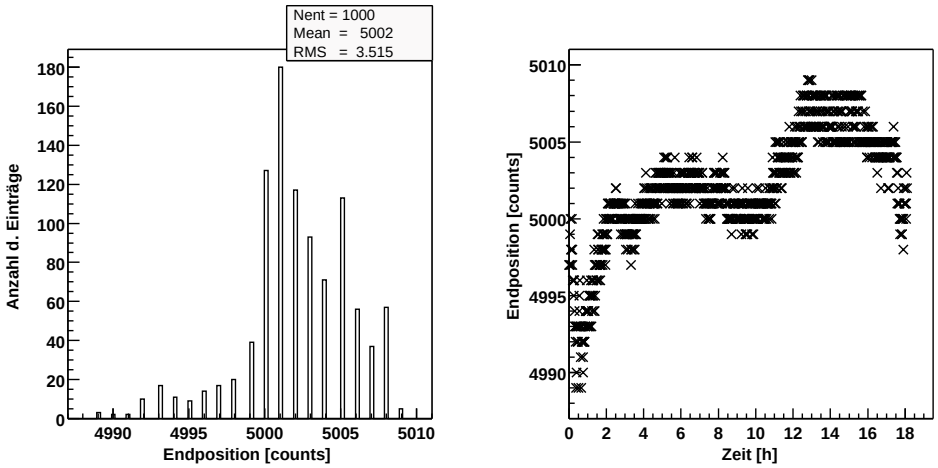


Abbildung 5.13: *Reproduzierbarkeit des Anschlags bei ausgefahrener Spindel (oberer Anschlag). Links: Histogramm der oberen Endposition über eintausend Anfahrten, wobei der Nullpunkt der Position willkürlich gewählt wurde. Rechts: Verteilung der oberen Endpositionen über die gesamte Testzeit zur Veranschaulichung des Langzeitverhaltens.*

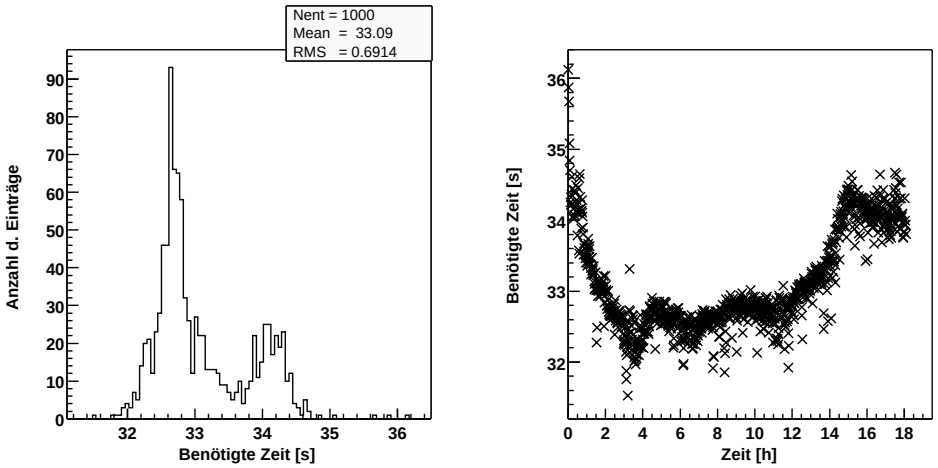


Abbildung 5.14: *Benötigte Zeit für die Fahrt vom Anschlag mit eingefahrener Spindel (unterer Anschlag) zum Anschlag bei ausgefahrener Spindel (oberer Anschlag). Links: Histogramm der benötigten Zeiten über eintausend Fahrten. Rechts: Zur Veranschaulichung des Langzeitverhaltens ist die benötigte Zeit über die gesamte Testzeit aufgetragen.*

Ende der Testzeit die Fahrzeiten deutlich angestiegen sind. Eine Erklärung konnte hierfür nicht gefunden werden, obwohl auch der andere Aktuator ein ähnliches Verhalten bei der Fahrt in den oberen Anschlag zeigte. Ein Problem sollte daraus jedoch nicht

resultieren.

Das deutliche Absinken der benötigten Fahrzeit zu Beginn des Tests dürfte auf die bessere Viskosität des erwärmten Gleitmittels zurückzuführen sein. Dieses Verhalten konnte bei allen Fahrzeiten der Heidelberger Aktuatoren festgestellt werden.

In Abbildung 5.15 ist die benötigte Fahrzeit vom oberen in den unteren Anschlag dargestellt. Bis auf die erwähnte deutliche Abnahme zu Beginn der Testzeit sind kei-

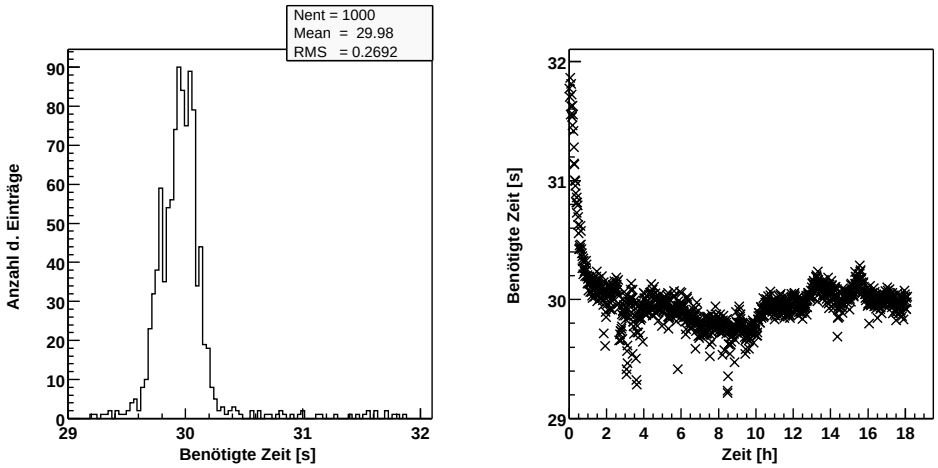


Abbildung 5.15: *Benötigte Zeit für die Fahrt vom Anschlag mit ausgefahrener Spindel (oberer Anschlag) zum Anschlag bei eingefahrener Spindel (unterer Anschlag). Links: Histogramm der benötigten Zeiten über eintausend Fahrten. Rechts: Zur Veranschaulichung des Langzeitverhaltens ist die benötigte Zeit über die gesamte Testzeit aufgetragen.*

ne Besonderheiten auszumachen. Nach Stabilisierung der Bedingungen ist der Verlauf nahezu konstant.

5.4.3 Absoluter Hub

Wie in Kapitel 5.3.3 dargelegt, ist die Untersuchung des absoluten Hubs, d. h. der Differenz von oberer und unterer Anschlagposition, wichtig für die Software zur Justierung der Spiegelfacetten. Die relative Unsicherheit des absoluten Hubs bestimmt, ob eine Festlegung des verfahrbaren Bereichs durch einmaliges Referenzieren möglich ist.

Die Ergebnisse der Messungen zum absoluten Hub sind in Abbildung 5.16 zu sehen. Der Mittelwert über alle zweitausend Fahrten beträgt 9769 Counts bei einer Standardabweichung (RMS) von 7,5 Counts. Es ist zudem auszumachen, daß die meisten Einträge in einem relativ engen Bereich liegen. Blickt man auf den Verlauf des absoluten Hubs über die gesamte Testzeit (s. Abb. 5.16, rechts), so wird deutlich, daß die wenigen Einträge außerhalb dieses Bereichs auf die schon erwähnte Einstellungsphase zu Beginn der Messung beschränkt sind. Nachdem sich die Betriebsbedingungen sta-

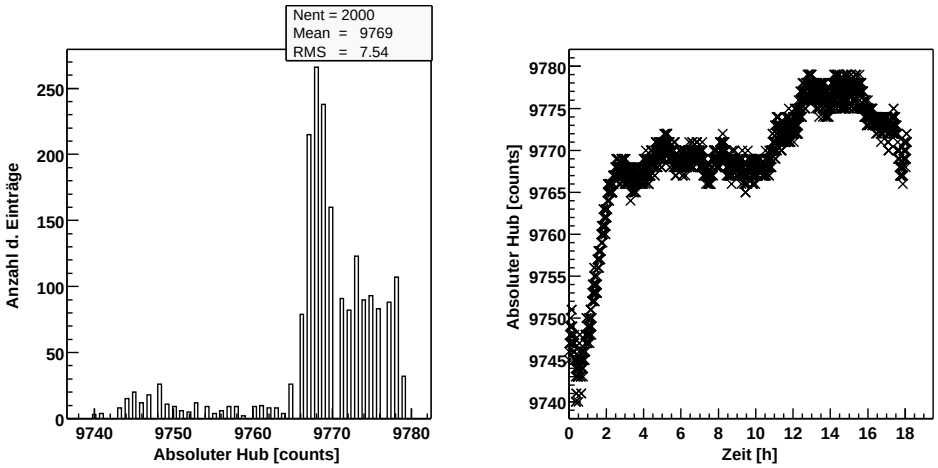


Abbildung 5.16: *Absoluter Hub des Aktuators. Links: Histogramm für alle zweitausend Fahrten von Anschlag zu Anschlag. Rechts: Verhalten des absoluten Hubs über die gesamte Testzeit zur Veranschaulichung des Langzeitverhaltens.*

bilisiert hatten (etwas über zwei Stunden nach Testbeginn) variierte der absolute Hub nur noch in engen Grenzen oberhalb von ca. 9765 Counts.

Die relative Unsicherheit des absoluten Hubs ergibt sich nach Gleichung (5.2) zu 0,4 %. Für den anderen Heidelberger Aktuator ergab sich sogar ein Wert von nur 0,2 %. Diese hervorragenden Werte ermöglichen tatsächlich, den verfahrbaren Bereich einmalig festzustellen, um anschließend Fahrten in den Anschlag – und damit ein mögliches Festfahren – zu vermeiden.

Allerdings sollte dies für jeden Aktuator gesondert durchgeführt werden, denn der verfahrbare Bereich (absoluter Hub) variiert von Aktuator zu Aktuator. So ergab sich für den anderen Heidelberger Aktuator ein mittlerer absoluter Hub von 9568 Counts und damit nur knapp 98 % vom Wert des hier gezeigten Aktuators.

Ergänzend ist in Abbildung 5.17 der absolute Hub, wiederum für beide Richtungen getrennt, gegen die benötigte Fahrzeit aufgetragen.

Wie aus Abbildung 5.16 (rechts) ersichtlich, beschränken sich die Einträge nach der Stabilisierungsphase auf absolute Hübe oberhalb von etwa 9765 Counts. In Abbildung 5.17 sind diese Bereiche (Einträge oberhalb von etwa 9765 Counts) für beide Richtungen gut definiert. Lediglich während der Stabilisierung der Betriebsbedingungen (Einträge unterhalb von etwa 9765 Counts) ist eine größere Variation der Einträge festzustellen.

Ein ungewöhliches Verhalten läßt sich nicht ausmachen, was auch für den hier nicht dargestellten Aktuator gilt.

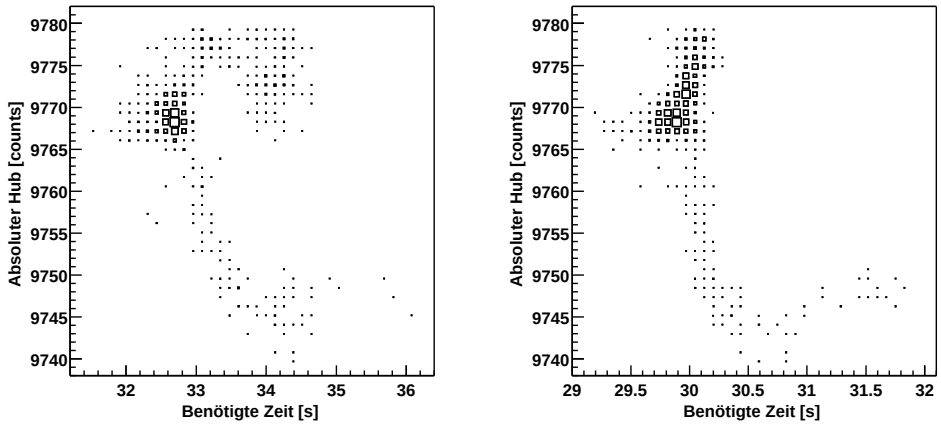


Abbildung 5.17: *Absoluter Hub des Aktuators gegen die benötigte Zeit. Links: Fahrten vom unteren in den oberen Anschlag (gegen die Federkraft). Rechts: Fahrten vom oberen in den unteren Anschlag (mit der Federkraft).*

5.4.4 Bewertung der Testergebnisse

Die Tests konnten nur an einem Prototyp des Heidelberger Entwurfs durchgeführt werden. In der Annahme, daß dieser Prototyp ein typischer Vertreter des Heidelberger Entwurfs ist, lassen sich die Testergebnisse folgendermaßen bewerten:

Die Tests haben gezeigt, daß die Abschaltung der Motoren bei für eine gewisse Zeit ausbleibenden Flankenwechseln der Hallsensoren die Endpositionen der Spindeln reproduzierbar zu definieren vermag. Es kam nie zur Abschaltung zwischen den Anschlägen und nie war ein Aktuator so weit in einen Anschlagbereich gefahren, daß er aus disem nicht mehr herausbewegt werden konnte.

Zudem ermöglicht die geringe Unsicherheit im verfahrbaren Bereich mit deutlich unter einem Prozent ein einmaliges Ausmessen des absoluten Hubs, ohne ihn zu sehr einschränken zu müssen. Dies sollte jedoch für jeden Aktuator gesondert durchgeführt werden, da die Werte von Aktuator zu Aktuator zu sehr variieren, um einen für alle Aktuatoren gültigen Wert ohne größere Einschränkungen festlegen zu können. Eine Einschränkung bedeutet dies jedoch nicht, denn es ist unproblematisch, den verfahrbaren Bereich für jeden Aktuator einzeln auszumessen.

Während der Tests sind keine Probleme (d. h. unerwartetes Verhalten jeglicher Art) aufgetreten, so daß die Steuerungssoftware so ausgelegt werden kann, daß bei ungewöhnlichem Verhalten eines Aktuators auf einen Fehler zu schließen ist. Dies wird die Entwicklung der Steuerungssoftware sehr vereinfachen, denn wäre ungewöhnliches Verhalten die Regel, müsste die Steuerung mit sehr vielen Sonderfällen umgehen können.

Abschließend läßt sich festhalten, daß die Kombination aus Hamburger Steuerelektronik und Heidelberger Spiegelfacettenmechanik die an sie gestellten Anforderungen

erfüllt und eine gute Grundlage für eine softwaregesteuerte Justierung der Spiegelfacetten bildet.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

In der vorliegenden Arbeit wurden Untersuchungen zur automatisierten Justierung der Spiegelfacetten der H·E·S·S· Cherenkov–Teleskope angestellt. Die Justierung wird durch an die Spiegelfacetten angebrachte Aktuatoren ermöglicht, die von kostengünstigen Getriebemotoren verstellt werden können. Zur direkten Ansteuerung der Aktuatormotoren kommt eine von der Universität Hamburg entwickelte Elektronik zum Einsatz, deren Funktionalität über den VMEbus von einem Steuerungsprogramm genutzt wird.

Das mit H·E·S·S· *Mirror Alignment Controll System* (MACS) bezeichnete Softwaresystem nimmt die Abbilder aller Spiegelfacetten auf dem Deckel der Teleskopkamera mittels einer CCD–Kamera auf. Durch eine geeignete Analyse der Bilder kann bestimmt werden, welche Verstellungen der Aktuatoren nötig sind, um eine Ausrichtung aller Spiegelfacetten zu erreichen.

Die erste Aufgabe im Rahmen dieser Arbeit bestand in der Vervollständigung der Hardwarekomponenten zur Ansteuerung der Aktuatormotoren. Hierzu wurden die am Markt erhältlichen industriellen Standardgeräte miteinander verglichen und geeignete Komponenten einer eingehenderen Betrachtung unterzogen. So konnten einige unterschiedliche Konzepte erarbeitet werden, die in der Folge innerhalb der Hamburger Gruppe zur Diskussion gestellt wurden. Nach der Entscheidung über die zu wählenden Komponenten wurden diese angeschafft und zusammen mit der Hardware zur Ansteuerung der Aktuatormotoren in Betrieb genommen.

Hiernach galt es das Zusammenspiel der verschiedenen Komponenten auf Probleme hin zu untersuchen. Durch ausführliche Tests konnten einige Fehler und Unzulänglichkeiten in den Eigenentwicklungen entdeckt und Vorschläge zu deren Behebung erarbeitet werden. Nach Beseitigung dieser Fehler und Unzulänglichkeiten sind trotz intensiver mehrmonatiger Verwendung aller Hardwarekomponenten keine weiteren Probleme zutage getreten.

Um eine weitgehend automatisierte Justierung der Spiegelfacetten zu ermöglichen, wurden verschiedene Softwaretechnologien auf ihre Tauglichkeit für die vorliegende

Aufgabe hin untersucht. Vier unterschiedliche Lösungsansätze für das Grundkonzept wurden ausgearbeitet und der Hamburger Gruppe zur Diskussion vorgelegt. Die Entscheidung fiel dabei zugunsten eines objektorientierten Ansatzes aus.

In der Folge wurde ein Detailkonzept des Systems erstellt und einige der darin enthaltenen Softwarekomponenten modelliert. Zur Abbildung immer wieder gebrauchter Funktionalitäten wurden Klassen implementiert, die allen auf sie aufbauenden Programmen als Basis dienen. Dadurch konnte die Entwicklung dieser Programme effizient erfolgen. Zudem wurden zur programmtechnischen Kapselung der Hardwarekomponenten spezielle Controller–Klassen implementiert, die über mehrschichtige Abstraktionsniveaus verfügen.

Die erste vollständig zu implemtierende Softwarekomponente war der zentrale Server–Daemon für die Kommunikation mit der Hardware zur Ansteuerung der Aktuatormotoren, da dieser unabdingbar für die an der Hardware durchzuführenden Tests war. Aufbauend auf die bereits erstellten Basis– und Controller–Klassen gestaltete sich der Entwicklungsprozeß recht kurz. Nach einer anfänglichen Fehlerbereinigungsphase lief das Programm über mehrere Monate hindurch fehlerfrei und konnte so seine Tauglichkeit für den Einsatz in Namibia unter Beweis stellen.

Des weiteren wurden sowohl Hamburger Entwürfe als auch ein Heidelberger Prototyp der Aktuatormechanik ausgiebigen Tests unterzogen. Im Falle der Hamburger Modelle konnten diese Tests einen Fehler im Entwurf aufdecken, der jedoch aufgrund der bereits eingestellten Entwicklung nicht weiter berücksichtigt wurde. Die Tests der Heidelberger Spiegelfacettenmechanik konnten zeigen, daß die Kombination mit der Hamburger Elektronik die an sie gestellten Anforderungen zu erfüllen vermag.

Abschließend wurde ein Programm zum Test der Verlötung der Motoren mit den Relaisboxen entwickelt und eine Dokumentation dazu angefertigt. Programm und Dokumentation wurden so gestaltet, daß sie von den mit dieser Aufgabe betrauten Mitarbeitern einfach zu bedienen bzw. zu verstehen sind.

6.2 Ausblick

Nicht alle Komponenten der Software zur automatisierten Justierung der Spiegelfacetten konnten im Rahmen dieser Arbeit fertiggestellt werden. Die bereits erstellten Teile sollten jedoch eine gute Basis zur Vervollständigung liefern.

Zum einen ist noch die Software zum Betrieb der CCD–Kamera zu entwickeln. Die CCD–Kamera wurde bisher nicht in die MACS–Software integriert, sondern nur mit den zur Verfügung stehenden Programmen zu Testzwecken betrieben. Aufbauend auf die bereits erstellten Teile der MACS–Software sowie auf die entsprechenden Hardwaretreiber sollte der Entwicklungsprozeß jedoch recht kurz gehalten werden können. Da die CCD–Kamera auch für andere Zwecke genutzt werden wird, ist für diese Softwarekomponente die allgemein zum Einsatz kommende Kommunikationstechnologie (s. u.) zu verwenden.

Zum anderen gilt es den zentralen Steuerungsprozeß zur Justierung der Spiegelfacetten zu implementieren. Dieser beinhaltet die Verwaltung aller Parameter und Zustands-

informationen der Hardware, die zur Justierung der Spiegelfacetten benötigt werden. Die Softwareteile für die Kommunikation mit der Hardware zur Ansteuerung der Aktuatormotoren wurden bereits fertiggestellt und können einfach integriert werden.

Zudem muß das Programm über die Möglichkeit verfügen, die Bilder der CCD–Kamera zu analysieren. Dazu stehen bereits einige verschiedene Programmbibliotheken bereit, von denen es eine geeignete Alternative auszuwählen und gegebenenfalls zu erweitern gilt.

Der größte Teil der noch zu leistenden Arbeit besteht in der Entwicklung der Algorithmen, welche die Berechnungsvorschriften für eine weitgehend automatisierte Justierung enthalten. Weicht ein Ergebnis bzw. Ereignis vom Erwarteten ab, so kann ein Operateur in einem interaktiven Prozeß relativ leicht den Zustand analysieren und über das weitere Vorgehen entscheiden. Einen Algorithmus so zu gestalten, daß es möglichst niemals zu Fehlinterpretationen und –entscheidungen kommt, ist äußerst schwierig; oft ist dies nur auf Kosten der Flexibilität zu erreichen. In diesem Bereich sind deshalb die größten Optimierungsarbeiten während und nach der ersten Inbetriebnahme zu erwarten.

Abschließend sind Möglichkeiten zur Kommunikation mit anderen Teilen der H·E·S·S·-Software zu schaffen.

Über die Technologie zur Kommunikation aller Softwareteile des H·E·S·S·–Projekts ist innerhalb der H·E·S·S·–Kollaboration noch keine Entscheidung gefällt worden. Zur Diskussion stehen zwei Alternativen, die beide mit der MACS–Software verträglich sind. Nach der Entscheidung muß ein auf die entsprechende Technologie aufbauendes Kommunikationsprotokoll festgelegt werden, das als Basis für die Verständigung zwischen der MACS–Software und den sie nutzenden Client–Programmen dient.

Des weiteren ist festzulegen, welche internen Parameter der MACS–Software von allgemeinem Interesse sind. Diese gilt es dann in der zentralen Datenbank das H·E·S·S·–Projektes abzulegen, wozu entsprechende Schnittstellen auf beiden Seiten zu schaffen sind.

Anhang A

Dokumentation: MACS Motor Test Applikation

Das Programm *macsMotorTest* wurde entwickelt, um die Verlötungen der ca. achthundert Motoren mit den Relaisboxen für jedes der H·E·S·S·–Teleskope weitgehend automatisiert testen zu können. Fehler wie

- defekte Motoren,
- defekte Relaisplatinen,
- defekte Kabel und Steckverbinder,
- fehlerhafte Verlötungen und
- Verpolungen in der Verlötung

sollten dabei möglichst ohne Eingriff des Bedienpersonals erkannt werden können.

Nachstehend findet sich die Dokumentation anhand der das Bedienpersonal die Tests mittels des Testprogramms durchzuführen hat.

MACS Motor Test Application (macsMotorTest) Dokumentation

22. Januar 2001

René Cornils
II. Institut für Experimentalphysik
Universität Hamburg

A.1 Übersicht

Die Spiegelfacetten der H·E·S·S·–Teleskope werden motorisch justiert, wozu sie mit zwei mit Motoren bestückten Aktuatoren versehen sind.

Die Verlötung der ca. achthundert Motoren mit den Relaisboxen für jedes der H·E·S·S·–Teleskope wird in Hamburg durchgeführt. Um eine aufwendige manuelle Kontrolle der Verlötungen sowie der Motoren, Relaisplatinen, Kabel und Steckverbinder zu vermeiden, wurde ein Programm entwickelt, das diesen Vorgang weitgehend automatisiert.

Zum Test der Einheiten steht ein sog. Branch mit 32 Knoten zur Verfügung, an dem die zu testenden Module vor Testbeginn anzuschließen sind. Sollten weniger als 32 Module zu testen sein, sind immer die höchsten Knoten zu besetzen, da für alle unbesetzten Knoten die Meldung „No Connection“ ausgegeben wird. So ist gewährleistet, daß die relevanten Meldungen nicht vom Bildschirm verschwinden.

A.2 Programmstart

Das Programm wird durch die Eingabe von

```
macsMotorTest
```

auf der Kommandozeile gestartet. Nach einigen Sekunden erscheint daraufhin die Meldung

```
MACS Motor Test Application (macsMotorTest)
Press '1' and <Return> to start motor test cycle.
Press '2' and <Return> to start direction test cycle.
Press <Ctrl-C> and <Return> to abort.
Command:
```

die die Bereitschaft des Programms anzeigt.

Alternativ kann dem Programm beim Start eine andere als die voreingestellte Branch–Nummer und/oder Dauer für den Richtungstest mitgegeben werden. Dies geschieht durch den Aufruf

```
macsMotorTest Branch=X DirSecs=Y
```

wobei `X` durch die Nummer des gewünschten Branchs (0–15) und `Y` durch die Dauer des Richtungstests in ganzen Sekunden pro Motor (1–999) zu ersetzen sind.

Experimentierfreudige Naturen können zudem durch die Eingabe von

```
macsMotorTest -help
```

eine Übersicht über alle erlaubten Kommandozeilen–Optionen erhalten.

A.3 Konfigurationsfile

Das Programm ist weitgehend konfigurierbar. Einige Optionen lassen sich beim Start des Programmes angeben, der überwiegende Teil wird jedoch in einem Konfigurationsfile festgelegt.

> Achtung: Änderungen am Konfigurationsfile sollten nur von autorisierten Personen vorgenommen werden, da nur so ein fehlerfreier Test gewährleistet werden kann!

Das Default–Konfigurationsfile kann durch die Eingabe von

```
macsMotorTest -c
```

auf der Kommandozeile erhalten werden. Es gestaltet sich wie folgt:

```
{
    Branch = 0;
    ConfFile = /usr/local/macs/etc/macsMotorTest.conf;
    DirSecs = 2;
    LogFile = /usr/local/macs/log/macsMotorTest.log;
    LogLevel = 4;
    NoCounter = 10;
    ServerID = 00;
    SlowMotion = 700;
}
```

An dieser Stelle sei nur eine kurze Beschreibung der einzelnen Optionen angeführt:

Branch: Legt die Nummer des Branchs fest, an dem die Motoren zum Test angeschlossen werden (0–15).

ConfFile: Zeigt den Pfad für das Default–Konfigurationsfile.

DirSecs: Bestimmt die Dauer des Richtungstest in Sekunden pro Motor (1–999).

LogFile: Gibt an, wohin die Log–Informationen geschrieben werden.

LogLevel: Legt fest, ab welcher Priorität Meldungen in das Logfile ausgegeben werden (vgl. `macsMotorTest -help`).

NoCounter: Sollte der Positionszähler nach dem Fahren eines Motors diesen Wert unterschreiten, wird der Fehler „No Counter“ ausgegeben.

ServerID: Legt die ID des Server–Daemons ‚macsVMEd‘ fest (z. Zt. fest `00`).

SlowMotion: Sollte der Positionszähler nach dem Fahren eines Motors diesen Wert unterschreiten, wird der Fehler „Slow Motion“ ausgegeben.

A.4 Bedienung

Ist das Programm gestartet, zeigt es seine Bereitschaft durch die Meldung

```
MACS Motor Test Application (macsMotorTest)
Press '1' and <Return> to start motor test cycle.
Press '2' and <Return> to start direction test cycle.
Press <Ctrl-C> and <Return> to abort.
Command:
```

an. Durch Betätigen der Taste `1` mit anschließendem `<Return>` wird ein Motortestzyklus (s. Kap. A.5) gestartet. Dieser ermittelt vollautomatisch das korrekte Funktionieren aller an dem Branch angeschlossenen Motoren.

Der automatische Testlauf ist allerdings nicht in der Lage, zu detektieren, ob sowohl Motor als auch Hallsensoren falsch gepolt sind. Dafür ist der Richtungstestlauf (s. Kap. A.6) vorgesehen, der mit `2` und anschließendem `<Return>` gestartet wird.

Durch Betätigung der Tastenkombination `<Ctrl-C>` mit darauffolgendem `<Return>` wird das Programm beendet. Das Programm kann zudem auch während eines Testlaufs jederzeit mit der Tastenkombination `<Ctrl-C>` abgebrochen werden.

Tritt beim Test eines Motors ein Fehler auf, wird dieser am Bildschirm ausgegeben (s. Kap. A.7).

A.5 Ablauf eines Motortestzyklus

Der Motortestzyklus ermittelt vollautomatisch das korrekte Funktionieren aller an dem eingestellten Branch angeschlossenen Motoren. Dabei werden folgende Schritte für alle Motoren abgearbeitet:

1. Select: Der Motor wird selektiert.
   Sollte die Anwahl eines Motors fehlschlagen, wird die Meldung „No Connection“ ausgegeben.

2. Drive Up, Speed 1: Der Motor wird für zwei Sekunden mit Geschwindigkeit 1 in positiver Richtung gestartet.

3. Check No Motion: Test, ob der Motor noch fährt.
   Sollte der Motor nicht mehr fahren, wird die Fehlermeldung „No Motion“ ausgegeben.

4. Drive Up, Speed 7: Die Geschwindigkeit wird für zwei Sekunden auf 7 gestellt.
5. Check No Motion: Test, ob der Motor noch fährt.
   Sollte der Motor nicht mehr fahren, wird die Fehlermeldung „No Motion“ ausgegeben.
6. Stop Motor: Der Motor wird gestoppt.
7. Check No Counter: Test, ob der Positionszähler den Minimalwert überschreitet.
   Sollte der Zähler den per Option ‚NoCounter‘ eingestellten Minimalwert nicht überschreiten, wird die Fehlermeldung „No Counter“ ausgegeben.
8. Check Wrong Direction: Test, ob der Positionszähler einen positiven Wert hat.
   Sollte der Zähler einen negativen Wert aufweisen, wird die Fehlermeldung „Wrong Direction“ ausgegeben.
9. Check Slow Motion: Test, ob der Positionszähler einen Minimalwert überschreitet.
   Sollte der Zähler den per Option ‚SlowMotion‘ eingestellten Minimalwert nicht überschreiten, wird die Fehlermeldung „SlowMotion“ ausgegeben.
10. Die Schritte 2 bis 9 werden mit negativer Drehrichtung durchlaufen.

Während eines Testlaufs wird die gerade ablaufende Aktion in überschreibender Weise am Bildschirm protokolliert. Eine Protokollzeile gestaltet sich dabei folgendermaßen:

```
Node: XY  Motor: Z    ACTION
```

`XY` (00–31) bezeichnet den angesprochenen Knoten, `Z` (0,1) den gerade aktiven Motor. `ACTION` gibt Auskunft über die auszuführende Aktion. Folgende Aktionen werden protokolliert:

```
Selecting
Driving Up,   Speed: 1
Driving Up,   Speed: 7
Driving Down, Speed: 1
Driving Down, Speed: 7
```

Tritt einer der oben angegebenen Fehler auf, wird dieser nicht–überschreibend am Bildschirm protokolliert und der Testlauf des aktuellen Motors abgebrochen. Der Testzyklus fährt daraufhin mit dem nächsten Motor fort.

A.6 Ablauf eines Richtungstestzyklus

Falls bei einem Motor sowohl die Polarität der Spannungsversorgung als auch die Beschaltung der Hallsensoren vertauscht wurden, ist die Software nicht in der Lage, einen Fehler festzustellen. Zwar dreht der Motor in die entgegengesetzte Richtung, jedoch auch der Positionszähler zählt entgegengesetzt, so daß sich beide Effekte kompensieren.

Um diese Fehlbeschaltung zu erkennen, ist die visuelle Kontrolle der korrekten Drehrichtung der Motoren vonnöten. Dazu wird im Richtungstestzyklus jeder Motor für eine festgelegte Dauer in positiver Richtung (rechts herum/im Uhrzeigersinn bei Draufsicht auf die Antriebsschnecke) gefahren, während dieser der Operateur die korrekte Drehrichtung zu kontrollieren hat.

Ein Zyklus wird erst vier Sekunden nach Eingabe des Startkommandos begonnen, um dem Operateur Zeit zu geben, den ersten Motor in Augenschein zu nehmen. Folgende Schritte werden dann für jeden Motor abgearbeitet:

1. Select: Der Motor wird selektiert.
   Sollte die Anwahl eines Motors fehlschlagen, wird die Meldung „No Connection“ ausgegeben.

2. Drive Up, Speed 6: Der Motor wird für die per Option ‚DirSecs‘ spezifizierte Zeit in Sekunden mit Geschwindigkeit 6 in positiver Richtung gestartet.
   Während dieser Phase ist vom Operateur zu kontrollieren, ob der Motor tatsächlich rechts herum bzw. im Uhrzeigersinn (bei Draufsicht auf die Antriebsschnecke) dreht.

3. Check No Motion: Test, ob der Motor noch fährt.
   Sollte der Motor nicht mehr fahren, wird die Fehlermeldung „No Motion“ ausgegeben.

4. Stop Motor: Der Motor wird gestoppt.

5. Check No Counter: Test, ob der Positionszähler den Minimalwert überschreitet.
   Sollte der Zähler den per Option ‚NoCounter‘ eingestellten Minimalwert nicht überschreiten, wird die Fehlermeldung „No Counter“ ausgegeben.

6. Check Wrong Direction: Test, ob der Positionszähler einen positiven Wert hat.
   Sollte der Zähler einen negativen Wert aufweisen, wird die Fehlermeldung „Wrong Direction“ ausgegeben.

Die aktuell ausgeführte Aktion wird analog zum Motortestlauf auf dem Bildschirm protokolliert:

```
Selecting
Driving Up,   Speed: 6
```

Wie beim Motortestzyklus wird im Fehlerfall eine Meldung nicht–überschreibend am Bildschirm ausgegeben. Entsprechend wird der Testlauf mit dem fehlerhaften Motor beendet und es wird mit dem nächsten Motor fortgefahren.

Falls die Dauer der Motorbewegung zur visuellen Kontrolle durch den Operateur nicht ausreicht bzw. zu viel Zeit in Anspruch nimmt, kann sie mit der Option ‚DirSecs‘ (s. Kap. A.2) auf den gewünschten Wert gesetzt werden.

A.7 Fehlermeldungen

Tritt während des Tests eines Motors ein Fehler auf, wird dieser am Bildschirm nach dem Muster

```
Node: XY  Motor: Z    ERROR
```

protokolliert. Im folgenden findet sich eine Auflistung aller Fehlermeldungen samt den möglichen Ursachen:

No Connection: Es ist kein Motor angeschlossen oder es besteht keine elektrische Verbindung zu ihm.

No Motion: Der Motor ist defekt oder äußerst schwergängig.
Auch ist es möglich, daß der Fehler ‚Overtemperature' an der *Branch Control Unit* aufgetreten ist.

No Counter: Die Hallsensoren sind nicht korrekt beschaltet.

Wrong Direction: Der Motor oder die Hallsensoren sind falsch gepolt.

Slow Motion: Der Motor erhält zu wenig Spannung oder ist schwergängig.

Zu beachten ist, daß softwareseitig nicht detektiert werden kann, ob sowohl Motor als auch Hallsensoren falsch gepolt sind. Dafür ist der Richtungstest (s. Kap. A.6) vorgesehen, der allerdings eine visuelle Kontrolle durch den Operateur erfordert.

A.8 Fragen und Probleme

Für Fragen steht R. Cornils (Durchwahl -2194) gerne zur Verfügung.

Sollten Probleme oder hier nicht beschriebene Fehlermeldungen auftreten, ist in jedem Falle R. Cornils zu kontaktieren. Alternativ kann auch Herr Riege (Durchwahl -2210) angesprochen werden.

Literaturverzeichnis

[Ahar 97a] F. Aharonian et al. (The H.E.S.S. Collaboration)
HESS Letter of Intent: An Array of Imaging Atmospheric Cherenkov Teleskopes for Stereoscopic Observation of Air Showers from Cosmic Gamma Rays in the 100 GeV Range
(March 24, 1997)
http://www-hfm.mpi-hd.mpg.de/HESS/public/hessloi3.ps.gz

Appendix A: Physics Motivation
http://www-hfm.mpi-hd.mpg.de/HESS/public/PhJ.ps.gz

Appendix B: Conceptual Design of the System Telescopes
http://www-hfm.mpi-hd.mpg.de/HESS/public/system.ps.gz

[Ahar 97b] F. Aharonian et al.
The potential of ground based arrays of imaging atmospheric Cherenkov telescopes, I. Determination of shower parameters
Astroparticle Physics, **6** (1997) 343–368

[Ahar 97c] F. Aharonian et al.
The potential of ground based arrays of imaging atmospheric Cherenkov telescopes, II. Gamma ray flux sensitivities
Astroparticle Physics, **6** (1997) 369–377

[Ahar 99] F. Aharonian et al. (The HEGRA Collaboration)
Observations of Mkn 421 during 1997 and 1998 in the energy range above 500 GeV with the HEGRA stereoscopic Cherenkov telescope system
Astronomy and Astrophysics, **350** (1999) 757–764

[Ahar 00] F. Aharonian et al. (The HEGRA Collaboration)
The Energy Spectrum of TeV Gamma-Rays from the Crab Nebula as measured by the HEGRA system of imaging air Cherenkov telescopes
The Astrophysical Journal, **539** (2000) Issue 1, 317–324

[Ahar 01] F. Aharonian et al. (The HEGRA Collaboration)
The TeV Energy Spectrum of Markarian 501 Measured with the Stereosco-

pic Telescope System of HEGRA during 1998 and 1999
The Astrophysical Journal, **546** (2001), Number 2, Part 1, 898–902

[Agil 99] Agilent Technologies, Inc.
Quadrature Decoder/Counter Interface ICs
Technical Data Library (1999)
ftp://ftp.agilent.com/pub/semiconductor/motion/hctl2000.pdf

[Appl 99] Apple Computer, Inc.
Object–Oriented Programming and the Objective–C Language
Apple Developer's Library/Technical Publications (1999)
http://developer.apple.com/techpubs/macosx/Cocoa/ObjectiveC/ObjC.pdf

[Appl 01] Apple Computer, Inc.
Foundation Objective-C API Reference
Apple Developer's Library/Technical Publications (2001)
http://developer.apple.com/techpubs/macosx/Cocoa/Reference/Foundation/ObjC_classic/FoundationTOC.html

[Ashe 99] S. Ashe
Library and Utilities for SBIG ST7/ST8/ST5C/ST237
version 2.0 (7 June 1999)
ftp://ftp.dimensional.com/users/ashe/sbig-linux-2.0.tgz

[Bern 00] K. Bernlöhr
hcomm (comm_threads) Reference Manual
MPI für Kernphysik, Heidelberg (Nov 21 2000)
http://www.mpi-hd.mpg.de/hfm/~bernlohr/HESS/Software/hcomm/doxygen/latex/refman.pdf

[Davi 57] J. M. Davies u. E. Cotton
Design of the Quartermaster Solar Furnance
Journal for Energy Sience and Engineering, **1** (1957) 16–22

[Elte 99] ELTEC Elektronik AG, Mainz
EUROCOM 138 Hardware Manual
Revision 1C (1999)
http://www.eltec.de/d/index.htm

[Gill 99] S. Gillessen
Überwachung der Abbildung eines Chrenkov–Teleskops und automatische Spiegeljustierung mit einer CCD–Kamera, Diplomarbeit
MPI für Kernphysik, Heidelberg (1999)

[GNUs 01] The GNUstep Project
About GNUstep
http://www.gnustep.org/information/aboutGNUstep.html

[Hann 00] J. Hannappel
Linux driver for the Tundra Semiconductor Universe PCI/VME bridge
http://lisa2.physik.uni-bonn.de/~hannappe/software/universe_doc/universe.html

[Hart 99] R. C. Hartmann et al.
The Third EGRET Catalog of High–Energy Gamma–Ray Sources
Astrophysical Journal Supplement Series, **524** (1999) 91–94

[HESS 00] Die H·E·S·S· Kollaboration
Ein System abbildender atmosphärischer Cherenkov–Teleskope zur stereoskopischen Beobachtung von gamma–induzierten Luftschauern im Energiebereich um 100 GeV
Allgemeine Projektübersicht (August 2000)

[Hess 12] V. F. Hess
Observation of Penetrating Radiation of seven Ballon Flights
Physikalische Zeitschrift, **13** (1912)

[Hofm 98] W. Hofmann
Alignment of the HESS mirrors using images of stars
Interne Mitteilung, MPI für Kernphysik, Heidelberg (1998)

[Horn 00] D.Horns
Suche nach TeV–Photonen aus intergalaktischen Kaskaden und nach Bose–Einstein–Kondensaten in TeV–Photonen, Dissertation
II. Institut für Experimentalphysik, Universität Hamburg (Dezember 2000)

[Jage 96] O. C. de Jager et al.
Gamma–Ray Observations of the Crab Nebula: A Stydy of the Synchrotron–Compton Spectrum
Astrophysical Journal, **457** (1996) 253–266

[Jage 00] O. C. de Jager et al.
Prospects of observing pulsed radiation from gamma–ray pulsars with H.E.S.S.
erscheint in: Proceedings of the Heidelberg International Symposium on High Energy Gamma-Ray Astronomy, Heidelberg (2000)
http://de.arxiv.org/abs/astro-ph/0010179

[Kato 99] A. A. A. Katona
Automatische Justierung der Spiegel der HESS–Teleskope, Diplomarbeit
MPI für Kernphysik, Heidelberg (1999)

[Kono 99] A. Konopelko et al. (The HEGRA Collaboration)
Performance of the stereoscopic system of the HEGRA imaging air Cherenkov telescopes: Monte Carlo simulations and observations
Astroparticle Physics, **10** (1999) 275–289

[Kraw 00] H. Krawczynski, A. Kohle, G. Heinzelmann u. H. Völk
Der Kosmos im Licht von Gamma–Strahlung sehr hoher Energie
Physikalische Blätter, **56** (2000) 3, 47–52

[Kunz 00] T. Kunzl, A. Jessner u. H. Lesch
Gleichschritt im All, Wie entsteht die Radiostrahlung von Pulsaren?
Sterne und Weltraum, **39** (11/2000) 11, 936–944

[Long 92] M. S. Longair
High Energy Astrophysics, 2nd ed., Vol. 1: Particles, photons and their detection
Cambridge University Press (1992)

[Long 94] M. S. Longair
High Energy Astrophysics, 2nd ed., Vol. 2: Stars, the Galaxy and interstellar medium
Cambridge University Press (1994)

[Lyn 93] A. G. Lyn u. F. Graham–Smith
Pulsare
Hüthig Verlag, Heidelberg (1993)

[Next 94] NeXT Computer, Inc.
OpenStep Specification (1994)
http://www.gnustep.org/resources/OpenStepSpec.pdf.gz

[OMG 00] The Object Management Group
The Common Object Request Broker: Architecture and Specification
Minor Editorial Revision: CORBA 2.4.1 (November 2000)
ftp://ftp.omg.org/pub/docs/formal/00-11-03.pdf

[Röhr 00] A. H. Röhring
Bestimmung des Energiespektrums und der chemischen Zusammensetzung der kosmischen Strahlung im Energiebereich von $3 \cdot 10^{14}$ *eV bis* 10^{16} *eV aus der Messung der elektromagnetischen Komponente ausgedehnter Luftschauer mit dem HEGRA–Experiment*, Dissertation
II. Institut für Experimentalphysik, Universität Hamburg (Mai 2000)

[Sche 90] H. Scheffler u. H. Elsässer
Physik der Sterne und der Sonne, 2., überarbeitete und erweiterte Auflage
B.I. Wissenschaftsverlag (1990)

[Stev 93] W. R. Stevens
Advanced Programming in the UNIX Environment
Addison–Wesley (1993)

[Stev 98] W. R. Stevens
UNIX Network Programming Vol. 1 – Networking APIs: Sockets and XTI, 2nd ed.
Prentice Hall (1998)

[Stev 99] W. R. Stevens
UNIX Network Programming Vol. 2 – Interprocess Communications, 2nd ed.
Prentice Hall (1999)

[TEB 00] H. Riege, J. Schütt u. R. van Staa
HESS Mirror Motor Control Manual
Interne Mitteilung, II. Institut für Experimentalphysik / Teschnische Entwicklung und Betrieb, Universität Hamburg (August 2000)
http://www.desy.de/~biskop/teb/Hess_Manual.pdf

[Tluc 01] M. Tluczykont
Knotenpositionen bei halbsegmentigem Kabelverlauf
Interne Studie, II. Institut für Experimentalphysik, Universität Hamburg (Januar 2001)

[Tota 00] T. Totani
An Interpretation of the Evidence for TeV Emission from GRB 970417a
The Astrophysical Journal Letters, **536** (2000) L23–L26

[Völk 99a] H. Völk
Gamma–Astronomie mit abbildenden Cherenkov–Teleskopen, Teil 1: Astronomische und Physikalische Grundlagen
Sterne und Weltraum, **38** (11/1999) 11, 948–953

[Völk 99b] H. Völk
Gamma–Astronomie mit abbildenden Cherenkov–Teleskopen, Teil 2: Erste Ergebnisse und Pläne für die Zukunft
Sterne und Weltraum, **38** (12/1999) 12, 1064–1070

[VME 93] W. D. Peterson
The VMEbus Handbook, expanded Third Edition
VMEbus International Trade Association (Publisher), Scottsdale, Arizona/USA (1993)

[Week 99] T. C. Weekes
VHE Astronomy before the New Millennium
Proceedings of the GeV–TeV Gamma Ray Workshop,
Snowbird (1999) 3–15

[Wieb 98] B. Wiebel–Sooth u. P. L. Biermann
Landolt–Börnstein Band VI 3C aus *New Series*
Kapitel 7.6: Cosmic Rays, Seiten 37–90, Springer (1998). In Druck.

Danksagung

Mein besonderer Dank gilt Prof. Dr. Götz Heinzelmann für die interessante Aufgabenstellung und die ausgezeichnete Betreuung.

Für die großartige Unterstützung und die ausgesprochen nette Arbeitsatmosphäre habe ich der Hamburger HEGRA–/H·E·S·S·–Gruppe zu danken: Matthias Beilicke, Konrad Bernlöhr, Niels Götting, Dieter Horns, Michael Raabe, Joachim Ripken, Jan Robrade, André Röhring und Martin Tluczykont.

Des weiteren habe ich Herrn Dipl. Ing. Harald Riege für die gute und sehr fruchtbare Zusammenarbeit zu danken. Additionally I would like to acknowledge the kind assistance of Richard Frith–Macdonald.

Zudem sind viele Personen mit verantwortlich dafür, daß die Zeit meines Studiums zu den bisher schönsten Zeiten meines Lebens gehört. Ich möchte von dem Versuch absehen, hier alle namentlich zu nennen. Ihre Namen werden indes nicht vergessen.

Erklärung:
Ich versichere, daß ich die Arbeit selbständig verfaßt und keine als die angegebenen Quellen und Hilfsmittel verwendet habe.

Hamburg, den René Cornils