TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik

Institut für Nachrichtentechnik

Oberseminar Kommunikationsnetze

zum Thema

Evaluation von Algorithmen für Compressed Sensensing durch KL1P-Bibliothek

vorgelegt von Xiang, Zuo
Matrikel-Nr 4117235
im Studiengang Informationstechnik, Jg. 2014

Betreuer: Dipl.-Ing. Carsten Herrmann
Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. Dr. h.c. Frank H.P. Fitzek
Tag der Einreichung: 14. April 2016

1 Einleitung

Ein wichtiges Ziel der Signalverarbeitung im technischen Gebiet ist die Rekonstruktion eines Signals durch eine Reihe von gemessenen Stichproben. Im Allgemeinen ist diese Aufgabe unmöglich. Aber mit vorherigen Kenntnissen sowie Annahmen des Signals, kann es durch beschränkte Messungen perfekt rekonstruiert werden.

Mit der Entwicklung der Kommunikationstechnik, vergrößert sich die Menge der zu verarbeitenden Daten ziemlich schnell. Deshalb ist die Komprimierung ziemlich von Bedeutung, um Ressource der Speicherung und Übertragung auszunutzen.

Compressed Sensing(kurz als CS) ist eine Technik zu effizienten Erwerb und Rekonstruktion des Signals, indem Lösungen unbestimmten linearen Systemen zu finden. Das basiert auf dem Prinzip, dass die Seltenheit des Signals durch gewisse Optimierung ausgenutzt werden kann, um es zu rekonstruieren mit deutlich weniger Proben als die von dem Abtasttheorem von Shannon-Nyquist erforderlichen. Dadurch können das originale Signal statt nach dem Erfassen sondern während der Abtastung schon wirksame komprimiert. Deshalb wird diese Technik für neue Kommunikationstechnik eine wichtige Rolle spielen, wie zum Beispiel das 5G Mobil Netzwerk.

Es gibt unterschiedliche Arten von Algorithmen, womit CS Probleme gelöst werden können. Im Rahmen dieser Arbeit werden drei praktisch repräsentative Variante OMP, CoSaMP, AMP sowie auch ein wichtiger Maßstab BP durch Verwendung einer portablen C++ Bibliothek KL1P durch Testparameter evaluiert, um die Funktionalität sowie Leistung dieser Bibliothek zu bewerten. Statistische Ergebnisse des Tests werden durch geeignete Grafiken veranschaulicht und analysiert. Diese Testdaten sowie Grafiken könnten für weitere Untersuchung der CS Algorithmen zur Verfügung stehen.

Dieses Protokoll wird in vier Teilen gegliedert. Im Kapitel 2 werden zuerst die genaue Aufgabenstellung, verwendete CS Algorithmen, das Evaluationsmodell sowie Werkzeuge vorgestellt. Danach werden im Kapitel 3 statistische Ergebnisse unterschiedlicher Testsituationen dargestellt und bewertet. Anschießend stehen im Kapitel 4 Zusammenfassungen sowie einige Ausblicke in Bezug auf Weiterentwicklungen dieser Arbeit und Verbesserungen der Evaluationsprogramm.

2 Methode

2.1 Aufgabenbeschreibung

Die Bewertung der CS Algorithmen kann entweder durch eine gegebene Simulationsumgebung wie Matlab, oder durch Implementierung auf einem realen Plattform wie Verwendung der KL1P C++ Bibliothek. In dieser Arbeit soll ein Programm mit der KL1P Bibliothek geschrieben werden, wodurch vier darin implementierte CS Algorithmen verwendet und evaluiert werden. Damit sollen statistische Daten bekommen und durch geeignete Arten von Grafiken veranschaulicht und analysiert werden. Außerdem wird diese Aufgabe auch vom anderen Studenten mit Matlab simuliert, um einen allgemeinen Vergleich dazwischen zu führen. Darüber hinaus wird diese Evaluation auch auf einem Einplatinencomputer mit dem Namen Odroid gemacht, um die Laufzeit dieser Bibliothek auf der ARM Architektur zu testen sowie mit der auf dem PC zu vergleichen. Ergebnisse sowie Daten können für weitere Arbeiten um den CS Bereich zur Verfügung stehen.

2.2 CS Algorithmen

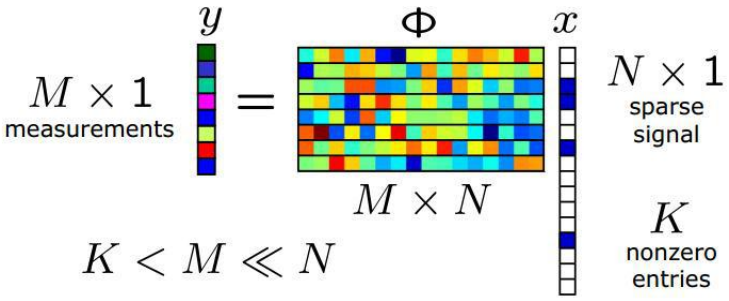


Abbildung 2.1: CS Messung [1]

Ein einfaches Beispiel von der CS Messung wird in der Abbildung 2.1 dargestellt. Das originale Signal x wird mathematisch als ein Vektor von N Elementen betrachtet. Durch Multiplizieren mit einer Abtastungsmatrix ϕ ($M \cdot N \, mit \, M \ll N$) wird das komprimierte Signal als Vektor y von M Elementen bekommen. Die Aufgabe der CS Algorithmen ist die Rekonstruktion von x, nämlich Lösung eines unterbestimmten(weil $M \ll N$) Gleichungssystems:

$$min\|x\|_0 \quad subject\ to \quad \phi \cdot x = y$$

wobei $\|x\|_0$ ist die Anzahl von Nicht-Null Elementen im Vektor x

Um das zu ermöglichen sollte hier angenommen werden, dass x spärlich ist. Deshalb können CS Probleme auch als Suchungen nach spärlicher Approximation betrachtet werden.

Es liegt mindestens fünf Kategorien von Algorithmen für Suchung nach spärlicher Approximation vor [2]. Diese Arbeit konzentriert sich auf vier typischen Algorithmen. Weil BP(Basis Pursuit) ein wichtiger Maßstab der Leistung eines CS Algorithmus ist, wird diese Methode von Minimierung des L1-Norms hier getestet. Außerdem sind schellere gierige Methoden der Anpassungsverfolgung(Matching Pursuit kurz als MP) von Interesse, wobei die Rekonstruktion von x durch iterative Verbesserung der gegenwärtigen Schätzung realisiert wird. Zwei Varianten davon werden bewertet, nämlich OMP(Orthogonal MP) und CoSaMP(Compressive Sampling MP). Diese gierige Algorithmen sind relativ schneller aber hat damit auch schlechtere Leistung wie BP. Deshalb wird endlich der AMP(Approximate Message Passing) Algorithmus verwendet, der eine Kombination von guter Lesitung sowie auch kurzer Laufzeit hat. [3]

BP (auch als L1-Minimierung) ist eine Methode von konvexer Optimierung. Nämlich die Lösung nach $min\{\|x\|_1\}\, mit\ y = \phi \cdot x$ und $\|x\|_1 = \sum_{i=1}^{n}|x_i|$. Dieser Algorithmus hat eine relative gute Leistung aber konvergiert sich langsame, weshalb er nicht geeignet für Anwendungen ist.

OMP ist die einfachste effektive Methode vom MP. Dabei wird iterative die stärkste korrelierte Spalte in der Abtastungsmatrix gewählt und damit den rekonstruierten Signalvektor schrittweise berechnet. Bei jeder Iteration wird ein Residuum berechnet und das Verfahren endet, falls dieses Residuum eine Schränke der Toleranz erreicht. Die genaue mathematische Beschreibung von OMP kann man in der Literatur [2] finden. Dabei ist der Schritt jeder Iteration für die Identifikation, nämlich Suchung nach stärksten korrelierten Spalten am aufwendigsten.

CoSaMP ist eine verbesserte MP-Methode, um die praktische Leistung zu steigern. Dabei sind einige neue Strategien zu verwenden, wie z.B. bei jeder Iteration mehrere Spalten auszuwählen(Tuningparameter) und danach die

Reihe von aktiven Spalten jeder Schätzung gezielt zu reduzieren. Das grundsätzliche Verfahren davon wird auch in der Literatur [2] dargestellt. Wegen dieser Verbesserungen ist CoSaMP in der Praxis normalerweise schneller und auch effektiver als OMP.

AMP ist ein iterativer Algorithmus, der gleichzeitig bessere Leistung als oben genannte MP-Methode sowie günstige Geschwindigkeit besitzt. Die ausführliche Beschreibung sowie Formeln dieses Algorithmus steht in der Literatur [3] zur Verfügung.

2.3 Evaluationsmethode

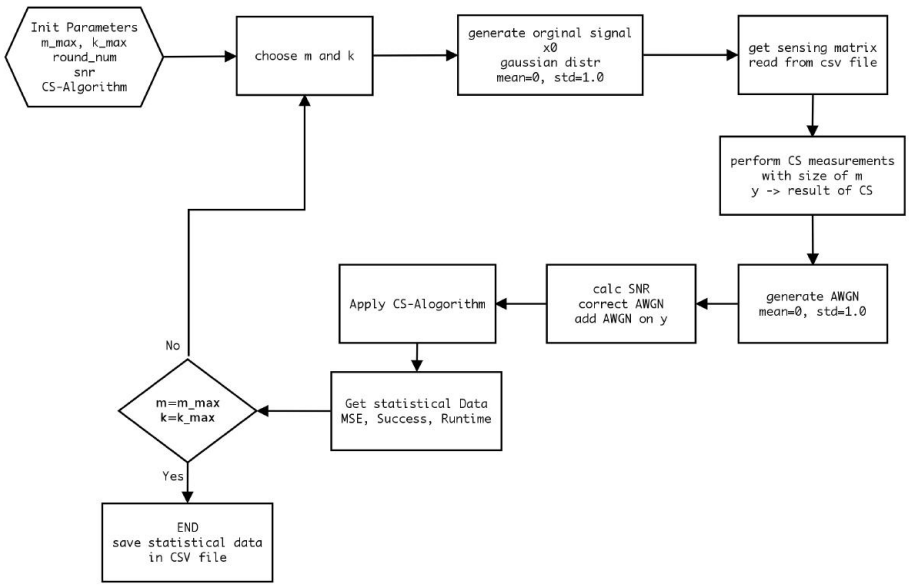


Abbildung 2.2: Evaluationsmodell

In der Abbildung 2.2 wird das Modell für die Evaluation dargestellt, wobei ein zufällig generiertes Signal x_0 mit Länge $n = 250$ durch Abtastung komprimiert(das erfasste Signal y addiert sich mit AWGN-Rauschen) wird und danach durch CS Algorithmen wieder rekonstruiert wird. Zuerst werden Parameter initialisiert. Es gibt hier für jede Testsituation vier Parameter, nämlich das maximale Sparcity des originalen Signals k_{max}, die Anzahl von

maximalen Abtastungen m_{max}, die Anzahl von Testrunden für jede Schleife sowie das Signal-Rauch-Verhältnis(SNR). Nach jeder Runde werden statistische Daten für die Leistung berechnet und gespeichert. Das Programm endet bis m und k den Maximum erreichen.

Die Anzahlung der Messungen(m) soll relativ kleiner als n, sonst spielt die Komprimierung keine Rolle. Gleichzeitig soll das originale spärlich sein, nämlich soll k einen relativ kleinen Wert betragen. Außerdem benötigt das Programm gemäß einfachen Tests eine relativ lange Laufzeit. Deshalb werden Testparameter auf zwei Plattformen wie folgendes beschränkt.

Auf dem PC : m: 1-125, k: 1-100, Testrunden: 100

Auf dem Odroid : m: 1-125, k: 1-60, Testrunden: 50

Auf beide: SNR : 0db, 10db sowie 20db

Für die Erzeugung des originalen Vektor($n \cdot 1$) wird eine Normalverteilung mit dem Mittelwert(μ) Null sowie Standardabweichung(σ) gleich Eins jeweils verwendet. Für jede Runde ist eine gleiche Abtastungsmatrix$\phi(m \cdot n)$ mit der Normalverteilung($\mu = 0, \sigma = 1.0$) zu benutzen. Diese Matrix wird jeweils aus einer CSV-Datei gelesen. Außerdem wird für AMP zusätzlich eine pseudo-Normalisierung von ϕ benötigt, nämlich für jede Element ϕ_i in ϕ, $\phi_i = \frac{1}{\sqrt{m}}$. Um einen Vergleich zwischen unterschiedlichen Algorithmen zu führen, wird diese nicht notwendige Normalisierung für andere drei Varianten auch gemacht.

Als Kriterien für die Leistung des Algorithmus werden drei Kenngrößen betrachtet, nämlich MSE(Mean Squared Error), die Erfolgsrate sowie die Laufzeit.

MSE kann als einen Maßstab für den absoluten Unterschied zwischen Vektoren verwendet werden. Für den originalen Signalvektor x_0 sowie den rekonstruierten Vektor x wird das MSE durch folgende Formel berechnet: $MSE = \frac{1}{n}\sum_{i=1}^{n}(x_0(i) - x(i))^2$

Erfolgsrate In vielen Situation ist der absolute Unterschied des Betrags der Rekonstruktion nicht bedeutend, sondern man interessiert sich dafür, ob alle Positionen der Nichtnull Elementen des originalen Signal getestet werden können. Damit spricht man von einer Erfolgsrate. Es gibt unterschiedliche Kriterien für den Erfolg einer CS Rekonstruktion. In dieser Arbeit wird eine ϵ Umgebung für die Betragsdifferenz jedes Vektorelements verwendet. für

bestimmt i als Index des Vektors: falls $|x_0(i) - x(i)| \leq \epsilon$ dann $s_i = 1$ für erfolgt, sonst $s_i = 0$ für durchgefallen. Danach wird diese Rate so berechnet: $Erfolgsrate = \frac{1}{n} \sum_{i=1}^{n} s_i$. Hier $\epsilon = 5 \cdot 10^{-8}$

Laufzeit Diese Laufzeit bedeutet die durchschnittliche Dauer jeder Rekonstruktion, nämlich für jede Runde des Programms. Jede Laufzeit einer Runde wird zuerst im Einheit von Millisekunden gemessen und in einem Vektor gespeichert. Danach werden Mittelwert sowie Varianz des Vektor berechnet.

2.4 Vorstellung der Werkzeuge

2.4.1 KL1P

KL1p ist eine portable C++ Bibliothek für die spärliche Rekonstruktion inverser Probleme von unbestimmten linearen Systemen, wie z.B im CS Gebiet. Sie ist plattformunabhängig und kann auf eine große Anzahl von Systemen sowie Plattformen mit einem C ++ konformen Compiler verwendet werden. Der Schwerpunkt davon liegt auf die Verwendbarkeit sowie Erweiterbarkeit. Einiger der häufigsten Algorithmen sind implementiert in dieser Bibliothek und neue Methoden können auch einfach darin addiert werden. Diese Bibliothek ist noch in der Entwicklung und damit nicht geeignet für die Produktion.[4]

Derzeit implementierte compressed sensing Algorithmen:

- Basis Pursuit (BP)
- Orthogonal Matching Pursuit (OMP)
- Regularized Orthogonal Matching Pursuit (ROMP)
- Compressive Sampling Matching Pursuit (CoSaMP)
- Subspace Pursuit
- Smoothed L0 (SL0)
- Approximate Message Passing (AMP)
- Expectation Maximization Belief Propagation (EMBP)

Mehr Informationen sind auf der Homepage des Projekts verfügbar(`http://kl1p.sourceforge.net/home.html`).

2.4.2 Testplattformen

PC Dieses Evaluationsprogramm wird zuerst auf einem PC mit X86_64-Architektur zum Lauf gebracht. Dieses PC mit dem Betriebssystem Ubuntu 14.04.03(LTS) hat einen Prozessor von Intel-Core-i7-6700(4-Kern 3.4GHz) sowie 16GB RAM.

Odroid Odroid ist eine Einplatinencomputerreihe vom Hardwarehersteller Hardkernel für die Entwicklung von Hard- und Software. Aktuelle Odroid-Typen haben einen Mehrkernprozessor mit ARM-Architektur und können Linux und Android als Betriebssystem verwenden. In dieser Arbeit wird der Type Odroid-XU4 mit dem Betriebssystem Ubuntu 14.04.03(LTS) benutzt. Der hat einen 8-Kern-Prozessor von Samsung Exynos 5422, der einen ARM-Cortex-A15(4-Kern 2GHz) sowie einen ARM-Cortext-A7(4-Kern 1.3GHz) Prozessor enthält. Der RAM davon beträgt 2GB.

Mehr Informationen vom Odroid sind auf der Homepage von Hardkernel verfügbar(`http://www.hardkernel.com/main/main.php`).

2.4.3 Matplotlib

Für die Erstellung der Grafiken wird die Python-Bibliothek: Matplotlib verwendet. Damit können schöne graphische Darstellungen von Daten schnell erstellt werden. Eine Vielzahl von Beispielen steht auf der Galerie zur Verfügung. In dieser Arbeit werden z.B. farbige Kontur sowie 3D-Fläche Grafiken verwendet.

Mehr Informationen kann man auf der Homepage des Projekts finden(`http://matplotlib.org/index.html`).

3 Ergebnisse

Durch das im Kapitel 2.3 dargestellte Evaluationsmodell, wird ein C++ Programm mit der KL1P Bibliothek geschrieben und damit folgende statische Ergebnisse bekommen. Weil für bestimme Testsituation, variiert sich die Kenngröße der Leistung gleichzeitig mit m und k, nämlich zweidimensional, werden Werte in einer Matrix mit dem Format CSV(Comma-Separated Values) gespeichert. Es liegt viele Daten unterschiedliche Testparameter.

Wegen des beschränkten Umfangs dieser Arbeit werden einige repräsentative Ergebnisse daraus durch Grafiken veranschaulicht und damit analysiert.

Alle Codes sowie CSV Daten von Matrizen sind auf das Github-Repository verfügbar(`https://github.com/stevelorenz/kl1p-test`). Ergebnisse der Leistungsgröße sind im Ordner test_results. Die Abtastungsmatrix liegt im Ordner rsc. Und Grafiken sind im Ordner figs zu finden.

3.1 Ressourcenverbrauch

Beim Lauf des Programms wird der Verbrauch des CPUs durch das Werkzeug *htop* überwacht. Dabei ist es zu bemerken, dass das Programm mit dieser Bibliothek recht rechenaufwendig ist. Es belegt beinahe hundert Prozent der Ressourcen des CPUs bei der ganzen Laufzeiten. Außerdem wird jeweils nur ein Kern des Prozessors stark belastet. Daraus kann man erfahren, dass diese Bibliothek noch nicht für die Mehr-Kern-Architektur parallel optimiert wird.

3.2 MSE

In der Abbildung 3.1 wird das MSE vier Algorithmen auf dem PC mit dem $SNR = 20db$ dargestellt. Während sich m von 1 bis 125 variiert, nimmt k drei typische Werte 10, 30 sowie 60, wodurch Situationen von gut, mittlere sowie schlecht spärlichem Signal repräsentiert werden.

Aus der Grafik wird es gezeigt, dass zuerst all Algorithmen eine gleiche abnehmende Tendenz aufweist, was den theoretischen Ergebnissen entspricht. Aber sind Kurven nicht alle monoton, nämlich Linien von OMP, AMP sowie BP beim kleinen m zuerst sich vergrößern. Außerdem werden in diesen Situationen Ausnahmen mit dem Namen “klab::KZeroNormLeastSquareException ”vom Programm geworfen. Um das Programm nicht zu unterbrechen, wird hier diese Ausnahme gefangen und statistische Berechnung auch gemacht. In dieser Situation wird gemäß Tests ein Vektor mit allen Null-Elementen als Ergebnis generiert.

Es liegt hier eine Vermutung für diese Ausnahmen vor. Wie in der Literatur [2] beschreibt, dass der Schritt der Schätzung von Koeffizienten in jeder Iteration eine Lösung des Least-Square-Problems benötigt, wobei ein Gleichungssystem gelöst werden soll. Wie folgendes Beispiel mit $m = 2$ und zweit ausgewählten Spalten:

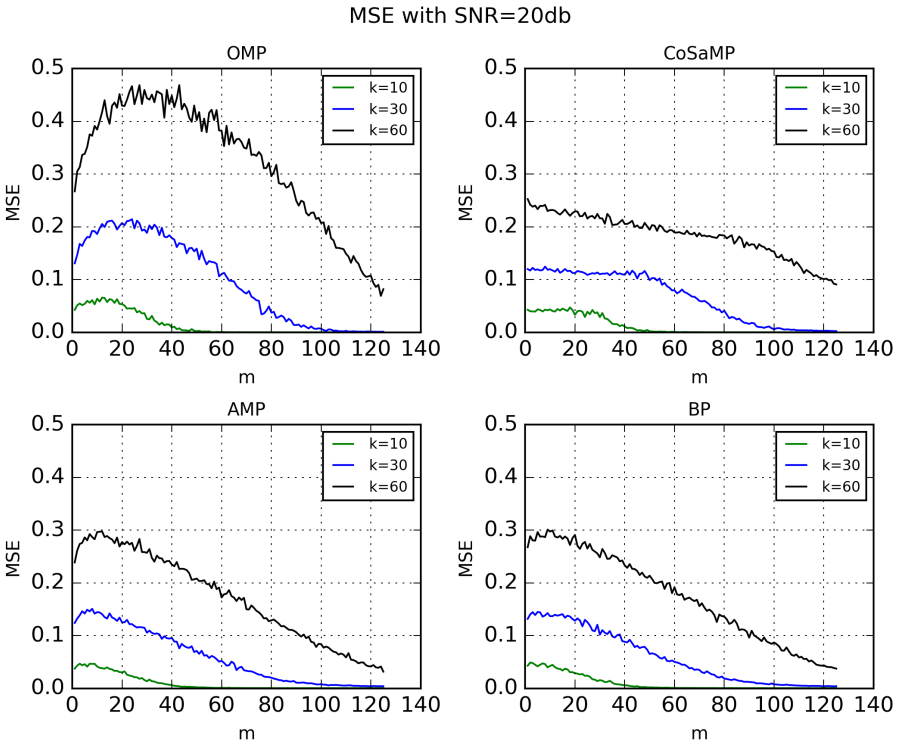


Abbildung 3.1: MSE with SNR=20db

$$y_i = A \cdot x_i \quad mit \quad A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Wenn z.B. zwei Spalten der Matrix A nicht linear unabhängig voneinander sind, kann diese Gleichung nicht gelöst werden. Dann wird bei KL1P-Bibliothek so eine "klab::KZeroNormLeastSquareException "geworfen und ein rekonstruierter Vektor mit nur Nullelementen erzeugt. Damit nimmt das MSE die Energie des originalen Signal und einen relativ größeren Wert beträgt. Die Wahrscheinlichkeit dieser linearen Abhängigkeit verkleinert sich mit zunehmenden m relativ schnell, weshalb diese Ausnahmen nur beim kleinen m auftreten. Wegen der beschränkter Arbeitszeit wird diese Vermutung noch nicht durch Analyse des Kodes or Testprogramm geprüft.

Außerdem kann man daraus erfahren, während CoSaMP beim kleinen m durchschnittlich kleinstes MSE hat, ist dieser Wert vom OMP relativ größer. AMP und BP haben fast gleiche Kurven und liegen in der Mitte. Darüber hinaus haben Schnittpunkten der Kurve beim kleinen k(10,30) vier Algorithmen mit der x Achse(m) ungefähr gleiche Werte, z.B $m = 50$ mit $k = 10$. Das heißt mit einer genügenden Anzahl von Abtastungen, hat das originale sowie rekonstruierte Signal durch alle vier Algorithmen fast keinen Unterschied. Falls das Signal nicht so spärlich ist(mit groß k), weist CoSaMP eine relativ kleinere Steigung als andere drei Algorithmen auf. Insgesamt besitzen AMP sowie BP eine durchschnittlich bessere Leistung von dem MSE für unterschiedliche k.

In der Abbildung 3.2 wird MSE auf dem PC mit SNR=10db veranschaulicht, nämlich mit stärkerem Rauschen . Alle Kurven haben ähnliche Tendenz und Wertbereich wie die von SNR=20db, wodurch eine gute Fähigkeit vier Algorithmen gegen das AWGN-Rauschen aufgewiesen wird. Während OMP einen relativ größeren Wachstum hat, verändern sich drei andere Algorithmen nicht deutlich. Außerdem ist es zu zeigen, dass Schnittpunkten mit x-Achse nach rechts verschoben werden. Das heißt, dass all Algorithmen mehr Abtastungen brauchen, um die Abweichung der Rekonstruktion zu reduzieren.

Aus diesen zwei Grafiken ist es zu zeigen, dass vier implementierten Algorithmen in der Bibliothek richtig funktioniert und Ergebnisse mit den theoretischen Analysen übereinstimmen.

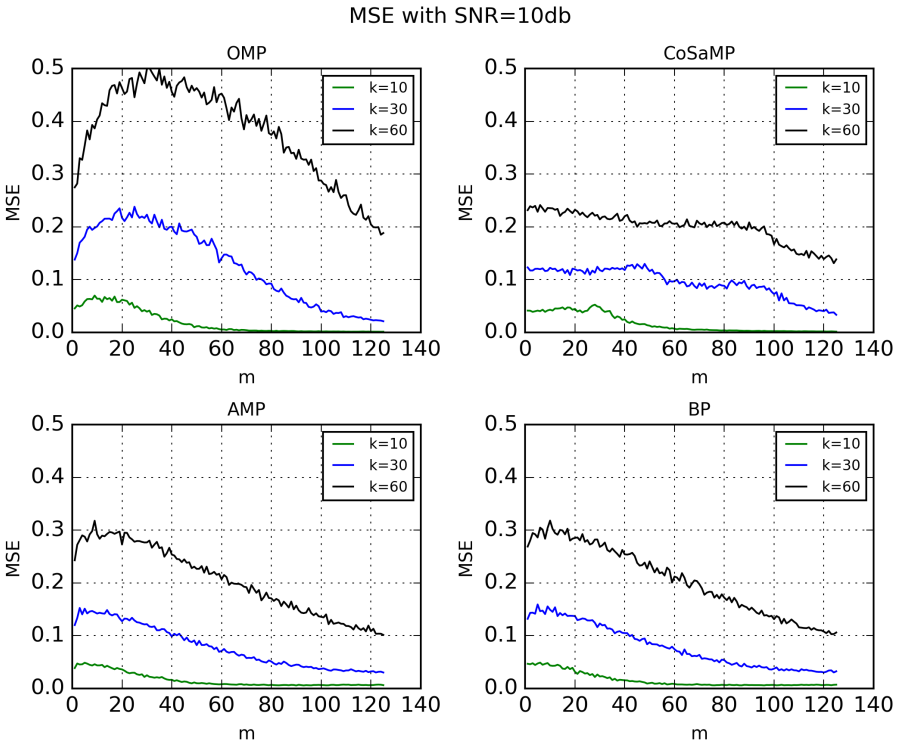


Abbildung 3.2: MSE with SNR=10db

3.3 Erfolgsrate

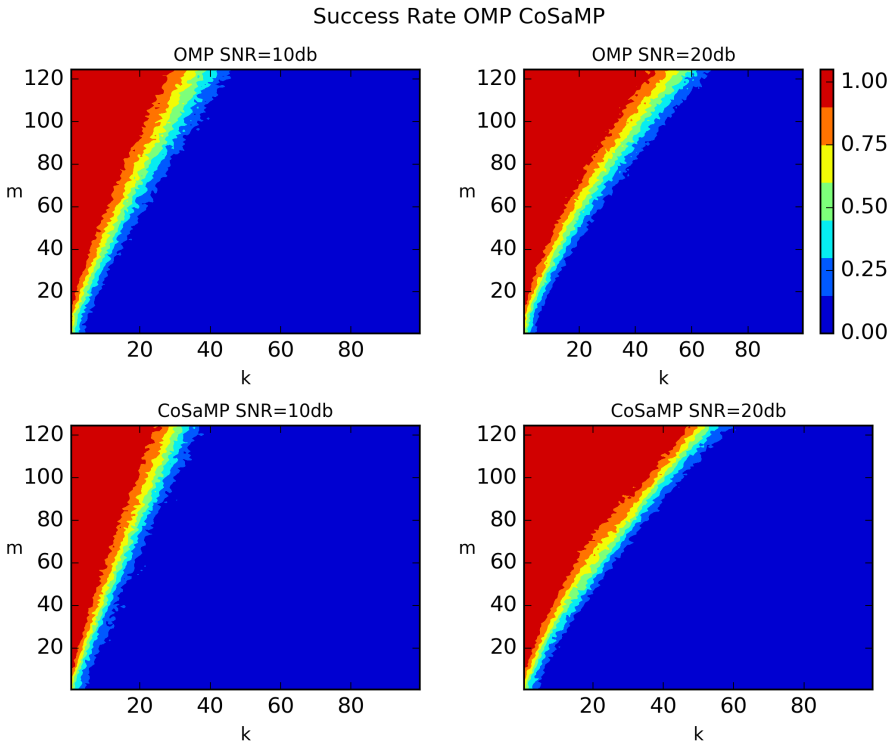


Abbildung 3.3: Success Ratio of OMP and CoSaMP

Die Abbildung 3.3 sowie 3.4 stellen die Erfolgsraten auf dem PC vier Algorithmen dar. In der Grafiken wird diese Rate, nämlich ein Wert zwischen Null und Eins, durch unterschiedliche Farben bezeichnet. Nämlich rot sowie dunkel blau jeweils die größte sowie kleinste Rate kennzeichnet. In allen Grafiken liegt ein Streifen mit farbiger Abstufung vor, was als die Grenze zwischen Erfolgen und Ausfällen betrachtet werden kann. Das Kriterium dafür ist im Kapitel 2.3 zu beschreiben. Aus den Grafiken kann man zuerst erfahren, dass AMP sowie BP vergleichsweise deutliche größere Erfolgsbereiche in beide Situationen besitzen. Außerdem wird es gezeigt, während OMP sowie CoSaMP sehr ähnliche Leistungsfähigkeiten aufweisen, verhält sich das AMP beinahe wie BP . Wie im vorigen Kapitel erwähnt, dass BP ein wichtiger Maßstab für die Erfolgsrate ist. Deshalb weist AMP bei diesem Kriterium die beste Leistung unter getesteten Algorithmen auf.

Das Ergebnis von Erfolgsrate trifft auch den theoretischen Kenntnissen zu.

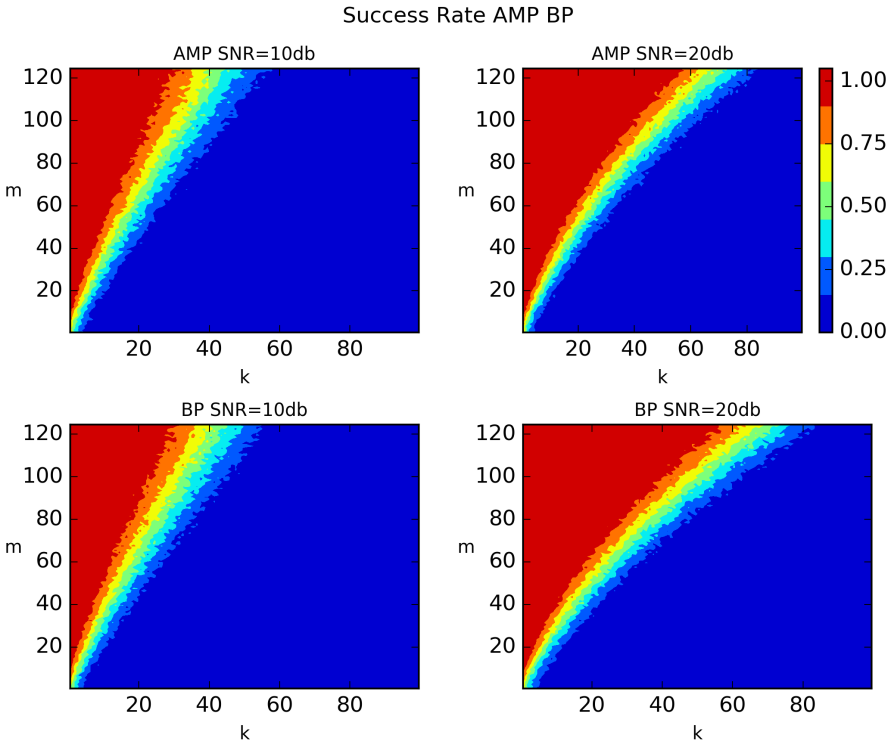
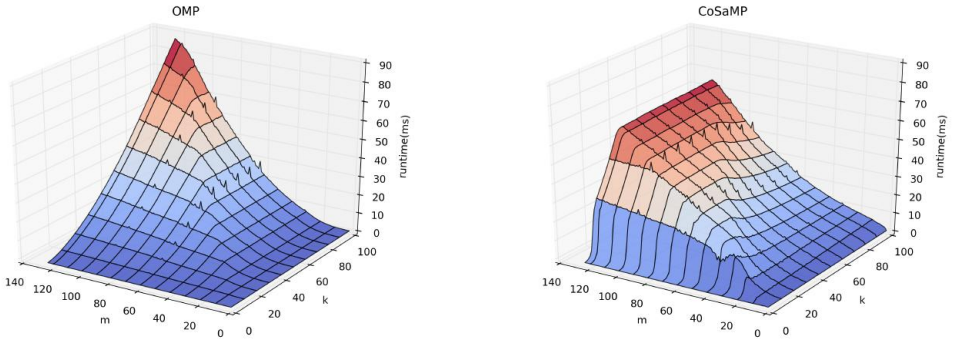


Abbildung 3.4: Success Ratio of AMP and BP

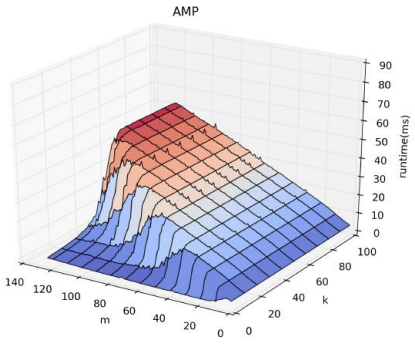
3.4 Laufzeit

Weil BP deutliche rechenaufwendig ist, wird es normalerweise nicht als eine praktische Applikation implementiert. Deshalb wird die Laufzeit jeder Rekonstruktion anderer drei Algorithmen auf dem PC sowie Odroid gemessen. Hier wird das abgetastete Signal nicht verrauscht, nämlich Tests ohne AWGN-Rauschen. Ergebnissen davon werden durch 3D-Fläche Grafiken veranschaulicht.



(a) OMP

(b) CoSaMP

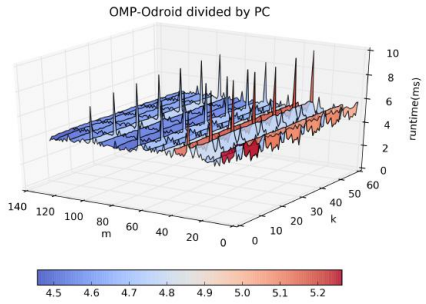


(c) AMP

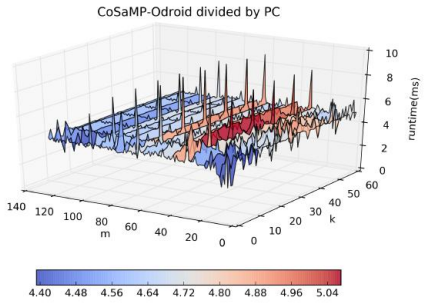
Abbildung 3.5: Laufzeit auf dem PC in Millisekunden

Im Abbildung 3.5 werden absolute Laufzeiten auf dem PC dargestellt. Für dem Maximum beim größten m und k, ist die Laufzeit vom OMP am längsten mit ungefähr 85 Millisekunden. Im Vergleich dazu beträgt dieser Wert von CoSaMP sowie AMP jeweils ungefähr 62 und 51 Millisekunden, weshalb AMP am schnellsten im schlechtesten Fällen ist. Außerdem kann

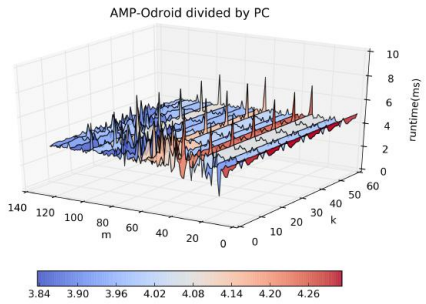
man erfahren, dass die zunehmende Tendenz unterschiedlicher Algorithmen verschiedene Beziehungen von m und k aufweist. Beim OMP ist die Laufzeit proportional zur beiden m und k, weil OMP mehr Iterationen und auch mehr Rechenaufwand jeder Iteration benötigt. Im Vergleich dazu, ist die Laufzeit vom CoSaMP und AMP empfindlich gegen die Veränderung vom m. Das heißt, dass sich die Laufzeit nicht so stark von das Sparcity des Signals abhängt. Beim CoSaMP reduziert sich die Laufzeit beim größeren k mit bestimmten Bereich von m. Die Laufzeit vom AMP bleibt beinahe unverändert im großen Bereich von k für bestimmt m. Zusammenfassend für den ganzen Testbereich besitzt AMP die beste Leistung für die Zeitaufwand, nämlich relativ günstig für gut sowie schlecht spärliche Signale. Die Ergebnisse stimmt in den theoretischen Kenntnissen überein.



(a) OMP



(b) CoSaMP



(c) AMP

Abbildung 3.6: Laufzeit in Millisekunden

Ergebnisse eines Vergleichs für die Zeitaufwand gleicher Situationen zwi-

schen auf dem X86_64 sowie ARM Architektur werden in der Abbildung 3.6 dargestellt. Daraus kann man erfahren, dass das Programm auf dem Odroid durchschnittlich ungefähr 4.6-fache Zeit benötigt als es auf dem PC. Für einzelnen Algorithmus nimmt OMP sowie CoSaMP einen Faktor von 4.9 sowie 4.75. Im Vergleich dazu beträgt dieser Faktor beim AMP nur 4. Das heißt, dass sich das AMP auf dem Odroid am besten verhält.

Außerdem kann man erfahren, dass es in den Flächen viele Extremwerten vorliegen. Der Grund dafür wird hier wegen der beschränkten Arbeitszeit nicht gesucht. Nach einer Vermutung ist das vielleicht auf die Schutzstrategien des Hardware auf dem Odroid zurückzuführen. Nämlich nach gewissen aufwendigen Rechnungen wird die Leistung des CPUs kontrolliert reduziert, um z.B. das Gerät zu kühlen. Darüber hinaus wie oben erwähnt, ist diese Bibliothek nicht für Mehr-Kern-Prozessor entwickelt. Deshalb wird das Potenzial des Octa-Kern-CPUs auf dem Odroid nicht ausgenutzt, weshalb die Laufzeit deutlich zunimmt.

Die genaue Werte der Laufzeit auf dem Odroid können im Repository gefunden. Sie können z.B. als Maßstab dafür, ob CS Algorithmen ohne parallele Optimierung darauf implementiert werden, weil die Verbesserung des Algorithmus für Mehr-Kern-Prozess sehr aufwendig sein könnte

4 Zusammenfassung und Aussicht

4.1 Zusammenfassung

Die Hauptaufgabe dieser Arbeit ist eine Evaluation vier verschiedenen CS Algorithmen durch das im Kapitel 2.3 beschriebene Modell und die KL1P Bibliothek. Mit dem Testprogramm werden statistische Kenngröße für die Leistung des Algorithmus bekommen. Danach sind diese Ergebnisse durch geeignet Grafiken zu veranschaulichen und damit zu analysieren. Ein allgemeiner Vergleich mit im Kapitel 2.3 beschriebenen Kriterien zwischen CS Algorithmen sowie Plattformen wird geführt, um die Funktionalität sowie Cross-Plattform Fähigkeit dieser Bibliothek zu testen.

Zusammenfassend funktionieren sich diese vier Algorithmen in der KL1P Bibliothek erfolgreich. Testergebnisse stimmen durchschnittlich mit den theoretischen Kenntnissen überein.

Allerdings ist diese Bibliothek nicht für den Mehr-Kern-Prozessor optimiert

und deshalb das Potential der parallelen Rechnung nicht ausnutzt. Wenn diese Bibliothek auf einem Plattform mit Mehr-Kern-Prozessor oder der Fähigkeit für verteilte Rechnung eingesetzt wird, ist die parallele Optimierung ziemlich wichtig für die Verbesserung der gesamten Leistung.

Das Testprogramm sowie alle Daten(Matrix im Format CSV, Grafiken sowie Dokumentationen) stehen auf dem Github-Repository zur Verfügung(`https://github.com/stevelorenz/kl1p-test`).

4.2 Aussichten

Wegen der beschränkten Arbeitszeit wird hier nur das Grundprinzip und Verfahren vier Algorithmen gelernt, wodurch einige Ergebnisse nicht so ausführlich theoretisch analysiert und erklärt werden können. Deshalb ist es sinnvoll, für weitere Arbeit mehr Kenntnisse der CS Algorithmen zu bekommen.

Außerdem werden in dieser Arbeit einige nicht geprüfte Vermutungen gemacht. Zum Beispiel die Gründe für den Wurf der Ausnahmen beim kleinen m sowie Extremwert der Laufzeiten auf dem Odroid. Prüfungen dieser Vermutungen sollen als sinnvolle weitere Arbeit betrachtet werden.

Literaturverzeichnis

[1] Stephen Hsu. *The human genome as a compressed sensor*. 2013. URL: `http://infoproc.blogspot.de/2013/10/the-human-genome-as-compressed-sensor.html` (siehe S. 2).

[2] Joel A Tropp und Stephen J Wright. „Computational methods for sparse solution of linear inverse problems“. In: *Proceedings of the IEEE* 98.6 (2010), S. 948–958 (siehe S. 3, 4, 8).

[3] David L Donoho, Arian Maleki und Andrea Montanari. „Message-passing algorithms for compressed sensing“. In: *Proceedings of the National Academy of Sciences* 106.45 (2009), S. 18914–18919 (siehe S. 3, 4).

[4] *Overview of KL1P lib*. 2012. URL: `http://kl1p.sourceforge.net/home.html` (siehe S. 6).