Integrationslösung zur Absicherung eines realen Radarsensors im Systemverbund mit der Hardware-in-the-Loop Testtechnologie

Marco Weiskopf, Christoph Wohlfahrt

RD/EDT
Daimler AG
HPC G025-BB
71059 Sindelfingen
marco.weiskopf@daimler.com
christoph.wohlfahrt@daimler.com

Dr. Albrecht Schmidt

Institute for Visualization and
Interactive Systems (VIS)
Universität Stuttgart
70569 Stuttgart
albrecht.schmidt@vis.uni-stuttgart.de

Abstract: Aufgrund der steigenden Komplexität, Anzahl und Vernetzung der Fahrerassistenzsysteme wächst die Notwendigkeit einer ausreichenden Absicherung in Bezug auf Zuverlässigkeit und Sicherheit. Dabei kommt die Hardware-in-the-Loop (HiL) Testtechnologie zum Einsatz, die automatisierte Tests in Echtzeit mit der realen Steuergeräte-Hardware ermöglicht. Bei der Durchführung von Systemtests für Fahrerassistenzsysteme am HiL-Prüfstand wird es immer wichtiger alle Komponenten wie z.B. Radarsensoren zu integrieren um eine möglichst realitätsnahe Testaussage zu bekommen. Viele Steuergeräte erfassen die Umgebung und benötigen ein gemeinsames Fahrzeugumfeld. Hierfür wird eine realistische 3D-Animation (Virtuelle Welt) verwendet. Dieser Beitrag zeigt zum einen, wie aus der virtuellen Welt relevante Daten für den Radarsensor gewonnen werden können. Dabei spielen Materialeigenschaften, Berechnung von Reflexionen, realitätsnahe Generierung, Interpolation von Detektionen und Echtzeitfähigkeit eine bedeutende Rolle. Zum anderen beschreibt dieser Beitrag eine echtzeitfähige Lösung zur Einspeisung der relevanten Daten in den Radarsensor.

1 Einleitung

Eine lange monotone Autobahnfahrt kann dazu führen, dass die Konzentrationsfähigkeit des Fahrers sinkt. Hierbei unterstützen Fahrerassistenzsysteme, indem sie kontinuierlich und unermüdlich die Fahrzeugumgebung nach Gefahrensituationen analysieren und falls erforderlich in die Fahrzeugdynamik eingreifen. Dabei kommen hochkomplexe Erkennungs- und Klassifikationsalgorithmen zum Einsatz, die verteilt auf verschiedenen Steuergeräten die Umgebung und den Verkehr beobachten und falls erforderlich die Fahrdynamik beeinflussen (siehe Abbildung 1). Diese Algorithmen dienen für Fahrerassistenzsysteme wie z.B. Abstandregeltempomat, Notbremsassistent und Ausweichassistent.

In Bezug auf die Sicherheit sind Eingriffe in die Fahrdynamik kritisch zu bewerten. Deswegen wird bei Daimler zur Absicherung dieser softwarebasierten

Fahrzeugfunktionen eine konkrete Teststrategie festgelegt. Ein wesentlicher und bedeutender Bestandteil dieser Teststrategie sind HiL-Tests auf Komponenten-, System- und Fahrzeugebene. Diese ermöglichen bereits in frühen Entwicklungsphasen ein reproduzierbares Testen im Labor

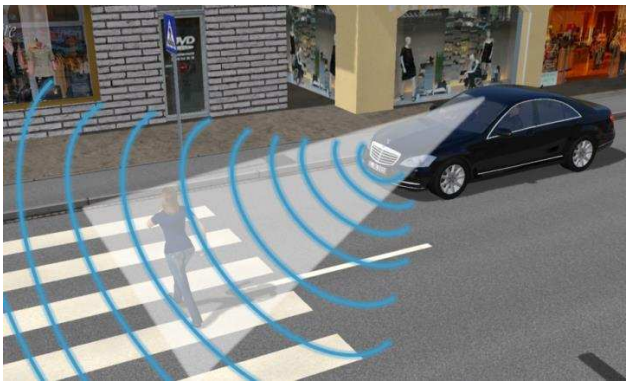

Abbildung 1: Fußgängererkennung mittels Radar (Halbkreise) und Kamera(Dreieck)

Um diesen Vorteil der HiL-Test zu nutzen, ist es notwendig alle am zu testenden System beteiligten Komponenten an das HiL-Testsystem anzubinden. Die Anbindung kann dabei auf zwei Arten erfolgen. Es wird das reale Steuergerät verwendet oder eine Restbussimulation. Erstrebenswert ist es, immer die realen Steuergeräte zu verwenden um eine möglichst realitätsnahe Testaussage zu bekommen. Momentan ist es durch die Arbeit von [WWS10] möglich, die hier betrachteten Fahrerassistenzsysteme auf der System-Teststufe mit einer Stereo-Kamera als reales Steuergerät abzusichern. Dazu wurden sämtliche Sensoren und Aktoren wie Stereokamera, Fahrerassistenzsteuergerät, Radarsensor, Bremse und Lenkung in das HiL-Testsystem integriert. Die Anbindung des Radarsensors erfolgte dabei über eine Restbussimulation, welche durch diesen Beitrag durch die Anbindung des realen Radar-Steuergerätes ersetzt werden soll. Für die umgebungserfassenden Sensoren wird ein zentrales Simulationsmodell (virtuelle Welt) verwendet, zur Bereitstellung einer gemeinsamen Umgebungsbasis. Über dieses Simulationsmodell können komplexe Verkehrssituationen in Echtzeit simuliert und z.B. für die Stereo-Kamera als Videopfad zur Verfügung gestellt werden.

In diesem Beitrag wird anhand einer neuartigen Integrationsstrategie kurz veranschaulicht, wie einem Radarsteuergerät ebenfalls Daten aus dem zentralen Simulationsmodell bereitgestellt werden.

2 Verwandte Arbeiten

Dieser Abschnitt zeigt bereits vorhandene Ansätze zur Simulation von Radardetektionen auf und gibt eine Einordnung in den Kontext des vorliegenden Beitrags. Die Idee zur Simulation von Radarstrahlen, dass aus der Computergrafik bekannte Ray Tracing Verfahren von [Ka86] zu verwenden, wurde erstmals von [Ba96] vorgestellt. Durch den hohen Berechnungsaufwand auf der Central Processing Unit (CPU) eignete sich diese

Simulation nicht für den Praxiseinsatz. Um dennoch eine echtzeitfähige Radarsimulation zu Entwicklungs- und Testzwecken verwenden zu können, entwickelte [BB06] ein Verfahren zur Erzeugung von realistischen Radarziellisten. Während der Entwicklung analysierte der Autor aufgezeichnete Radarziellisten eines Radarsensors im Auto und ermittelte Punkte, die mit hoher Wahrscheinlichkeit die Radarsignale zurück reflektieren. Basierend auf den ermittelten Punkten definierte er ein Objektmodell mit Reflexionspunkten für ein Fahrzeug. Mittels einer spezifischen Simulationsumgebung erzeugte er mit Hilfe des Objektmodells eine ideale Zielliste, die durch künstliches Rauschen modifiziert wird. Ziel dieser Arbeit ist es, realistische Radarziellisten zur Unterstützung der Entwicklung von Verarbeitungsalgorithmen für Radarziellisten zu erzeugen.

Der Nachteil dieses Verfahrens liegt in der Erweiterung des Objekt-Repertoires. Denn für jedes weitere Objekt (wie z.B. Häuser, Motoräder, Fahrräder, Fußgänger usw.) muss ein spezifisches Objektmodell erzeugt werden. Für dieses Objektmodell ist wiederum eine aufwendige Analyse von aufgezeichneten Radarziellisten notwendig.

Zur Verwendung eines generischen Verfahrens für alle Objekte stellte [PLK08] einen Ansatz vor, indem er eine shader-basierte Simulation von Radardetektionen verwendet. [Sh12] optimierte die shader-basierte Simulation um diese für einen Software-in-the-loop (SIL) Testsystem einzusetzen. Für eine shader-basierte Simulation wird die Möglichkeit, eigene Shader-Programme in die OpenGL Rendering Pipeline einzuspeisen, ausgenutzt. Dadurch wird die Berechnung von der CPU auf die Graphics Processing Unit (GPU) verlagert und damit eine schnellere Berechnung ermöglicht. [Sh12] demonstriert, dass mit diesem Ansatz eine Berechnung der Radardetektionen in Echtzeit möglich ist. Die Möglichkeit, dass Radarstrahlen mehrfach reflektiert werden können ist in [Sh12] nicht berücksichtigt worden.

Um dies zu realisieren, wird wieder auf das eingangs erwähnte Ray Tracing Verfahren zurückgegriffen. In [Pa10] wird gezeigt, dass es möglich ist, mittels des Frameworks OPTIX, Ray Tracing in Echtzeit zu berechnen. Dabei wird die schnelle parallelisierte Berechnung auf der GPU für das Ray Tracing ausgenutzt.

Optix wird für die Generierung der Radar Detektionen innerhalb der hier betrachteten 3D Animation eingesetzt.

3 Bestehendes HiL-Testsystem für Fahrerassistenzsysteme

Zur Systemabsicherung von Fahrerassistenzsystemen existiert bereits ein HiL-Testsystem, dass die Stereokamera als reale Komponente integriert. Die Radarsensoren werden durch ein Modell simuliert [WWS10].

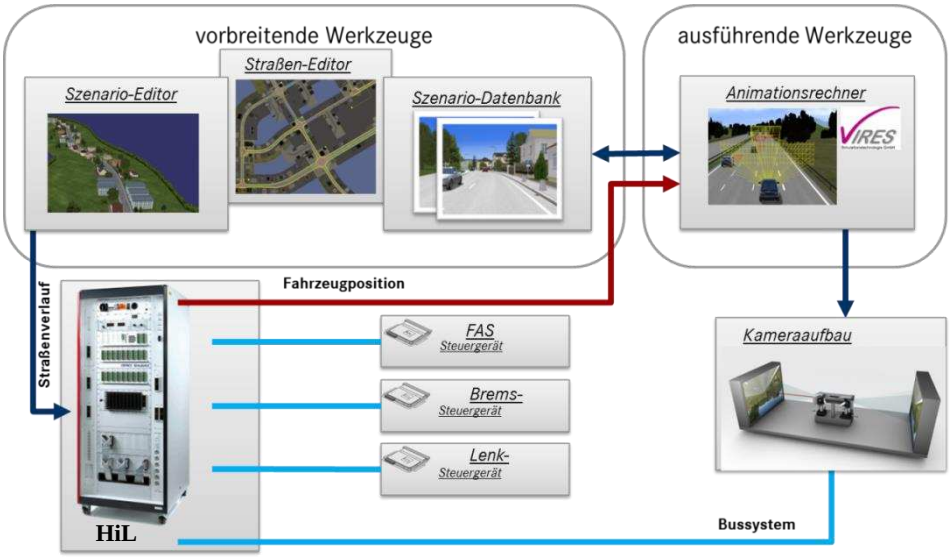

Abbildung 2: Architektur des bestehenden HiL-Testsystems für Fahrerassitenzsysteme

Für das bestehende HiL Testsystem für Fahrerassistenzsysteme definierte [WWS10] die in Abbildung 2 veranschaulichte Architektur. Das Ego-Fahrzeug wird im Echtzeitrechner des HiL-Simulators mit einer detaillierten Fahrdynamiksimulation simuliert. Über die Bus- und IO-Schnittstellen stellt der HiL-Simulator die Signale den angebundenen Steuergeräten zur Verfügung. Bei dieser Architektur wurde das klassische HiL-System durch einen Animations- und Simulationsrechner erweitert, welcher die Video-Daten für die am System beteiligte Stereo-Kamera generiert. Mittels der vorbereitenden Werkzeuge im Animationsrechner können verschiedene Szenarien definiert und in der Simulation ausgeführt werden. Über ein Netzwerkprotokoll wird der HiL-Simulator und der Animationsrechner synchronisiert.

Damit alle beteiligten Komponenten des Fahrerassistenzsystems am HiL möglichst realitätsnah abgesichert werden können, müssen in das bestehende HiL-System die Radarsensoren integriert werden. Um eine gemeinsame Umgebungsbasis mit der Stereo-Kamera zu besitzen, ist es zwingend notwendig, die Radarsensoren mit Daten aus der Simulation des Animationsrechners zu speisen. Weiterhin muss eine Design-for-Test (DfT) Schnittstelle definiert werden, die diese Daten dem Radarsensor bereitstellt.

4 Einspeisungsschnittstelle

Zur Spezifizierung einer DFT-Schnittstelle für den Radarsensor werden die wesentlichen Verarbeitungsschritte eines Radarsensors in Abbildung 3 skizziert.

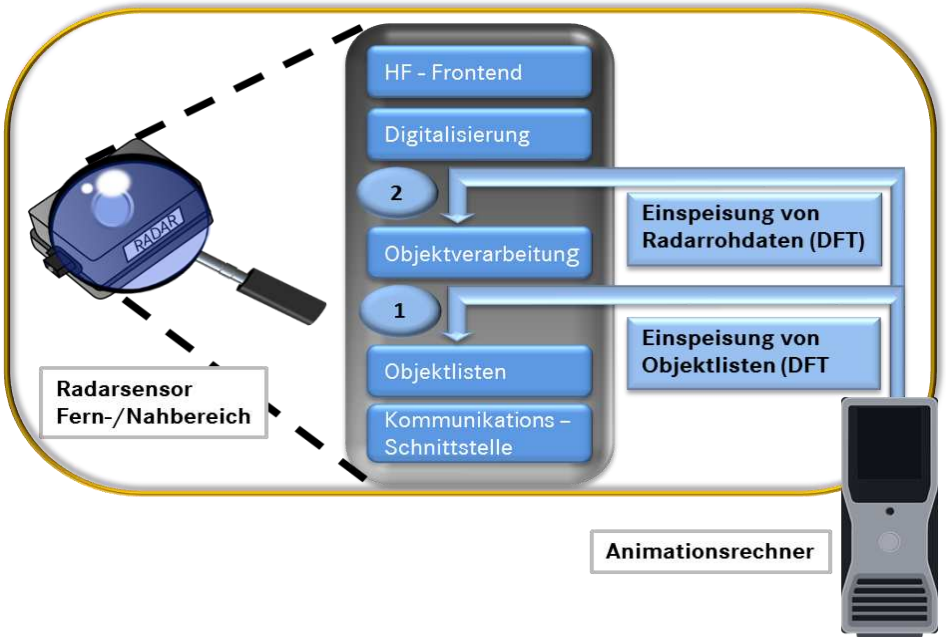

Abbildung 3: DFT-Schnittstellen

Theoretisch kann vor jedem Verarbeitungsschritt eine Einspeisung erfolgen. Wie in Abbildung 3 zu sehen ist, wurden zwei sinnvolle Möglichkeiten für die Einspeisung identifiziert. Diese befinden sich in Punkt 1, nach dem Verarbeitungsschritt Objektverarbeitung und in Punkt 2, nach der Digitalisierung der Radar-Echos.

Bei der Einspeisung von Objektlisten kann nur die Funktionalität der Kommunikationsschnittstelle des Radarsensors getestet werden. Um die Testaussage zu erhöhen und um möglichst viel Funktionalität testen zu können, ist das Ziel, die Daten nach der Digitalisierung einzuspeisen. Deshalb wurde die Design-for-Test (DFT) Schnittstelle für den Einspeisungspunkt 2 definiert.

Durch nähere Erläuterung der einzelnen Verarbeitungsschritte wird deutlich, in welcher Art die Daten nach der Digitalisierung vorliegen. Im HF-Frontend werden elektromagnetische Wellen mit Lichtgeschwindigkeit ausgesendet und während einer Sendepause empfangen. Diese empfangenen Wellen werden von der Digitalisierung abgetastet und die Informationen als Frequenzen abgespeichert. Dementsprechend müssen diese Frequenzen vom Animationsrechner bereitgestellt werden.

Zur Erzeugung dieser Frequenzen zeigt Abbildung 4 eine ausführlichere Systemarchitektur des HiL-Testsystems. Hier werden die Schnittstellen zwischen den drei beteiligten Komponenten HiL, Animations-PC und HiL-Box veranschaulicht. Der HiL simuliert, wie bereits im vorangegangen Kapitel beschrieben, das Ego-Fahrzeug und kommuniziert dessen Position über ein Netzwerkprotokoll an den Animationsrechner(2). Weiterhin werden über dieses Netzwerkprotokoll zusätzliche Steuer- und Simulationsdaten ausgetauscht. Basierend auf den empfangenden Positionsdaten

aktualisiert der Animationsrechner die Umgebung und stellt die aktualisierten Daten den am System beteiligten umgebungserfassenden Sensoren zur Verfügung.

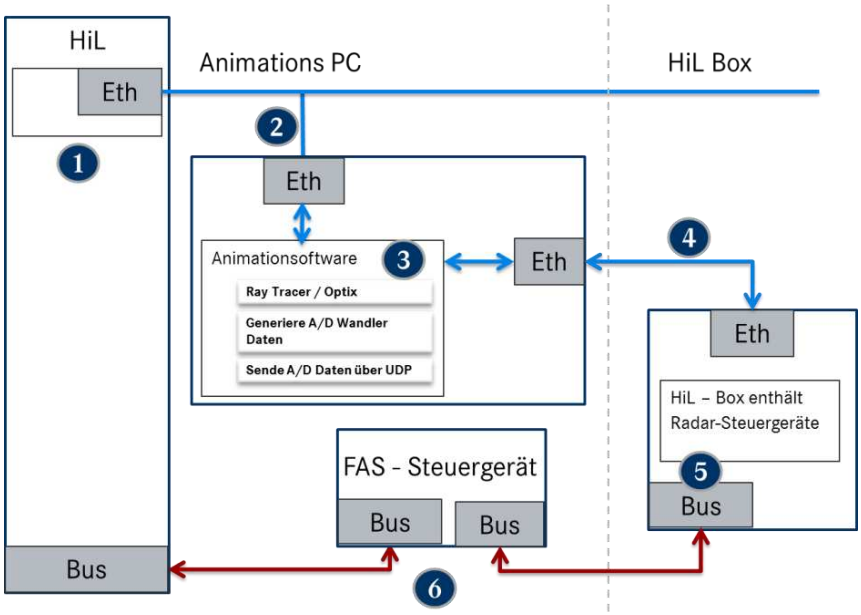

Abbildung 4: Detaillierte Systemarchitektur des HiL-Testsystems

Während der Aktualisierung der Umgebung arbeitet auch der Raytracer (3), welcher die Strahlen simuliert, die vom Radarsensor ausgesendet werden würden. Die erzeugten digitalen Daten müssen dann durch einen Frequenzgenerator in Frequenzdaten umgewandelt werden. Diese Frequenzdaten werden über UDP an die HiL Box (4) geschickt. In der HiL-Box sind die realen Radarsensoren mit einer DFT-Schnittstelle versehen über welche die empfangen Frequenzdaten als Radar-Detektionen in den A/D Wandler Speicher geschrieben werden. Dann beginnt das Radar-Steuergerät auf den eingespeisten Daten seine Erkennungs- und Klassifikationsalgorithmen auszuführen und stellt abschließend eine Objekteliste auf dem Fahrzeugbus bereit. Diese Objektliste wird vom FAS-Steuergerät entgegengenommen und weiterverarbeitet.

5 Erzeugung und Interpolation virtueller Radar Detektionen

Zur Simulation von Radar Detektionen in einer Echtzeit 3D Animation wird das Raytracing-Verfahren eingesetzt. Beim Raytracing-Verfahren (Abbildung 5) wird für jeden Pixel des Ausgangsbildes ein Primärstrahl in die 3D Szene geschossen. Diese Primärstrahlen treffen analog zu den Radarstrahlen auf Objekte in der Szene. Wie in Abbildung 5 veranschaulicht, trifft ein Primärstrahl z.B. auf eine halbtransparente (halbtransparent bezogen auf die elektromagnetischen Materialeigenschaften) Kugel. Basierend auf den Materialeigenschaften, dem Einfallswinkel, der Geometrie des Objekts und der Oberflächenbeschaffenheit berechnet der Raytracer Sekundärstrahlen in

Form von Transmissions- und Reflexionsstrahlen. Diese Sekundärstrahlen werden ebenfalls in die Szene geschossen und können wieder auf weitere Objekte treffen. Dieser Vorgang kann rekursiv wiederholt werden bis der Strahl die Szene verlässt oder ein definiertes Abbruchkriterium wie z.B. die Anzahl an Iterationen erreicht wurde. Das Ausgangsbild (Umgebungs-Scan) hat die Dimension des Sichtbereiches des Radarsensors und enthält die Auftreffpunkte (Detektionen) der Strahlen in Richtung des Radarsensors.

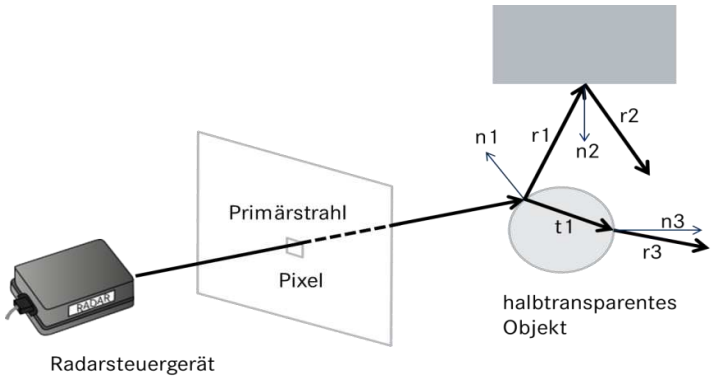

Abbildung 5: Funktionsweise Raytracing

Die benötigten Parameter wie Reflexions- und Transmissionsfaktor werden für jede Textur aus einer XML-Datenbank geladen. Weiterhin enthält die XML-Datenbank Radarquerschnitte für die in der 3D-Animation verwendeten Objekte wie Autos, Häuser, Straßenbelag, Türen, usw… Diese Parameter basieren auf gemessen Daten mit dem realen Radarsensor. Mithilfe dieser Parameter kann die Intensität der Sekundärstrahlen beeinflusst werden, die direkte Auswirkungen auf den Radarquerschnitt der Detektion hat. Sinkt die Intensität eines Strahles unter einen bestimmten Schwellwert, kann die Nachverfolgung auf Grund der geringen Auswirkungen auf das Ausgangsbild abgebrochen werden. Das erhaltene Ausgangsbild ist für den verwendeten Sensor viel zu exakt und muss deshalb durch real gemessene Rauschwerte modifiziert werden.

Obwohl es gelungen ist, das Raytraceing zur Simulation der Radarstrahlen in Echtzeit (60 Hz) zu berechnen, reicht diese Geschwindigkeit zur vollständigen Simulation der Radarstrahlen nicht aus. Zum Vergleich berechnet der Radarsensor ein Umgebungs-Scan ähnlich dem Raytracer in weniger als 1 ms. Weiterhin führt er mehrere Umgebungs-Scans hintereinander aus, sodass er bereits 16-mal die Umgebung erfasst hat, bis der Raytracer, den ersten Umgebungsscan geliefert hat. Hinzu kommt, dass der Radarsensor mit zwei unterschiedlichen Modi arbeitet, die in Bezug auf Reichweite, Öffnungswinkel und Abtastungsraster differieren. Zusätzlich werden noch mehrere Antennen eingesetzt zum Aussenden und Empfangen der Radarstrahlen.

Um dennoch den Hardware Radarsensor am HiL zu testen, muss ein Interpolation-Algorithmus eingesetzt werden, der die fehlenden Umgebungs-Scan interpoliert, die verschiedenen Modi berechnet und die Detektionen unter Berücksichtigung des Doppler Effekts auf die einzelnen Empfangsantennen aufteilt.

Der Raytracer kann dementsprechend parametriert werden, dass er ein Ausgangsbild für beide Modi liefert oder separat für jeden Modus. Anschließend werden die Daten konvertiert um performant über das Netzwerk übertragen zu werden und um die Einspeisung beim Empfang durch die HiL-Box zu erleichtern. Die HiL-Box speist die empfangen Detektionen direkt in den A/D Wandler Speicher ein.

Als Raytracer wird das bereits im Abschnitt „Verwandte Arbeiten“ erwähnte echtzeitfähige Raytracing Framework Optix verwendet.

6 Zeitliche Evaluation für die gesamte Integrationskette

Laut Definition läuft ein HiL-Testsystem immer in Echtzeit. Daraus resultiert, dass die Integrationslösung des Radarsensors diese Anforderung erfüllen muss. Kann die Echtzeit nicht eingehalten werden, hätte das erhebliche Auswirkungen auf die Fusion der Daten. Durch die Verletzung der Echtzeit, wird das Objekt nicht rechtzeitig auf dem Fahrzeugbus bereitgestellt und deshalb ist eine Fusion der Daten nicht mehr möglich. Zusätzlich liest das Radarsteuergerät Daten auf dem Fahrzeugbus mit und überprüft diese durch interne Plausibilitätskontrollen mit den erfassten Daten. Diese Kontrollen würden ebenfalls fehlschlagen.

Aufgrund dessen muss eine exakte Untersuchung des Zeitverhaltens von der Aktualisierung der Umgebung bis hin zur Bereitstellung der Daten auf dem Fahrzeugbus durchgeführt werden.

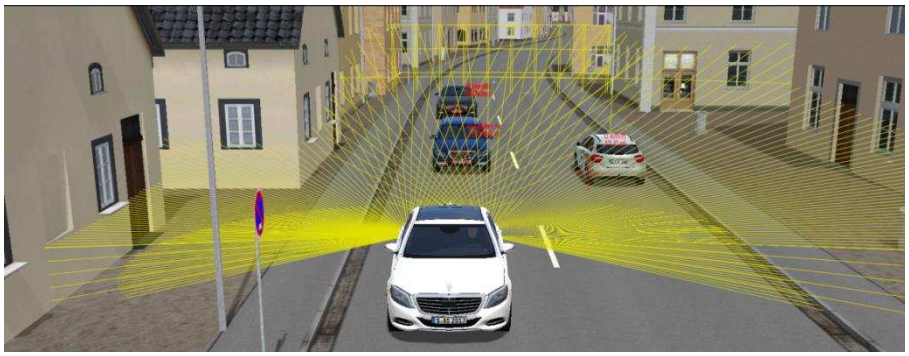

Abbildung 6: Veranschaulichung des Test-Szenarios

Für die Untersuchung wird ein einfaches Test-Szenario mit drei Fahrzeugen verwendet. Das vordere weiße Fahrzeug in der Abbildung 6 repräsentiert das Testfahrzeug (Ego Fahrzeug), dass die zu testenden Sensoren enthält. Bei diesem Szenario führt, das Ego Fahrzeug eine Notbremsung durch, aufgrund eines Fußgängers der in die Straße läuft. Im Diagramm auf der linken Seite der Abbildung 7 sind die Positionen der beteiligten Fahrzeuge veranschaulicht. Der Punkt (0,0) zeigt die Position des Ego-Fahrzeuges. Ausgehend vom Ego-Fahrzeug sind in der gleichen Fahrspur noch zwei weitere Fahrzeuge positioniert. Weiterhin ist ein entgegenkommendes Fahrzeug (-15,-1,2) definiert.

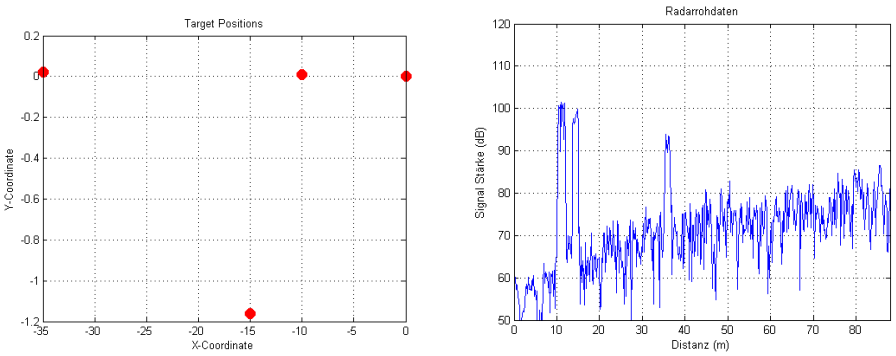

Abbildung 7: Objekt Positionen und Generierte Radar A/D Wandler Daten

Dieses Szenario wird am HiL-Testsystem geladen und ausgeführt. Mittels des Raytracing Verfahrens werden die Radar-Detektionen ermittelt. Diese Detektionen werden über den Interpolations-Algorithmus in Frequenzdaten für den A/D Wandler Speicher konvertiert. Die rechte Seite in Abbildung 7 veranschaulicht die konvertierten Daten. Zu sehen sind die drei Objekte als Peak zu den Distanzen 10, 15 und 35 Meter. Diese Daten werden über eine Netzwerkschnittstelle an die HiL-Box geschickt und direkt in den A/D Wandler Speicher geschrieben.

Angesichts des enormen Zeitverlusts des Interpolations-Algorithmus wurde dieser für mehrere Detektionen untersucht um festzustellen, wie der Algorithmus skaliert. Abbildung 8 dokumentiert die zeitliche Untersuchung für den sequentiellen und parallelen Ansatz. Hier ist zu sehen, dass sich die Anzahl der Detektionen zur benötigten Zeit linear verhalten. Aufgrund der hohen Komplexität des Interpolations-Algorithmus, stellt die Parallelisierung eine große Herausforderung dar. Dabei besteht die Komplexität aus der Verteilung der Detektionen über mehrere Antennen unter Berücksichtigung mehrerer Umgebungsscans und aus der Verarbeitung der Detektionsparameter wie Position, Geschwindigkeit und Radarquerschnitt. Nach einer intensiven Untersuchung auf Parallelisierung gelang es, einzelne Verarbeitungsschritte herauszulösen und in Threads auszulagern. Durch die parallele Ausführung der Thread konnte die Performance deutlich gesteigert werden. Die Auffälligkeit, dass die maximale Performance der Parallelisierung erst ab Detektion 24 erreicht wird, ist auf die Implementierung zur Aufteilung auf die einzelnen Antennen zurückzuführen. Die Auslegung des Algorithmus war für eine große Anzahl an Detektionen geplant. Trotz des positiven Ergebnisses erfüllt der Interpolations-Algorithmus das vorgegebene Kriterium nicht und ist für eine echtzeitfähige Umsetzung nicht geeignet. Das Kriterium ist, die Ausführungszeit des Interpolations-Algorithmus für eine definierte Anzahl an Detektionen unter der Zykluszeit des Radarsensors zu halten. Die Zykluszeit des Radarsensors umfasst die Umgebungsscans, die Signalaufbereitung, die Objekterkennung und -klassifikation sowie die Ausgabe der Objektliste auf dem Fahrzeugbus. Sie ist definiert als ein konkretes Zeitintervall, in dem diese Aufgaben bearbeitet werden müssen.

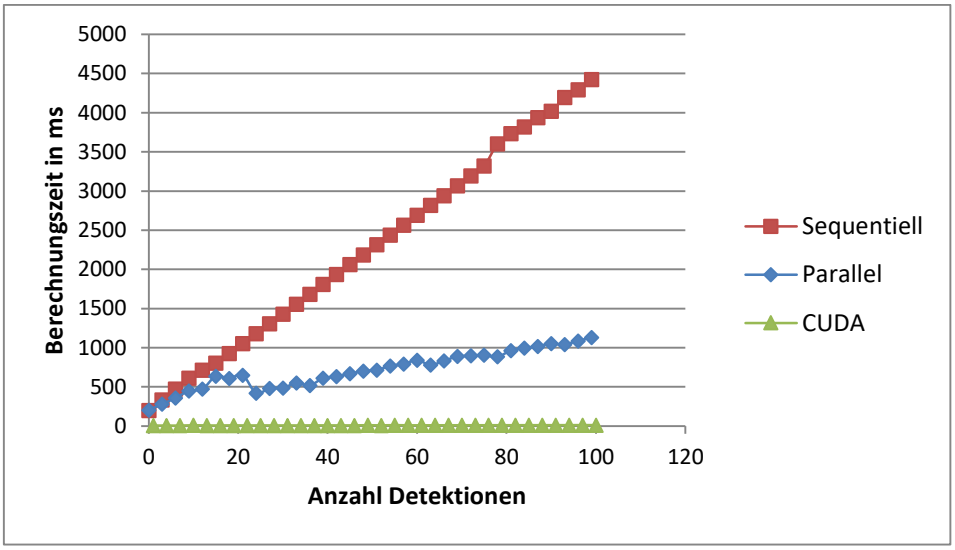

Abbildung 8: Berechnungszeit des Interpolations-Algorithmus

Aufgrund der Verletzung der Zykluszeit musste der Interpolations-Algorithmus weiter parallelisiert, separiert und portiert werden, damit er mittels des Frameworks CUDA auf der Grafikkarte hoch parallelisiert zur Ausführung gebracht werden konnte. Dies gelang mit der Grafikkarte „GFX 480“ für einige Detektionen. Durch die Verwendung einer leistungsfähigeren Grafikkarte „K6000“ konnte die Anzahl der Detektionen gesteigert und der Toleranzbereich für die Ausführung des Interpolations-Algorithmus eingehalten werden(siehe Abbildung 9). Damit ist es möglich, die in Abbildung 4 skizzierte Einspeisungskette innerhalb der Zykluszeit des Radarsensors auszuführen. Dies bedeutete, dass die Berechnung des Raytracers mit Reflexions- und Transmissionsstrahlen, die Konvertierung und Interpolation durch den Interpolations-Algorithmus, die Übertragung zur HiL-Box sowie die Einspeisung in den A/D Wandler Speicher innerhalb eines Zyklus erfolgt. Durch die Einhaltung der Zykluszeit ist der Interpolations-Algorithmus für eine echtzeitfähige Umsetzung geeignet.

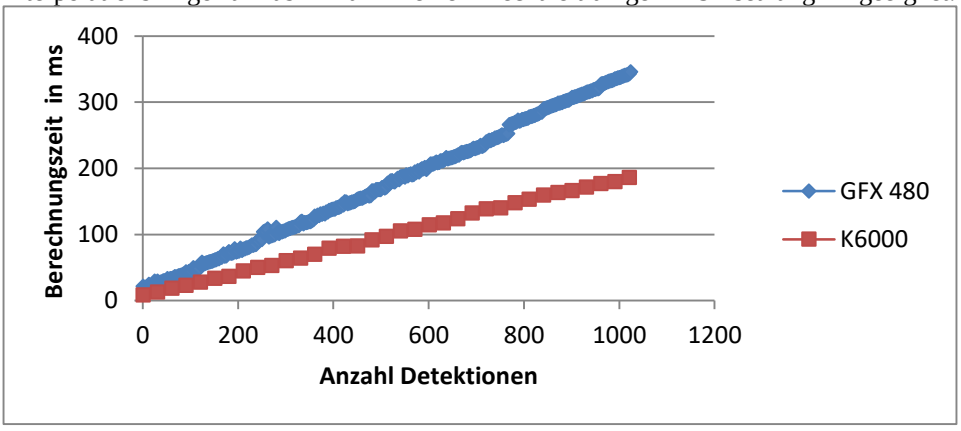

Abbildung 9: Berechnungszeit des CUDA optimierten Interpolations-Algorithmus

7 Validierung der Gesamtkette

Im vorangegangenen Kapitel wurde gezeigt, dass es aus zeitlicher Sicht möglich ist, virtuelle Radardetektionen zu generieren und rechtzeitig in den realen Radarsensor einzuspeisen. Nach der Einspeisung in den Sensor führt dieser Erkennungs- und Klassifikationsalgorithmen aus und liefert abschließend auf dem Fahrzeugbus eine Objekteliste mit relevanten Objekten. Im Zug der Validierung wurde das für die Zeitmessung verwendete Testszenario in der realen Welt nachgestellt. Zu diesem realen Testszenario wurden die Objektlisten aufgezeichnet, welche der Radarsensor über den Fahrzeugbus sendet. Diese aufgezeichneten Objektlisten werden mit den Objektlisten basierend auf den virtuell eingespeisten Detektionen verglichen. Dadurch kann validiert werden, ob diese Listen identisch sind. Stellt sich heraus, dass die Listen nach wiederholtem Vergleichen immer noch kongruent sind, kann für dieses Szenario behauptet werden, dass es im Labor ausreichend abgesichert werden kann. Um eine generelle Aussage zur ausreichenden Absicherung im Labor treffen zu können, müssen noch weitere verschiedene Testszenarien mit den identischen Testszenarien im Labor verglichen werden. Eine konkrete Validierung folgt.

8 Fazit

In diesem Beitrag wurde ein neuwertiger Lösungsansatz zur Einspeisung von virtuellen Radardetektionen in einen realen Radarsensor in Echtzeit unter den gegebenen Rahmenbedingungen präsentiert. Somit können Testszenarien mit höchsten 5 beteiligten Objekten im Labor unter HiL Bedingungen getestet werden. Der eingesetzte Interpolations-Algorithmus lässt sich auf mehrere Grafikkarten aufteilen. Dadurch ergibt sich für zukünftige Untersuchungen die Möglichkeit, diesen Aspekt zu untersuchen um eine weitere Performancesteigerung zu erreichen und so das Limit für 5 Objekte zu erhöhen.

Literaturverzeichnis

[Ba96] Bair, George L.: Airbone radar simulation. Camber Corporation 1996

[BB06] Bühren, Markus; Bin, Yang: Simulation of Automotive Radar Target Lists using a Novel Approach of Object Representation. Intelligent Vehicles Symposium 2006. S. 314-319

[BB07] Bühren, Markus; Bin, Yang: Extension of Automotive Radar Target List Simulation to consider further Physical Aspects. Telecommunications 2007; S. 1-6

[Be13] Belbachir, Assia et. al.: Simulation-Driven Validation of Advanced Driving-Assistance Systems. Procedia-Social and Behavioral Sciences, 2012, 48. Jg., S. 1205-1214.

[Ka86] Kajiya, James T.: The rendering equation. SIGGRAPH 1986.

[NSM11] Nentwig, Mirko; Schieber, Reinhard; Miegler, Maximilian: Hardware-in-the-Loop-Test für vernetzte Fahrerassistenzsysteme. ATZ elektronik, 04/2011, 6. Jahrgang, S. 20-25

[Pa10] Parker, Steven et. al.: Optix: a general purpose ray tracing engine. ACM Trans. Graph. 2010.

[PLK08] Peinecke, N.; Lueken, T.; Korn, B. R.: Lidar simulation using graphics hardware acceleration. Digital Avionics Systems Conference 2008

[Sh12] Shuiying, Wang et. al.:. Shader-based sensor simulation for autonomous car testing. Intelligent Transportation System 2012. S. 224-229

[Su11] Sume, A et. al.: Radar Detection of Moving Targets Behind Corners, Transactions on Geoscience and Remote 2011.

[WWS10] Wohlfahrt, Christoph; Weizenegger, Florian; Smuda, Peer: HiL-Testtechnologie für kamerabasierte Fahrerassistenzsysteme. AutoTest 2010