Transiente Lichtemission und Reaktionen an der MgO/Si Grenzfläche

Von der Naturwissenschaftlichen Fakultät der
Gottfried Wilhelm Leibniz Universität Hannover

zur Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation
von
Tim Gebensleben, M. Sc.

2021

Referent: Professor Dr. rer. nat. Jörg August Becker

Korreferent: Professor Dr. rer. nat. Jens-Uwe Grabow

Tag der Promotion: 15.12.2020

Kurzzusammenfassung

Die vorliegende Arbeit befasst sich mit der Untersuchung von Siliciumproben auf Magnesiumoxid-Substraten unter Hochvakuum und bei hohen Temperaturen. Für kleine Proben (bis 100 µg) wurde ab einer Temperatur von etwa 1400 Kelvin ein charakteristischer, transienter Lichtemissionseffekt beobachtet. Die Probe verliert über einen Zeitraum von wenigen Sekunden an Helligkeit und leuchtet dann innerhalb von Sekundenbruchteilen wieder auf. Der Effekt konnte während der Experimente mit Hilfe eines Videomikroskops beobachtet und aufgezeichnet werden.

In anfänglichen Versuchen wurden Siliciumkörnchen als Probe verwendet. Um eine einheitliche Probengeometrie und ebene Auflagefläche auf dem Substrat zu gewährleisten, kamen später Silicium-Cantilever aus der Rasterkraftmikroskopie zum Einsatz. Für die Platzierung der hoch empfindlichen Cantilever auf den Substratplättchen musste eine Präparationsmethode konzipiert werden.

Zur Aufklärung des Effekts wurden Experimente mit verschiedenen Versuchsparametern entwickelt und durchgeführt, zum Beispiel Langzeit-Temperexperimente oder Vergleichsexperimente mit anderen Substratmaterialien. Die Reaktionsprodukte wurden mittels Elektronenmikroskopie und EDX-Spektroskopie untersucht. Für einige Experimente wurde eine dünne, wahrscheinlich silikathaltige Schicht von Reaktionsprodukten an der Probe-Substrat-Grenzfläche gefunden.

Der Emissionseffekt ist wahrscheinlich thermischer Natur und hängt mit dem Aufbau und Zerfall dieser Silikatschicht und der endothermen Bildung von gasförmigem Siliciummonoxid zusammen. Um die Temperatursprünge beim Aufleuchten der Probe zu quantifizieren, wurde ein Computerprogramm entwickelt. Die Software ermöglicht die zeitliche Auswertung der Probenhelligkeit und deren Umrechnung in Absoluttemperaturen auf Basis von Videomikroskopie-Daten.

Es wurde ein einfaches Reaktionsmodell zum Auf- und Abbau einer Reaktionsschicht entwickelt. Auf Grundlage der mit der Software gewonnenen Probentemperaturen und Frequenzen des Emissionseffekts wurden die Aktivierungsenergien für die wesentlichen Schritte des Modells (Schichtaufbau und Lochbildung) abgeschätzt.

Schlagworte: Lichtemission, Pidgeon-Prozess, Grenzflächen

Abstract

This work discusses the investigation of silicon samples on magnesium oxide substrates under high vacuum and at high temperatures. For small samples (up to 100 µg) a characteristic, transient light emission effect was observed at temperatures of 1400 Kelvin and above. The sample loses brightness over a period of a few seconds, then lights up again within fractions of a second. The effect was observed and recorded with a video microscope.

In initial experiments, small silicon splinters were used as samples. Later experiments were conducted with silicon cantilevers as used in atomic force microscopy in order to ensure a consistent sample geometry and smooth area of contact with the substrate. Placement of the highly fragile cantilevers on the substrate plates demanded for the development of a sample preparation method.

The effect was investigated with a series of varied experiments, such as long term tempering or using different substrate materials. The reaction products were examined with electron microscopy and EDX spectroscopy. For some experiments a thin layer of reaction products, likely containing silicates, was found at the interface of sample and substrate.

The emission effect is in all likelihood thermal and related to the formation and decay of a silicate layer and the endothermic formation of gaseous silicon monoxide. In order to quantify the temperature differences during the darkening and lightening of the sample brightness, a computer program was developed. The software allows for the temporal analysis of the sample brightness and the conversion of brightness values into absolute temperatures on the basis of the video microscopy recordings.

A simple reaction model for the formation and decay of a reaction layer was developed. For the two main steps (layer formation and hole formation) the activation energies were estimated on the basis of the absolute sample temperatures and frequencies of the emission effect.

Keywords: light emission, Pidgeon process, interfaces

Inhaltsverzeichnis

Abkürzungsverzeichnis

AES	Auger-Elektronen-Spektrometer
AFM	Rasterkraftmikroskop (atomic force microscope)
BGR	Blau Grün Rot
EDX	energiedispersive Röntgenspektroskopie (energy dispersive X-ray spectroscopy)
FIB	fokussierter Ionenstrahl (focused ion beam)
GUI	grafische Benutzeroberfläche (graphical user interface)
HTAES	Hochtemperatur-Auger-Elektronen-Spektrometer
PBN	pyrolytisches Bornitrid
QMS	Quadrupol-Massenspektrometer
REM	Rasterelektronenmikroskop
ROI	Beobachtungszone (region of interest)
TEM	Transmissionselektronenmikroskop
USGS	United States Geological Survey

1 Einleitung

1.1 Magnesium und der Pidgeon-Prozess

Magnesium ist ein Erdalkalimetall der zweiten Hauptgruppe des Periodensystems mit der Ordnungszahl zwölf und der Elektronenkonfiguration [Ne]$3s^2$. Das Element kommt nach Binnewies u. a. in der Natur hauptsächlich in Form von Doppelsalzen wie Carnallit ($MgCl_2 \cdot KCl \cdot 6\,H_2O$) und Dolomit ($MgCO_3 \cdot CaCO_3$) vor [1]. Außerdem ist es Bestandteil der Silikate Enstatit ($MgSiO_3$) und Forsterit (Mg_2SiO_4). Magnesium ist zudem in großen Mengen im Meerwasser enthalten – hier bildet Mg^{2+} mit einer Konzentration von etwa $1{,}3\,mg\,l^{-1}$ das dritthäufigste Ion nach Na^+ und Cl^- [1, 2]. Der Massenanteil von Magnesium an der Erdhülle beträgt etwa zwei Prozent [3].

Das stark negative Reduktionspotential von Magnesium ($-2{,}36\,V$) lässt eine hohe Reaktivität erwarten, die Lagerung des Metalls vereinfacht sich allerdings durch die Tatsache, dass sich mit Luftsauerstoff rasch eine passivierende Schicht von Magnesiumoxid (MgO) auf der Metalloberfläche bildet [1]. Tabelle 1.1 fasst einige physikalisch-chemische Daten des Elements zusammen.[1]

Tabelle 1.1: Physikalisch-chemische Daten von Magnesium [1].

Atommasse	$24{,}305\,u$
Dichte	$1{,}74\,g\,cm^{-3}$
Schmelzpunkt	$650\,°C$
Siedepunkt	$1093\,°C$
Elektronegativität	1,3
Standardpotential	$-2{,}36\,V$
Ionisierungsenthalpie I	$0{,}74\,MJ\,mol^{-1}$
Ionisierungsenthalpie II	$1{,}46\,MJ\,mol^{-1}$

Hervorzuheben ist hier die geringe Dichte von $1{,}74\,g\,cm^{-3}$, inbesondere im Vergleich zu Aluminium mit einer Dichte von $2{,}70\,g\,cm^{-3}$ [4]. Magnesium ist das leichteste großindustriell produzierte Werkmetall und etwa die Hälfte der weltweiten

[1] Sofern nicht anders angegeben oder aus dem Kontext ersichtlich gelten im Folgenden temperaturabhängige Literaturwerte für 25 °C.

Produktion wird zu Magnesium-Aluminium-Legierungen weiterverarbeitet [1, 5]. Neben dem Einsatz als Werkmetall findet Magnesium zum Beispiel Anwendung beim Druckgießen, bei der Entschwefelung von Stahlschmelzen und bei der Titandarstellung im Kroll-Prozess [5]. Bei der Entschwefelung werden üblicherweise Magnesium, Calciumoxid (CaO) und Calciumcarbid (CaC_2) eingesetzt, welche mit dem Schwefel in der Metallschmelze zu schwer löslichen Sulfiden reagieren [6]. Das Entfernen des Schwefels verhindert die Entstehung von Eisensulfid-Einschlüssen (FeS) beim Abkühlen der Schmelze, welche die Brüchigkeit des Stahls erhöhen [7]. Im Kroll-Prozess wird Titantetrachlorid ($TiCl_4$) mit Magnesium gemäß

$$\mathrm{TiCl_4(g) + 2\,Mg\,(l) \longrightarrow Ti\,(l) + 2\,MgCl_2(l)} \tag{1.1}$$

bei etwa 1000 °C und unter Edelgasatmosphäre zu Titan reduziert [2]. Der hierbei gewonnene „Titanschwamm“ wird in weiteren Verfahrensschritten aufgereinigt.

Das Legieren von Magnesium ist oftmals nötig, um die mechanischen Eigenschaften des Werkstoffs zu verbessern. Reines Magnesium ist spröde und zerbricht leicht entlang von Scherbändern – schon geringe Beimengungen anderer Metalle (z. B. 1 Gew.% Al und 0.1 Gew.% Ca) können die Duktiliät drastisch erhöhen [8]. Abbildung 1.1 zeigt die Brüchigkeit reinen Magnesiums. Bereits geringfügige Deformationen durch Walzen führen zu makroskopischen Frakturen im Metall. Die Sprödheit wird durch den Mangel an unabhängigen Deformationsmoden im hexagonalen Kristallsystem erklärt [9, 10].

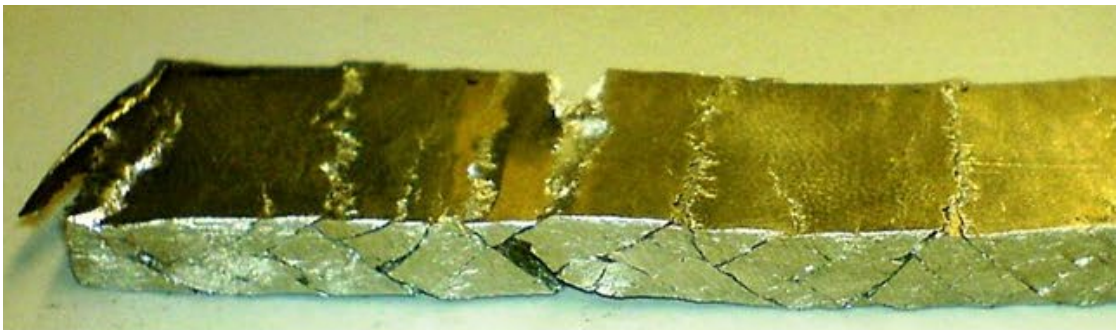


Abbildung 1.1: Polykristallines Magnesium, kalt gewalzt auf neunzig Prozent der ursprünglichen Dicke. Die Abbildung wurde in abgewandelter Form von Sandlöbes u. a. übernommen [8].

Neben Legierungsverfahren kann die Verformbarkeit von reinem, polykristallinem Magnesium auch durch das Erreichen kleiner Korngrößen (etwa 1 µm) erheblich gesteigert werden [11]. Weitere Herausforderungen beim Einsatz von Magnesium als Werkstoff sind die Tendenz zu korrodieren und die Entzündlichkeit des Metalls. Die für die Luftfahrt entwickelte neodym-, gadolinium- und zinkhaltige

Magnesiumlegierung „Elektron 21“ zeichnet sich durch besondere Korrosionsbeständigkeit aus [12]. Die Verbrennung von Magnesium mit Sauerstoff

$$2\,\mathrm{Mg(s)} + \mathrm{O_2(g)} \longrightarrow 2\,\mathrm{MgO(s)} \qquad \Delta_\mathrm{R}H^\ominus = -1202\,\mathrm{kJ\,mol^{-1}} \tag{1.2}$$

ist stark exotherm mit Flammtemperaturen von bis zu 3100 °C und erzeugt ein hoch intensives, weißes „Magnesiumlicht“, welches bei der Blitzerzeugung in der frühen Fotografie eingesetzt wurde [1, 13]. Die Selbstentzündungstemperatur von Magnesiumpulver liegt bei etwa 500 °C [14, 15]. Magnesiumbrände sind schwierig zu löschen, da brennendes Magnesium selbst unter Wasser gemäß

$$\mathrm{Mg(s)} + \mathrm{H_2O(l)} \longrightarrow \mathrm{MgO(s)} + \mathrm{H_2(g)} \tag{1.3}$$

unter Bildung von hoch entzündlichem Wasserstoff zu Magnesiumoxid reagiert [16]. Außerdem reagiert es exotherm mit den üblicherweise inerten Gasen Stickstoff und Kohlenstoffdioxid [2, 17]. Mit Stickstoff bildet sich Magnesiumnitrid ($\mathrm{Mg_3N_2}$) in Folge der Reaktion

$$2\,\mathrm{Mg(s)} + 3\,\mathrm{N_2(g)} \longrightarrow \mathrm{Mg_3N_2(s)} \tag{1.4}$$

und Kohlenstoffdioxid wird gemäß

$$2\,\mathrm{Mg(s)} + \mathrm{CO_2(g)} \longrightarrow 2\,\mathrm{MgO(s)} + \mathrm{C(s)} \tag{1.5}$$

zu Kohlenstoff reduziert [2]. Aus diesen Gründen wird aktiv an schwer entzündlichen Magnesiumlegierungen geforscht [18].

Magnesiumlegierungen spielen insbesondere durch ihre Leichtigkeit eine wichtige Rolle bei der Konstruktion von Flugzeug- oder Automobilbauteilen. Im Jahr 2014 erreichte die weltweite Primärproduktion erstmals eine Million Tonnen [19]. Die stetig steigende Nachfrage nach dem Metall lässt sich teilweise mit dem wachsenden Druck auf die Automobilbranche erklären: Leichtbau-Konstruktionen auf der Basis von Magnesium sollen dabei helfen, verschärfte Brennstoffeffizienz-Vorschriften einzuhalten [19, 20]. Das Innenministerium der Vereinigten Staaten stufte Magnesium im Jahr 2018 im Rahmen der Executive Order 13817 („A Federal Strategy to Ensure Secure and Reliable Supplies of Critical Minerals“) als besonders wichtigen Rohstoff ein [19].

Die Darstellung kleiner Mengen Magnesium in Amalgamform gelang erstmals Sir Humphry Davy, welcher den Elektrolyseprozess in seiner Arbeit zur elektrochemischen Trennung von Erdalkalimetallverbindungen im Jahr 1808 beschrieb.[2]

[2] Die Idee zur Verwendung einer Quecksilberelektrode bekam Davy durch einen Briefwechsel mit Berzelius, der zuvor erfolgreich Barit ($\mathrm{BaSO_4}$) elektrolysierte [21].

> *That the same happy methods must succeed with strontites and magnesia, it was not easy to doubt, and I quickly tried the experiment.*
>
> – Sir Humphry Davy, 1808

Magnesia wurde von Davy als Oxid eines neuen Metalls erkannt, welches er zunächst Magnium nannte [21]. Elementares Magnesium wurde 1828 von Antoine Bussy durch die Reduktion von Magnesiumchlorid ($MgCl_2$) mit Kalium erhalten, was die Grundlage für die erste industrielle Erzeugung des Metalls im Deville-Caron-Prozess schuf (Frankreich, um 1850) [5].

Heutzutage sind die zwei wichtigsten Verfahren zur Herstellung von Magnesium die Elektrolyse von Magnesiumchlorid und die silikothermische Reduktion von Magnesiumoxid nach dem Pidgeon-Prozess. Der Magnatherm-Prozess (Frankreich bis 2001) und der Bolzano-Prozess (Brasilien) sind später entwickelte silikothermische Verfahren von geringer globaler Bedeutung [22]. Die elektrochemische Trennung von $MgCl_2$ geht auf Experimente von Michael Faraday im Jahr 1833 zurück und ist heute Teil des Dow-Verfahrens und verwandter Prozesse [5]. Die einst vorherrschende Elektrolysemethode findet unter anderem in den USA Anwendung, die globale Produktion wird inzwischen durch den Pidgeon-Prozess dominiert [20, 23, 24]. Binnewies u. a. gliedern den Elektrolyseprozess in drei Stufen. Zunächst wird Magnesiumhydroxid ($Mg(OH)_2$) aus einer Suspension aus Meerwasser und vermahlenem Calciumhydroxid ($Ca(OH)_2$) ausgefällt. $Mg(OH)_2$ ist das schlechter löslich als $Ca(OH)_2$ und scheidet sich gemäß

$$Ca(OH)_2(s) + Mg^{2+}(aq) \longrightarrow Ca^{2+}(aq) + Mg(OH)_2(s) \tag{1.6}$$

aus der Suspension ab.[3] Durch den Umsatz des Hydroxids mit Salzsäure entsteht Magnesiumchlorid in Folge der Reaktion

$$Mg(OH)_2(s) + 2\,HCl(aq) \longrightarrow MgCl_2(aq) + 2\,H_2O(l). \tag{1.7}$$

Nach Eindampfung der Lösung wird das Magnesiumchlorid ähnlich wie bei einer Downs-Zelle der Elektrolyse unterzogen, wobei an den Elektroden die folgenden Reaktionen ablaufen

$$Mg^{2+} + 2\,e^- \longrightarrow Mg(l) \quad \text{(Kathode)} \tag{1.8}$$

$$2\,Cl^- \longrightarrow Cl_2(g) + 2\,e^- \quad \text{(Anode).} \tag{1.9}$$

Das freigesetzte Chlorgas kann schließlich wieder zur Gewinnung von Salzsäure eingesetzt werden [1]. Die sehr ähnlichen Abläufe im sogenannten Dow-Verfahren sind in Abbildung 1.2 dargestellt.

Der Pidgeon-Prozess basiert auf der silikothermischen Reduktion von Magnesiumoxid mit Silicium nach der Reaktion

$$2\,MgO(s) + Si(s) \longrightarrow SiO_2(s) + 2\,Mg(g). \tag{1.10}$$

[3] Die Löslichkeitsprodukte von Magnesiumhydroxid und Calciumhydroxid betragen $K_L(Mg(OH)_2) = 1{,}5 \cdot 10^{-12}\,mol^3\,l^{-3}$ und $K_L(Ca(OH)_2) = 3{,}9 \cdot 10^{-6}\,mol^3\,l^{-3}$ [2].

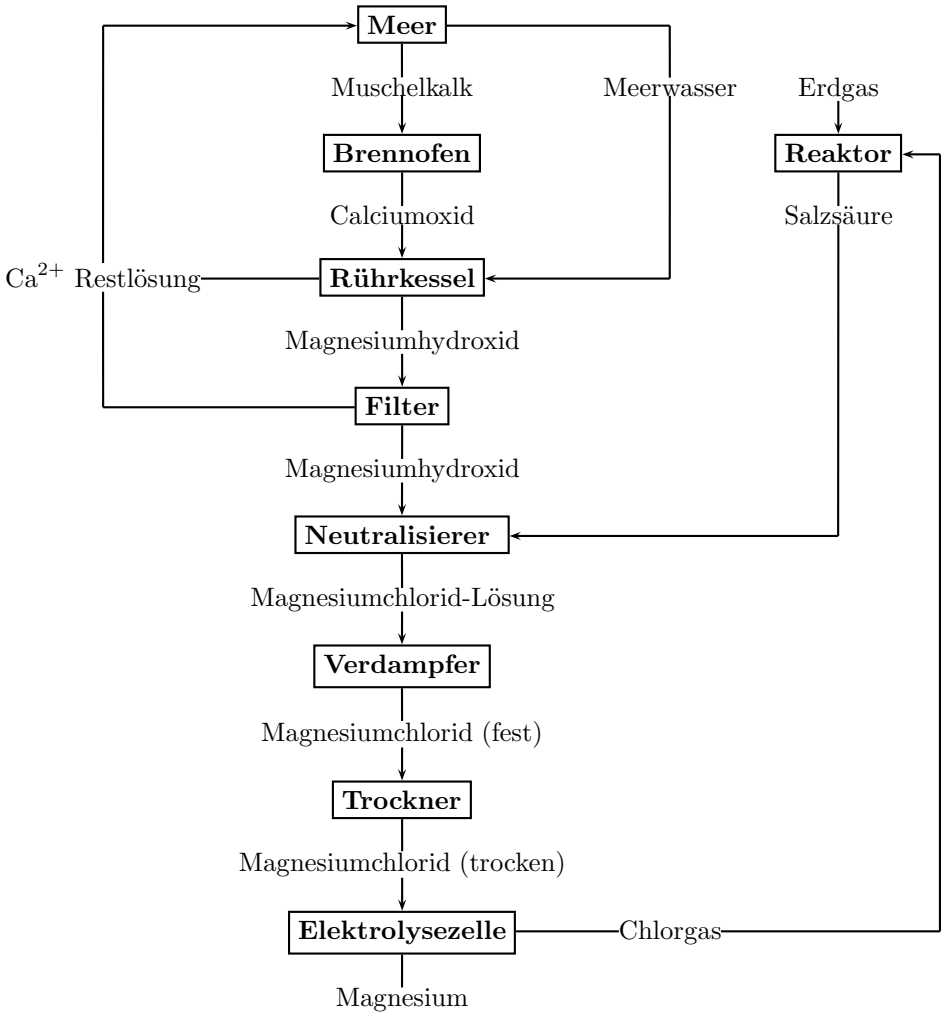


Abbildung 1.2: Prozessabläufe bei der Magnesiumgewinnung im Dow-Verfahren unter Einsatz der Rohstoffe Muschelkalk, Meerwasser und Erdgas. Das Diagramm wurde einer Darstellung in *Magnesium and Magnesium Alloys* nachempfunden [5].

Die technische Realisierung und systematische Untersuchung dieser Reduktion geht auf den kanadischen Chemiker Lloyd Montgomery Pidgeon zurück. Das im Jahr 1942 von Pidgeon eingereichte Patent „Method and Apparatus for Producing Magnesium“ beschreibt einen für die Magnesiumherstellung entwickelten Retortenofen mit Kondensatoreinheit im Detail. Calcinierte magnesiumhaltige Erze werden mit Ferrosilicium (meist notiert als FeSi, Verhältnis der Elemente kann

variieren) bei etwa 1150 °C unter Vakuum reduziert.[4,5] Gasförmiges Magnesium scheidet sich in einer Kondensatoreinheit mit gestaffelten Metallscheiben ab, welche auch Fremdmetalle wie Natrium in getrennten Fraktionen auffängt [25].

Gegenwärtig werden hauptsächlich Dolomiterze als Magnesiumquelle im Pidgeon-Prozess eingesetzt, welche in großen Mengen und guter Reinheit in der Natur vorkommen [5, 26]. Die Calcinierung von Dolomit zu MgO und CaO erfolgt im Brennofen bei etwa 1300 °C [20]. Die Reduktion mit Ferrosilicium gemäß

$$2\,(\mathrm{MgO}\cdot\mathrm{CaO})\,(\mathrm{s}) + \mathrm{FeSi}\,(\mathrm{s}) \longrightarrow 2\,\mathrm{Mg}\,(\mathrm{g}) + \mathrm{Ca_2SiO_4}(\mathrm{s}) + \mathrm{Fe}(\mathrm{s}) \tag{1.11}$$

wurde bereits von Toguri und Pidgeon im Hinblick auf Druck, Temperatur, den Einfluss fluoridhaltiger Katalysatoren und die Zusammensetzung des Ferrosiliciums ausführlich untersucht [26]. Die Magnesiumausbeute erhöht sich im Bereich zwischen 1050 und 1560 °C um einen Faktor von etwa 1,55 bei einer Temperaturerhöhung um 50 °C, mit steigendem Druck sinkt die Ausbeute hingegen. Die Reaktionsrate kann durch Verwendung möglichst reinen Siliciums verbessert werden, ist wirtschaftlich aber nicht sinnvoll. Calciumfluorid (CaF_2) stellte sich gegenüber Bariumfluorid (BaF_2) und Magnesiumfluorid (MgF_2) als der effektivste Katalysator heraus [26]. Die thermodynamischen Gleichgewichte im Kontext von Reaktion 1.11, insbesondere im Hinblick auf den Einfluss von Calcium, werden von Wynnyckyj und Pidgeon im Rahmen einer späteren Arbeit diskutiert [27]. Abbildung 1.3 zeigt einen Überblick der Folge von Prozessabläufen bei der industriellen Realisierung des Pidgeon-Prozesses.[6]

Die weltweite Magnesiumproduktion wird durch China dominiert – 2018 stellte das Land 800 und im Jahr zuvor 930 der weltweiten Produktion von etwa 1000 Kilotonnen her, wobei der Rückgang seitens des U. S. Geological Survey (USGS) durch erhöhte Energiekosten und verschärfte Umweltstandards erklärt wird [19]. Die weltweite Primärproduktion der Jahre 2017 und 2018 ist in Tabelle 1.2 gezeigt. Außerdem sind zum Vergleich die Jahre 1992 und 2002 aufgeführt. Die Vereinigten Staaten produzierten 1992 bis 1998 jeweils zwischen 100 und 140 Kiltotonnen Magnesium und halten ihre Produktionsdaten seit 1999 zurück.[7] Die Daten des USGS basieren zum Teil auf Schätzwerten und Eigenangaben der Länder. Außerdem stehen einige Produkionszahlen nur inklusive der Sekundärproduktion zur Verfügung.[8] Es fällt auf, dass im Zuge der wachsenden Produktion in China viele

[4] Das in FeSi enthaltene Eisen nimmt an Reaktion 1.10 nicht teil. Kohlenstoff und Calciumcarbid (CaC_2) wurden als Reduktionsmittel kommerziell verdrängt [26].

[5] Nach Ramakrishnan u. a. erfolgt die Extraktion des Magnesiumgases aus der Reduktionszone in chinesischen Anlagen bei einem Druck von 1–2 Pa [20].

[6] Details können variieren. Nach Ramakrishnan u. a. werden in China z. B. hauptsächlich Eisenoxide statt Alteisen als Rohstoff bei der FeSi-Herstellung verwendet [20].

[7] Der führende amerikanische Hersteller ist US Magnesium LLC [23].

[8] Eine vollständige Liste mit Anmerkungen und Zusatzinformationen befindet sich im Anhang. Tabelle 6.1 zeigt die Daten für 1992 bis 2018 im Detail.

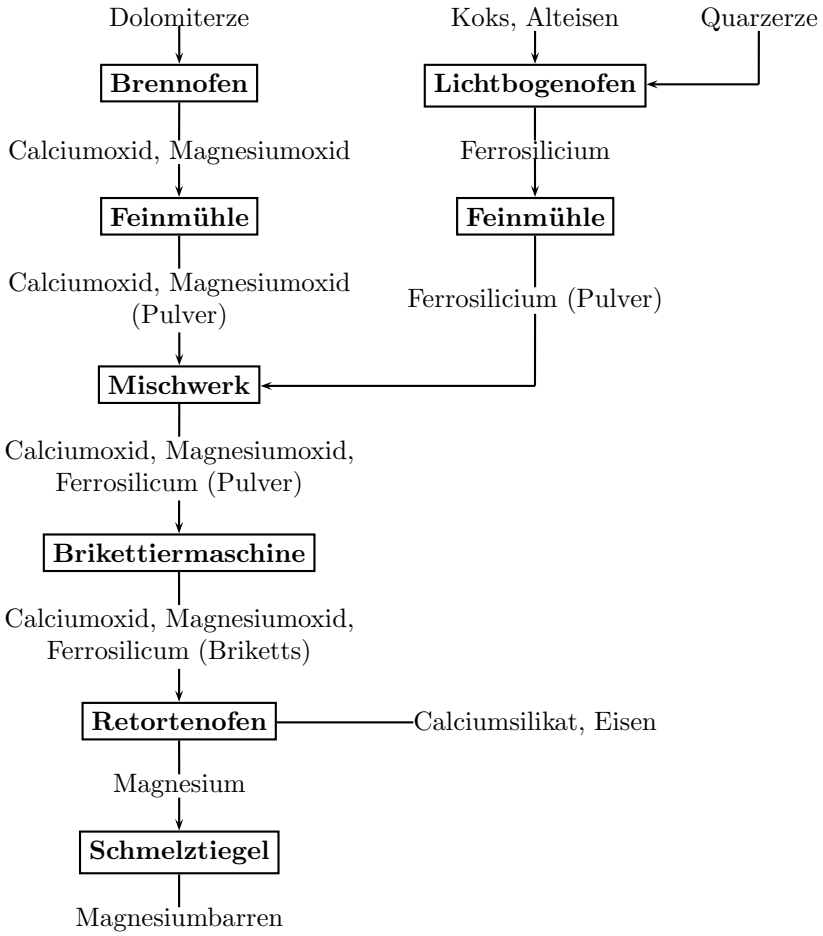


Abbildung 1.3: Prozessabläufe bei der Magnesiumgewinnung im Verfahren nach Pidgeon. Das Diagramm wurde einer Darstellung in *Magnesium and Magnesium Alloys* nachempfunden [5].

Erstwelt-Länder die Darstellung von Magnesium aufgaben. Italien und Japan produzierten bis in die frühen neunziger Jahre, Frankreich und Norwegen stellten die zum Teil beträchtliche Produktion Anfang der zweitausender Jahre ein. Kanada stellte nennenswerte Mengen bis 2008 her. Die Lohn- und Energiekosten dieser Industriestaaten erwiesen sich als nicht konkurrenzfähig gegenüber den günstigen Produktionsbedingungen in China [24]. Abbildung 1.4 veranschaulicht den Aufstieg Chinas zum Weltmarktführer.

Der Pidgeon-Prozess steht in der Kritik, weil er im Vergleich zum Elektrolyseverfahren und den Magnatherm- und Bolzano-Prozessen mit der höchsten Umweltbelastung einhergeht und die geringste Energieeffizienz aufweist [22]. Im Rahmen der Arbeit „Global Warming Impact of the Magnesium Produced in China Using

Tabelle 1.2: Magnesiumproduktion in Kilotonnen [19]. Einige geringfügig produzierende Länder sind nicht aufgeführt.

	1992	2002	2017	2018
China	11	250	930	800
USA	137	–	–	–
Russland	40	40	40	65
Israel	0	26	23	25
Kasachstan	3	18	9	23
Ukraine	15	3	8	19
Brasilien	7	6	15	15
Türkei	0	0	14	10
Südkorea	0	0	10	10
Kanada	25	80	0	0
Norwegen	30	10	0	0
Frankreich	14	0	0	0
Welt	295	432	1050	970

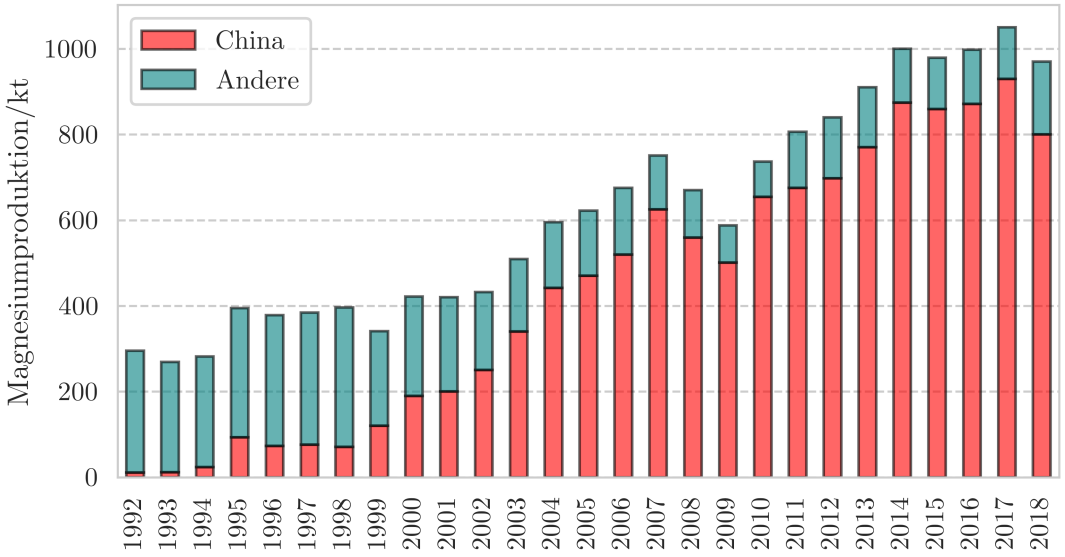


Abbildung 1.4: Weltweite Magnesium-Primärproduktion 1992 bis 2018. Die Daten wurden aus den Berichten des U. S. Geological Survey zusammengestellt [19].

the Pidgeon Process“ führten Ramakrishnan u. a. im Jahr 2004 eine cradle-to-gate Lebenszyklusanalyse des Pidgeon-Prozesses durch, welche alle Prozessschritte bis

zum fertigen Magnesiumbarren berücksichtigt.[9] Die Autoren betrachten einen repräsentativen Modellprozess mit den sechs Schritten

1. Abbau und Transport von Dolomitgestein und Kohle
2. Ferrosilicium-Produktion und -Transport
3. Calcinierung von Dolomit
4. Brikettproduktion
5. Thermischer Reduktionsprozess
6. Magnesiumbarren-Herstellung.

Die FeSi-Produktion, die Calcinierung und die Reduktion stellen sich als die wesentlichen Stufen im Hinblick auf Energiebedarf und Treibhausgasemission heraus. Das verwendete Ferrosilicium hat einen Siliciumanteil von etwa 75 %, dessen Herstellung sich mit den Gleichungen

$$\mathrm{Fe_2O_3(s) + 4\,SiO_2(s) + 11\,C(s) \longrightarrow 2\,FeSi_2(s) + 11\,CO(g)} \tag{1.12}$$

$$\mathrm{SiO_2(s) + 2\,C(s) \longrightarrow Si(s) + 2\,CO(g)} \tag{1.13}$$

beschreiben lässt. Hierbei werden mit einem Energieaufwand von 45 MJ etwa 15 kg CO_2 Äquivalente pro Kilogramm Magnesiumbarren erzeugt ($15\,\mathrm{kg\,CO_2/kg\,Mg}$). Der Wert ergibt sich durch die Nutzung elektrischer Energie aus Kohle und einer Energieeffizienz von etwa 50 % im Lichtbogenofen-Prozess. Die Calcinierung von Dolomit bei 1300 °C folgt der Reaktionsgleichung

$$\mathrm{(CaCO_3 \cdot MgCO_3)(s) \longrightarrow (CaO \cdot MgO)(s) + 2\,CO_2(g)} \tag{1.14}$$

und trägt mit etwa $10\,\mathrm{kg\,CO_2/kg\,Mg}$ zur Ökobilanz bei. Neben der direkten Freisetzung von Kohlenstoffdioxid besteht ein durch Kohle gedeckter Energieaufwand von 50 MJ pro Kilogramm Magnesium (Prozesseffizienz circa 60 %). Der Reduktionsprozess zu elementarem Magnesium geht mit $16\,\mathrm{kg\,CO_2/kg\,Mg}$ in die Lebenszyklusanalyse ein. Pro Kilogramm Magnesium werden 180 MJ Energie aus Kohle eingesetzt, bei einer Energieffizienz von nur 12 %. Für den Gesamtprozess ergeben sich $(42 \pm 5)\,\mathrm{kg\,CO_2/kg\,Mg}$ unter dem Einsatz von 10–15 kg Dolomit, typischerweise 1,1 kg Ferrosilicium und 8–13 kg Kohle. Die Magnesiumproduktion besitzt damit etwa das 1,7 −fache Treibhauspotential der chinesischen Aluminiumproduktion mit $25\,\mathrm{kg\,CO_2/kg\,Al}$ [20].

[9] Seit 2004 hat sich die chinesische Jahresproduktion von etwa 400 auf 800 kt verdoppelt (siehe Abbildung 1.4).

1.2 Motivation und Beschreibung des untersuchten Effekts

Die Reduktion von Magnesiumoxid mit Silicium ist die Schlüsselreaktion bei der Magnesiumherstellung und daher von großer wirtschaftlicher Bedeutung für die Primärproduzenten. Ein Abklingen der Nachfrage nach Magnesium durch die weiterverarbeitende Industrie zeichnet sich nicht ab. Der Einsatz von Leichtbauteilen erhöht die Treibstoffeffizienz von Automobilen und Flugzeugen und reduziert das Gewicht portabler Elektronik. Zugleich beschäftigt sich die Forschung aktiv mit der Entwicklung neuer, immer robusterer Magnesiumlegierungen, deren Einsatz auch dann möglich ist, wenn elementares Magnesium oder einfache Magnesium-Aluminium-Legierungen die Grenze ihrer Belastbarkeit erreichen. Dennoch gibt es kaum physikalisch-chemische Studien, welche den Reduktionsprozess in situ untersuchen.[10] Diese Tatsache ist hauptsächlich der Schwierigkeit geschuldet, Reaktions-Grenzflächen bei hohen Temperaturen zu beobachten.

Eine Anlage, welche die Untersuchung von Metall-Substrat-Grenzflächen unter Reaktionsbedingungen erlaubt, wurde von Lukas Alphei entwickelt. Die Apparatur ermöglicht Hochtemperatur-Auger-Elektronen-Spektroskopie (HTAES) bei gleichzeitiger Beobachtung der Reaktionszone mit einem Videomikroskop. Alphei konzentrierte sich hierbei auf die Reaktionsprodukte beim Kontakt von Siliciumschmelzen mit keramischen Substraten und die Beschreibung der Spreizbewegungen von kleinen Metalltröpfchen unter Reaktionsbedingungen [29, 30]. Bei der Verwendung von transparenten Substraten (MgO, SiO_2) können die lateralen und spreizenden Bewegungen von Tröpfchen während des Experiments von unten durch das Substrat beobachtet werden. Dies ermöglicht das Anvisieren der Schmelze oder entnetzter Bereiche des Substrats mit dem Elektronenstrahl des Auger-Elektronen-Spektrometers (AES). Das Prinzip der Probenbeobachtung ist in Abbildung 1.5 näher dargestellt.

Bei der Untersuchung von Siliciumproben auf Magnesiumoxid-Substraten wurde ein bisher unbeobachteter Effekt festgestellt. Im Zuge des Aufheizvorgangs zeigten sich wiederkehrende Lichtemissionserscheinungen, die in Echtzeit mit dem Videomikroskop verfolgt werden konnten. Die Probe vollzieht plötzliche Helligkeitssprünge, welche sich kurz vor Erreichen der Schmelztemperatur besonders ausgeprägt zeigen. Umgangssprachlich kann das Emissionsverhalten als „Blinkeffekt“ bezeichnet werden.

[10] Einige Ergebnisse der Untersuchungen wurden vom Autor schon im Artikel „Transient Light Emission from the Silicothermic Reduction of Magnesium Oxide with Potential for Monitoring Intermediate Compound Formation and Decay“ veröffentlicht [28]. Dieser Publikation entnommene Abbildungen sind entsprechend gekennzeichnet. Der folgende Text kann Formulierungen mit englischer Entsprechung im Artikel enthalten. Auf die nochmalige Zitierung der Publikation wird in diesen Fällen verzichtet. Die vollständige Beschreibung der Ergebnisse wird in der Dissertation vorgenommen.

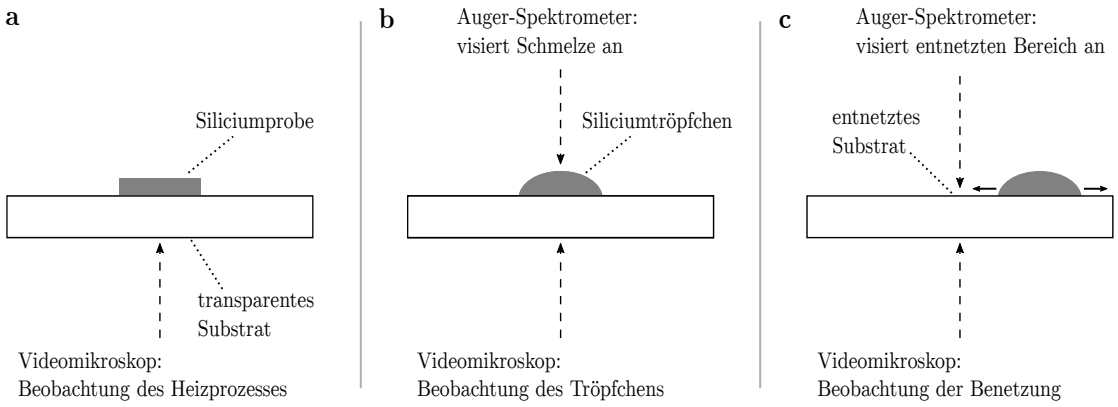


Abbildung 1.5: Geometrie der Probenobservation im HTAES-Experiment nach Alphei. **a** Probenkontrolle in der Aufheizphase. **b** Reaktive Benetzung, AES zielt auf Schmelze. **c** Reaktive Benetzung und Tröpfchenbewegung, AES zielt auf Substrat. Die Abbildung wurde bereits in ähnlicher Form veröffentlicht [28].

Alpheis Beobachtung konnte im Rahmen von Vorversuchen mit unregelmäßig geformten Siliciumspänen (20–100 µg) auf MgO-Plättchen reproduziert werden. Auf dem Substratmaterial platzierte Siliciumkörnchen wurden bei verschiedenen, über Minuten konstant gehaltenen Heizleistungen mit dem Videomikroskop aufgezeichnet (Beobachtung gemäß Abbildung 1.5 **a**). Bei Vergleichsexperimenten mit Quarzsubstraten zeigten sich keine ungewöhnlichen Emissionserscheinungen. Aus den Videodaten wurden Bilderserien zur Illustration des Blinkeffekts erstellt. Üblicherweise folgen sehr schnelle Helligkeitssprünge auf vorangegangene „Dunkelphasen“. Die Probe verliert im Laufe einiger Sekunden an Helligkeit (Intensität) und vollzieht dann einen abrupten Intensitätssprung. Abbildung 1.6 zeigt eine Bilderserie über sechs Sekunden und einer zeitlichen Auflösung von 200 ms einer kleinen Siliciumprobe auf Magnesiumoxid. Bei 3400 , 5400 und 5600 ms sind deutliche Änderungen der Emissionsintensität der Probe zu beobachten. Der Messrechner erlaubt die Aufnahme mit sechzig Bildern pro Sekunde. Abbildung 1.7 stellt eine mit 60 Hz aufgelöste Bilderserie der gleichen Probe über 150 ms (und anderem Startzeitpunkt) dar. Auch bei dieser zeitlichen Auflösung sind noch abrupte Sprünge in der Intensität (50 ms, 117 ms) zu erkennen.

Die folgenden Betrachtungen widmen sich der experimentellen Untersuchung der Lichtemission aus der MgO/Si-Grenzfläche und der Formulierung einer Folge von chemischen Reaktionen, welche einen Beitrag zur Aufklärung des Emissionseffekts liefern können.

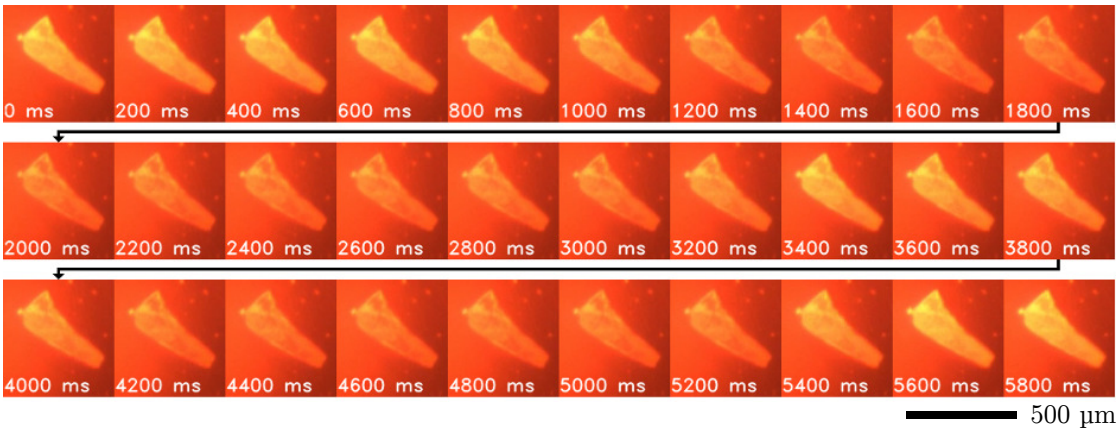


Abbildung 1.6: Zeitserie eines Siliciumkörnchens (40 µg) auf einem MgO-Substrat bei konstanter Heiztemperatur in der Nähe des Schmelzpunkts. Einzelbilder sind mit der vergangenen Zeit seit einem willkürlich gewählten Start bei 0 ms annotiert. Die zeitliche Auflösung innerhalb einer Bilderreihe beträgt 200 ms. Bilder in derselben Spalte besitzen einen Abstand von zwei Sekunden. Diese Zeitserie wurde bereits veröffentlicht [28].

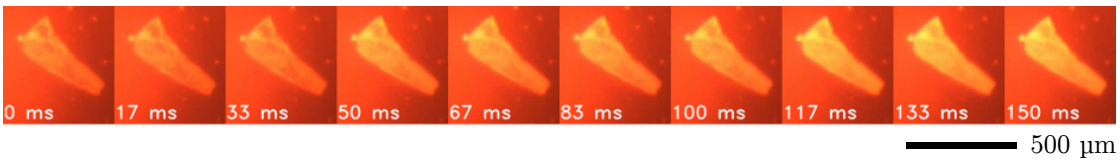


Abbildung 1.7: Zeitserie der Probe aus Abbildung 1.6 mit höherer Auflösung (60 Hz). Diese Zeitserie wurde bereits veröffentlicht [28].

2 Experimenteller Teil

2.1 Beschreibung der Hochvakuum-Apparatur

Die Experimente werden in der von Alphei beschriebenen Hochvakuum-Apparatur durchgeführt [29]. Bei früheren Versuchen hat sich herausgestellt, dass gasförmige Reaktionsprodukte wie Mg(g) und SiO(g) die hoch empfindliche Ionenoptik des Auger-Spektrometers beschädigen. Insbesondere metallische Reaktionsprodukte können bei Ablagerung an den schwer zugänglichen, inneren Bauteilen des Spektrometers zu Aufladungseffekten und Ablenkung von Elektronen-Flugbahnen führen. Das Spektrometer wurde daher mit einer Blende aus Molybdän-Folie abgeschirmt und in den im Folgenden beschriebenen Experimenten nicht verwendet. Zusätzlich wurden von außen mehrere schwere Metallgewichte mit Ketten an die Apparatur gehängt, um vibrationsbedingte Bewegungen des Probenhalters zu minimieren. Die Auswertung des im Experiment gesammelten Videomaterials gestaltet sich umso einfacher, je weniger die Probe ihre Position bezüglich der Bildkoordinaten ändert.

Abbildung 2.1 zeigt einen schematischen Querschnitt der Anordnung von Substrathalter mit Substratplättchen und Probe auf dem Heizelement (Momentive BORALECTRIC). Sowohl der Heizer als auch der Substrathalter bestehen aus Bornitrid (BN). Der Substrathalter besitzt eine Vertiefung für das Substratplättchen mit einer kreisförmigen, etwa 4 mm breiten Öffnung. Die leicht vorstehende Platzierung des Halters auf dem Heizer ermöglicht die Beobachtung der Probe mit Hilfe des Videomikroskops durch die Öffnung und das transparente Substratmaterial (gestrichelter Pfeil). Auf dem Substrathalter ist eine dünne Tantalfolie angebracht, die der thermischen Isolation dient.[1] Die Folie besitzt ein 4 mm breites Stanzloch auf Höhe der Probe. Details zur Befestigung der einzelnen Elemente oder zur Stromversorgung sind in Abbildung 2.1 nicht dargestellt.

Eine detaillierte Aufsicht auf den Probentisch ist durch Abbildung 2.2 gegeben. Das Stanzloch in der Tantalfolie wird mit der Beobachtungsöffnung im Substrathalter (siehe Abbildung 2.1) zur Deckung gebracht. Der unterhalb des Substrathalters befindliche Teil des Molybdän-Hitzeschildes (der obere Teil ist in Abbildung 2.3 zu sehen) besitzt ebenfalls eine rundliche Öffnung auf gleicher Höhe, um die

[1] Bei Einsatz des Auger-Spektrometers verhindert der Kontakt der Metallfolie mit dem Substrat außerdem statische elektrische Aufladung.

Beobachtung der Probe von unten zu ermöglichen. Die Manipulation des Probentisches in x-Richtung erfolgt mit Hilfe einer Magnetschubstange. Der Tisch kann außerdem in y-Richtung bewegt werden. Ein Metallhaken greift in ein sich in der Probenkammer befindliches Gegenstück, welches von außen einige Millimeter mit Hilfe einer Mikrometerschraube bewegt werden kann. Die exakte Manipulation der Schraube ist mit Hilfe eines kleinen Schrittmotors möglich, für dessen Ansteuerung ein LabVIEW-Programm geschrieben wurde [31]. Die Bewegung der Probe in y-Richtung wurde hauptsächlich bei früheren Experimenten mit dem Auger-Spektrometer verwendet, um unterschiedliche Bereiche von Probe oder Substrat mit dem Elektronenstrahl anzuvisieren. Auch wenn das Spektrometer nicht verwendet wird, ist die Bewegung des Probentisches in y-Richtung oft hilfreich, um dessen finale Position zu justieren. Ist der Probentisch im Laufe des Experiments Vibrationen ausgesetzt, können diese durch eine kleine Änderung der y-Position oft gemindert werden.

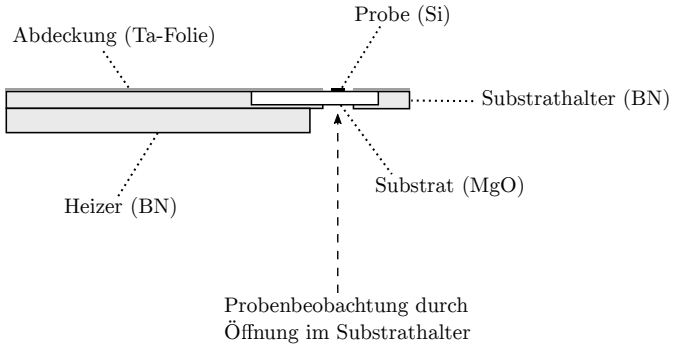


Abbildung 2.1: Querschnitt von Substrat mit Probe in Substrathalter und Platzierung auf Bornitrid-Heizer. Die Abbildung wurde bereits in ähnlicher Form veröffentlicht [28].

Abbildung 2.3 zeigt einen Querschnitt des Probentisches in der Probenkammer. Die Kammer wird mit zwei Scrollpumpen[2] auf ein Vorvakuum von etwa 10^{-2} mbar evakuiert. Das Hochvakuum (etwa 10^{-7} mbar) wird von vier Turbomolekularpumpen[3] aufrecht erhalten. Die Beobachtung der Probe erfolgt durch ein Saphirglasfenster sowie kreisförmige Öffnungen im Molybdän-Hitzeschild und Substrathalter. Beim Videomikroskop handelt es sich um ein Lichtmikroskop der Firma Novex, auf dem eine Kamera (AlliedVision Guppy-F146) mit Verbindung zum Messrechner montiert ist.

[2]Varian SH-110, Varian TriScroll 300
[3]Agilent TwisTorr 84 FS, Varian TV 301 Navigator, Varian TV 70 LP, Varian TV 551 Navigator

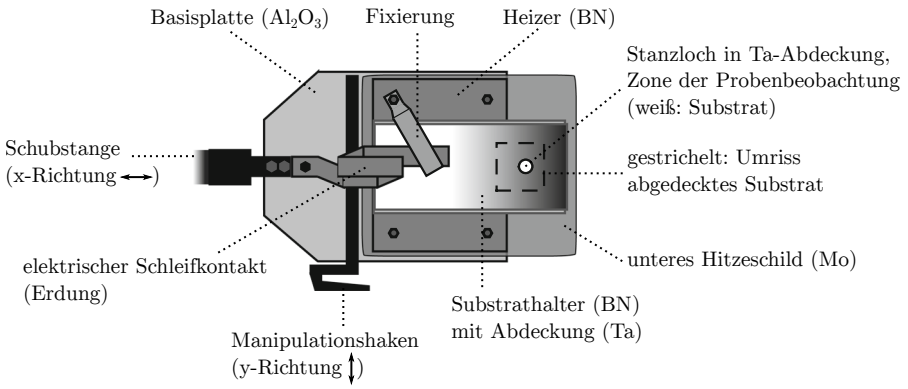


Abbildung 2.2: Detaillierte Aufsicht auf den Probentisch. Die Abbildung wurde in abgewandelter Form von Alphei u. a. entnommen [29].

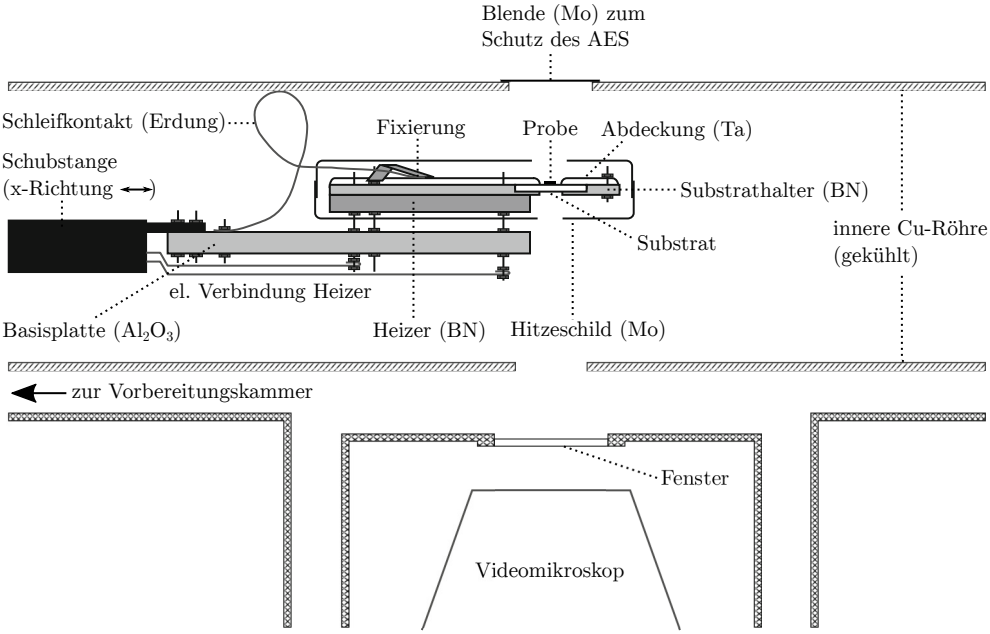


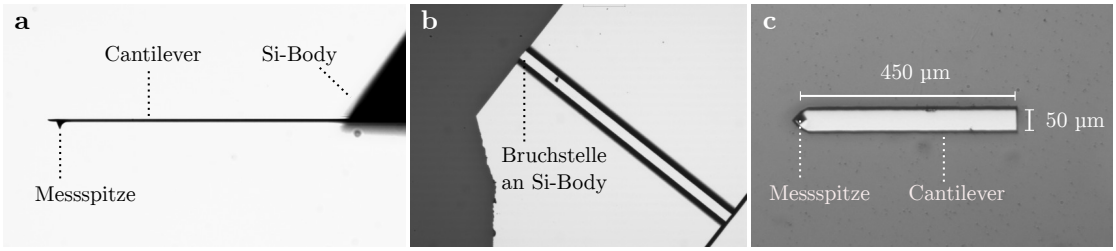
Abbildung 2.3: Querschnitt der Probenkammer. Die Abbildung wurde in abgewandelter Form von Alphei u. a. entnommen [29].

2.2 Probenpräparation

Bei den Vorversuchen mit kleinen Siliciumkörnchen (siehe Abbildung 1.6 und 1.7) stellte sich heraus, dass der Blinkeffekt besonders gut mit leichten, flachen Proben mit ebener Auflagefläche reproduzierbar ist. Bei den anfänglich verwende-

ten Siliciumspänen handelt es sich um Bruchstücke eines größeren Einkristalls. Die Stücke sind unregelmäßig geformt und variieren im Gewicht (20–100 µg). Aufgrund der schlechten Auflagefläche springen sie schon bei bei leichten Erschütterungen während des Experiments aus dem Beobachtungsfeld des Videomikroskops.[4]

Aus diesem Grund wurden in den nachfolgend beschriebenen Experimenten Silicium-Cantilever (NANOSENSORS PPP-CONT) als Proben verwendet. Hierbei handelt es sich um kleine, sehr dünne Stege aus Silicium mit den Abmessungen 450 µm × 50 µm × 2 µm. Sie werden in der Kraftmikroskopie (Atomic Force Microscopy, AFM) eingesetzt, um die Topologie von fein strukturierten Oberflächen zu vermessen. Sie besitzen am Kopf eine Messspitze und sind am anderen Ende fest mit einem Silicium-Basiskörper (Si-Body) verbunden (Abbildung 2.4 **a**). Bei der Probenpräparation muss der Cantilever vom Si-Body abgebrochen und auf das Substratplättchen befördert werden. Abbildung 2.4 **b** zeigt einen Si-Body nach dem Abbrechen des Cantilevers, letzterer wird dann mit der Messspitze nach oben zeigend auf das Substrat aufgebracht (Abbildung 2.4 **c**). Die Verwendung von Cantilevern an Stelle von Siliciumkörnchen gewährleistet eine ebene Auflagefläche auf dem Substrat und eine einheitliche Probengeometrie.[5]



(Bild **c**) Adaptiert/Übersetzt mit Genehmigung von Springer Nature: Springer SN Applied Sciences [28], 2020

Abbildung 2.4: Lichtmikroskopaufnahmen zur Beschreibung der Probengeometrie. **a** Cantilever an Si-Body (Seitenansicht). **b** Si-Body, Cantilever abgebrochen (Aufsicht). **c** Cantilever auf MgO-Substrat (Aufsicht). Teilbild **c** wurde bereits veröffentlicht [28].

Die Platzierung eines Cantilevers auf einem Substratplättchen ist mit präparativem Aufwand verbunden. Jegliche Berührung des Cantilevers, zum Beispiel mit einer Nadel oder Pinzettenspitze, führt zum Abbrechen vom Silicium-Body. Typischerweise springt der Cantilever hierbei mehrere Zentimeter in eine unvorhersehbare Richtung. Daher wurde das Ablösen zunächst in einer Glaspetrischale durchgeführt, um die Probe anschließend auf das Substratplättchen zu befördern. Der Cantilever ist allerdings zu dünn (2 µm), um mit einer Pinzette aus der Petrischale gehoben zu werden. Aufgrund seines geringen Gewichts und ebenen Auflagefläche haftet er zudem am Glas, lässt sich also nicht durch Schütteln, Klopfen

[4] Das Hochschalten der Drehzahl einer Turbomolekularpumpe oder ein draußen vorbeifahrender LKW reichen bereits aus.

[5] Unsauber abgebrochene Cantilever-Proben können kürzer als 450 µm sein.

oder Umdrehen der Schale kontrolliert bewegen. Mit mäßiger Erfolgsaussicht kann versucht werden, den Cantilever in der Petrischale vorsichtig mit einer Nadel oder Pinzettenspitze zu berühren. Nach etlichen Versuchen bleibt der Cantilever dabei an der Spitze der Pinzette hängen und wird dann sehr vorsichtig auf das Substrat abgestreift. Meistens zerbricht der Cantilever im Laufe dieser Vorgehensweise, wird mit Staub verunreinigt oder geht verloren. Ohne Erfolg war auch der Versuch, die Petrischale mit einem Bohrloch zu versehen und den Cantilever dann durch das Loch auf das Substrat zu schieben. Der Cantilever bleibt am Rand oder an der Innenseite des Lochs hängen.

Schließlich wurde eine Platzierungsmethode entwickelt, die (mit etwas Übung) stets zum Erfolg führt. Ein Substratplättchen (MgO, 10 mm × 10 mm × 0,5 mm) wird zehn Minuten im Ultraschallbad mit Ethanol gereinigt. Anschließend wird es wie in Abbildung 2.5 gezeigt in einem Probenhalter mit besonders tiefer Auskerbung platziert. Die Auskerbung wird dann mit Ethanol aufgefüllt. Ein Silicium-Body (mit Cantilever) wird mit Hilfe einer Pinzettenspitze am Boden des Substrats fixiert. Unter Ethanol kann der Cantilever nun ohne wegzuspringen an seiner Basis mit einer zweiten Pinzettenspitze abgebrochen werden. Dieser Schritt erfolgt am besten unter Beobachtung im Lichtmikroskop. Der Si-Body wird zügig herausgenommen um ein Anhaften des Cantilevers zu vermeiden. Es muss darauf geachtet werden, dass der umherschwimmende Cantilever nicht den Rand des Substrathalters berührt, da er nur schwer ohne Beschädigung oder Verunreinigung wieder abzulösen ist. Durch vorsichtiges Berühren der Flüssigkeit mit einer Pinzettenspitze kann die Bewegungsrichtung des Cantilevers kontrolliert werden. Ist das Ethanol verdampft, bleibt der Cantilever auf dem Substrat liegen und kann mit einer Nadel oder einem Einhaarpinsel an die finale Position verschoben werden. Mit dem Lichtmikroskop wird kontrolliert, ob der Cantilever flach aufliegt. Die Messspitze am Kopf des Cantilevers muss hierfür wie in Abbildung 2.4 **c** nach oben (vom Substrat weg) zeigen. Erfahrungsgemäß richtet sich der Cantilever immer von alleine so aus, dass er flach auf dem Substrat zum Liegen kommt. Bei einigen Hilfsexperimenten verbleibt das Substratplättchen im tiefen Substrathalter, ansonsten wird es in einen flacheren Halter umgesetzt.

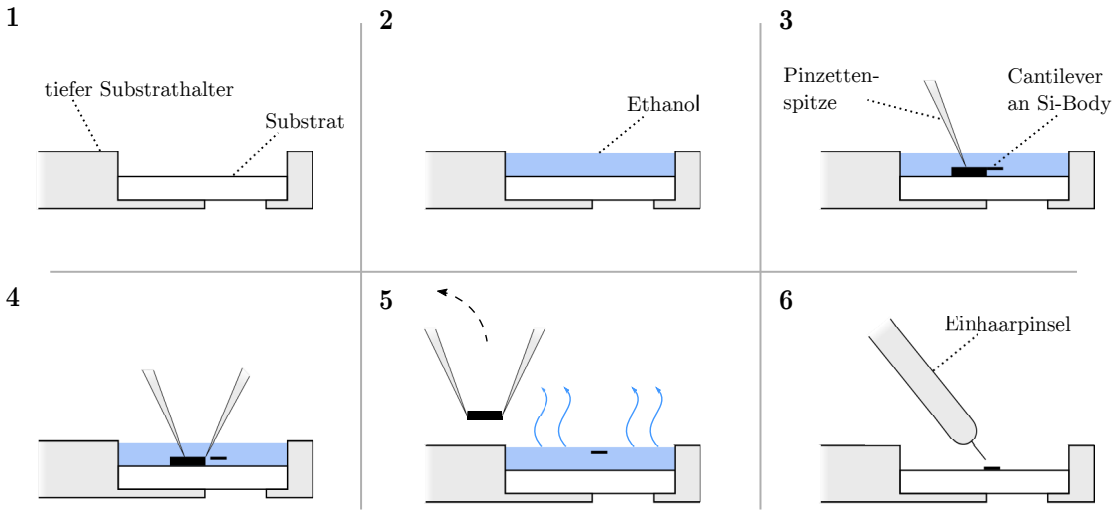


Abbildung 2.5: Cantilever-Platzierung auf Substrat. **1** Substrat in tiefem Substrathalter. **2** Substrathalter mit Ethanol aufgefüllt. **3** Fixierung von Si-Body mit Cantilever durch Pinzettenspitze. **4** Ablösen des Cantilevers. **5** Entfernung des Si-Bodys. Ethanol verdampft. **6** Finale Platzierung mit Einhaarpinsel.

2.3 Durchführung des Hauptexperiments

Das Hauptexperiment besteht in der Aufzeichnung des Emissionsverhaltens eines Silicium-Cantilevers auf einem beidseitig polierten Magnesiumoxid-Substrat (Orientierung <100>) mit Hilfe eines Videomikroskops mit Verbindung zum Messrechner. Die Auswertung der Videodaten erfolgt nach dem Experiment mit einer eigens entwickelten Software, welche in Kapitel 4 beschrieben wird. Der Substrathalter wird gemäß Abbildung 2.1 auf dem Heizelement platziert. Nach Aufsetzen eines Molybdän-Hitzeschildes wird die Anordnung dann mit einer Magnetschubstange über der Kamera positioniert (siehe Abbildung 2.3). Durch Kontrolle des Kamerabildes auf dem Messrechner wird sichergestellt, dass das Beobachtungsloch im Substrathalter korrekt über der Kamera positioniert und der Substrathalter nicht verrutscht ist. Hierfür wird vorübergehend eine Ringleuchte am Videomikroskop in Betrieb genommen und die Kamera mit Hilfe der Software am Messrechner auf maximale Lichtempfindlichkeit eingestellt. Bei korrekter Positionierung kann die Apparatur nun auf etwa 10^{-7} mbar evakuiert werden und die Wasserkühlung wird in Betrieb genommen.

Die Widerstandsheizung wird über eine von Hand regelbare Spannungsquelle angesteuert. Zunächst wird über einen Zeitraum von etwa 20–30 Minuten die Heizspannung vorsichtig von null auf etwa 30 V (Stromfluss ca. 5 A) erhöht und danach die Videoaufzeichnung gestartet. Im Laufe des Experiments wird die Heizspannung jetzt schrittweise erhöht und dann jeweils über mehrere Minuten konstant gehal-

ten. Typische Werte sind hier Spannungsschritte von 2,5 V alle fünf Minuten bis zu einer Spannung von 45 V (Stromfluss ca. 8–8,5 A). Im Bereich dieser Heizleistung schmilzt das Silicium erfahrungsgemäß. Die Uhrzeit, die aktuelle Heizspannung mit korrespondierendem Stromfluss, der Druck (typischerweise 10^{-7}–10^{-6} mbar) und Beobachtungen (Cantilever-Aktivität, Bildjustagen, Erschütterungen, etc.) werden im Laborjournal festgehalten. Ist das gewünschte Videomaterial aufgezeichnet, wird das Experiment durch langsames Herunterregeln der Heizspannung über einen Zeitraum von 20–30 Minuten beendet. Ein abruptes Ausschalten der Widerstandsheizung kann zum Platzen des Substratplättchens führen.

Abbildung 2.6 zeigt das während des Experiments aufgezeichnete Kamerabild. Die Software am Messrechner wurde so konfiguriert, dass Datum und Uhrzeit automatisch in das Video eingefügt werden. Außerdem werden im Video die aktuelle Heizspannung und zugehörige Stromstärke sowie der Status der Ringlampe am Videomikroskop und einige Videoparameter dokumentiert. Heiz- und Videoparameter werden während des Experiments von der Aufnahmesoftware aus einer Textdatei gelesen und dann automatisch in das Videobild eingefügt. Wird beispielsweise die Heizspannung erhöht, muss der neue Wert in die Datei eingetragen werden. Zum Vergleich der Probenhelligkeiten bei verschiedenen Heizleistungen müssen Parameter, welche die Helligkeit des Bildes beeinflussen (Shutter, Gamma, etc.), während des Experiments konstant gehalten werden. Einerseits wird ein möglichst helles Bild angestrebt, um das Leuchten der Probe bei niedrigen Heizleistungen beobachten zu können. Andererseits dürfen keine Kameraeinstellungen gewählt werden, die bei hohen Heizleistungen zu einer Sättigung des roten Farbkanals (der Kanal mit der höchsten Empfindlichkeit) führen. Diese macht sich durch das Erreichen des maximalen Werts von 255 im RGB-Farbsystem bei der Auswertung der Einzelbilder bemerkbar. Ein Sprung in der Probenhelligkeit kann durch Analyse des roten Farbkanals nicht bemerkt werden, wenn der Wert 255 bereits erreicht wurde. Die in Abbildung 2.6 verwendeten Parameter wurden durch Ausprobieren gefunden und gewährleisten die Aufzeichnung auswertbaren Videomaterials zwischen etwa 30–45 V Heizspannung.

Der Emissionseffekt kann mit Silicium-Cantilevern reproduziert werden. Abbildungen 2.7 und 2.8 zeigen typische Beispiele des transienten Emissionsverhaltens, analog zu den Abbildungen 1.6 und 1.7 mit Siliciumkörnchen.

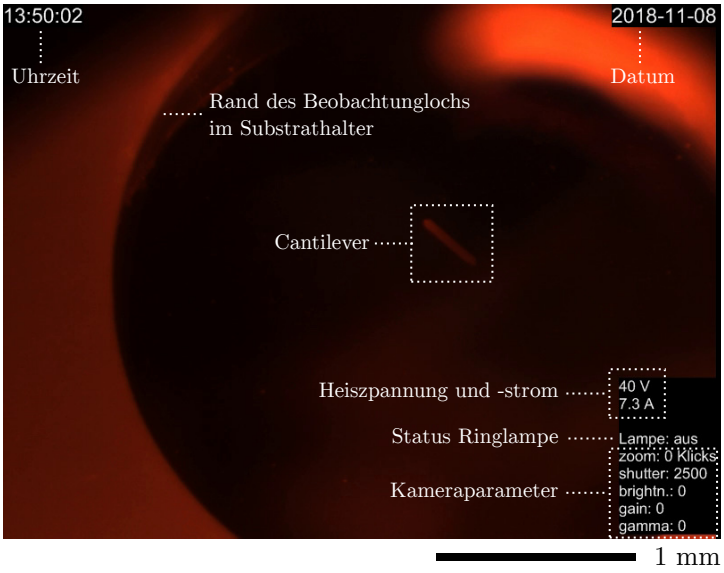
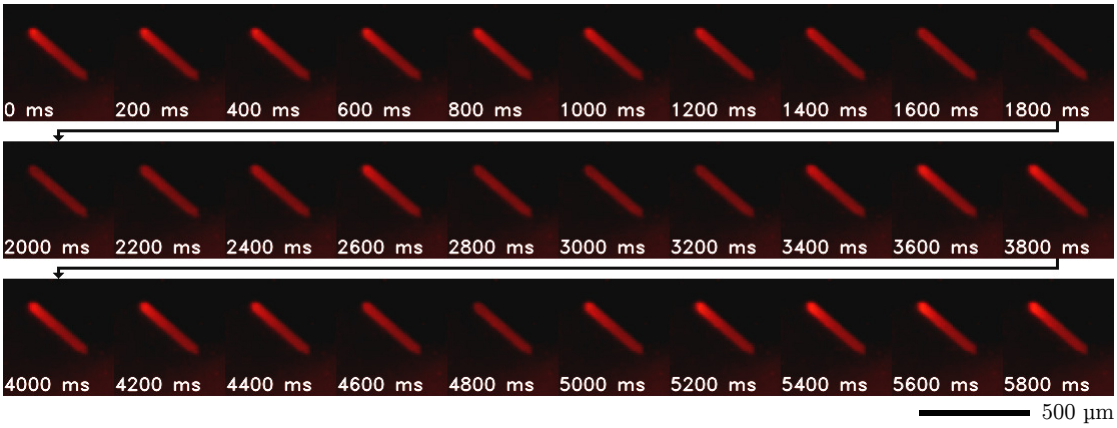


Abbildung 2.6: Probenbeobachtung während des Experiments.



Adaptiert mit Genehmigung von Springer Nature: Springer SN Applied Sciences [28], 2020

Abbildung 2.7: Zeitserie eines Silicium-Cantilevers auf einem MgO-Substrat bei konstanter Heiztemperatur in der Nähe des Schmelzpunkts. Einzelbilder sind mit der vergangenen Zeit seit einem willkürlich gewählten Start bei 0 ms annotiert. Die zeitliche Auflösung innerhalb einer Bilderreihe beträgt 200 ms. Bilder in derselben Spalte besitzen einen Abstand von zwei Sekunden. Helligkeit und Kontrast wurden so angepasst, dass der Effekt besonders deutlich zu erkennen ist. Diese Zeitserie wurde bereits veröffentlicht [28].

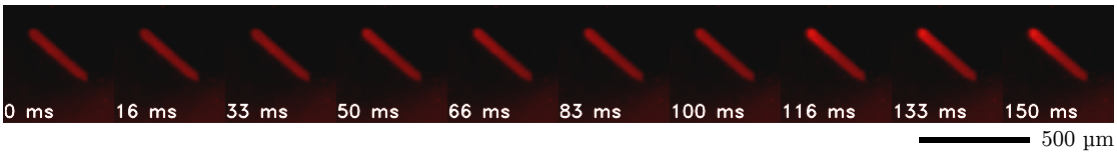


Abbildung 2.8: Zeitserie der Probe aus Abbildung 2.7 mit höherer Auflösung (60 Hz). Helligkeit und Kontrast wurden so angepasst, dass der Effekt besonders deutlich zu erkennen ist. Diese Zeitserie wurde bereits veröffentlicht [28].

2.4 Beschreibung der Hilfsexperimente

2.4.1 Tempern

Die Durchführung der Hauptexperimente beinhaltet die Erhöhung der Heizleistung im zeitlichen Abstand von wenigen Minuten. Darüber hinaus wurde eine Reihe von Langzeit-Temperexperimenten von Silicium-Cantilevern auf MgO-Substraten durchgeführt. Bei diesen Experimenten wurde die Heizleistung über einen Zeitraum von fünf Stunden jeweils konstant gehalten. Durch Ausprobieren verschiedener Heizspannungen zwischen 22,5 V (Stromfluss ca. 4 A) und 35 V (Stromfluss ca. 6–6,5 A) zeigte sich, dass ab einer Heizspannung von etwa 35 V eine chemische Reaktion einsetzt, bei welcher der Cantilever eine Ätzgrube im Substratmaterial hinterlässt. Ab etwa dieser Heizleistung setzt erfahrungsgemäß auch der transiente Emissionseffekt fühestens ein (siehe auch Abbildung 5.9). Die Untersuchung der Ätzgruben ex situ mit mikroskopischen Methoden wird in Kapitel 3 beschrieben.

2.4.2 Andere Substrate

Zum Hauptexperiment analoge Versuche wurden mit Substraten aus amorphem SiO_2, kristallinem SiO_2 und pyrolytischem Bornitrid (PBN) durchgeführt. Der transiente Emissionseffekt, der sich bei Silicium-Proben auf MgO-Substraten zeigt, konnte beim Einsatz dieser Substrate während des Experiments nicht beobachtet werden. Die nachträgliche Auswertung des Videomaterials mit der in Kapitel 4 beschriebenen Software nach dem Vorgehen aus Kapitel 5 bestätigte, dass der Emissionseffekt nur mit MgO-Substraten auftritt. Trotzdem wurden die Proben zu Vergleichszwecken mikroskopisch untersucht (Kapitel 3).

Außerdem wurde festgestellt, dass der Emissionseffekt auch mit MgO-Substraten der Orientierung <111> und <110> auftritt (im Hauptexperiment werden standardmäßig <100>-Substrate verwendet).

2.4.3 Reaktionszellen

In der Anfangsphase wurde mit verschiedenen Zellenaufbauten experimentiert. Eine Reaktionszelle besteht aus drei gestapelten MgO-Substratplättchen. In der Mittelplatte befinden sich ein oder mehrere kleine, mithilfe eines Diamantkopfbohrers angefertigte Löcher. Diese Löcher bieten einen Freiraum für die Platzierung verschiedener Proben (Metallkörnchen oder Silicium-Cantilever) auf dem unteren Substratplättchen. Auf diese Weise können mehrere, räumlich getrennte Reaktionskammern während desselben Experiments beobachtet werden. Abbildung 2.9 zeigt schematisch den Querschnitt des Zellenaufbaus. Experimente mit Zellen wurden

mit einem Probenhalter durchgeführt, dessen Vertiefung drei gestapelte Substratplättchen fasst.

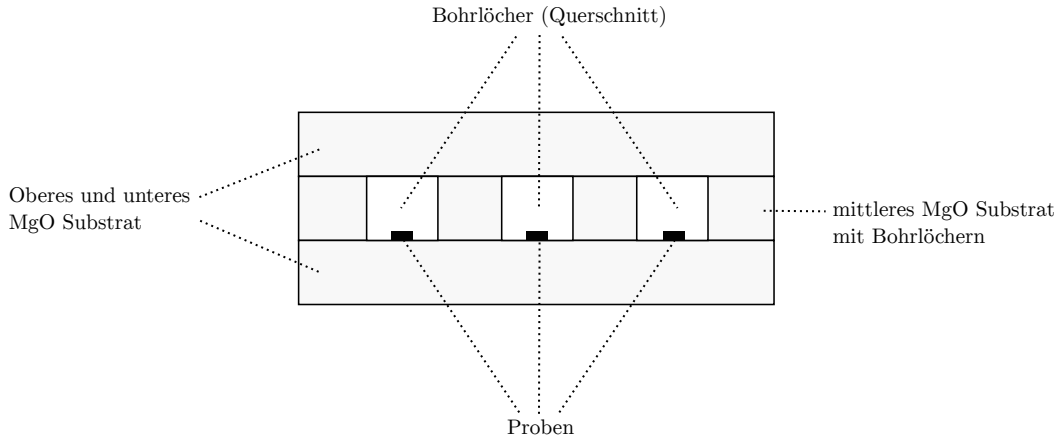


Abbildung 2.9: Zellenaufbau aus drei MgO-Substraten für drei Proben (Querschnitt).

Mit der Verwendung der Zelle wurden zwei Ziele verfolgt. Als noch Metallkörnchen statt Cantilevern als Proben verwendet wurden, mussten Experimente regelmäßig abgebrochen werden, weil schon geringe Vibrationen an der Anlage (z. B. durch das Hochschalten der Drehzahl einer Turbomolekularpumpe) zum Verrutschen oder Wegspringen der Proben führen konnten. Des Weiteren bestand die Hoffnung, durch den Einsatz weiterer Metallproben mit niedrigerem Schmelzpunkt als Silicium die Zuordnung von Heizleistungen zu Probentemperaturen zu unterstützen.[6] Hierbei kamen unter anderem Germanium und Mangan zum Einsatz.[7] Abbildung 2.10 zeigt eine Reaktionszelle im Einsatz.

Leider kann der Blinkeffekt innerhalb dieses Aufbaus nicht zuverlässig reproduziert werden. Anscheinend ist der Abtransport von gasförmigen Reaktionsprodukten kritisch für das Ablaufen einer mit dem Effekt korrelierten chemischen Reaktion und selbst eine locker aufgelegte MgO-Deckelplatte unterdrückt den Abtransport. Diese Vermutung wird durch die Tatsache bestätigt, dass in den Reaktionskammern geschmolzene Metallkörnchen für gewöhnlich keine charakteristischen

[6] Eine generelle, experimentunabhängige Zuordnung von Heizleistungen zu Probentemperaturen, zum Beispiel durch das Aufschmelzen verschiedener Metalle in einer Reihe von Experimenten, wurde als zu ungenau angesehen. Probengeometrie, -auflagefläche, genaue Positionierung von Probe und Probenhalter sowie die Dicke des verwendeten Probenhalters und die Kontaktierung des Heizers besitzen einen Einfluss auf die erreichten Temperaturen.

[7] $T_{\mathrm{smp,Ge}} = 1211\,\mathrm{K}$, $T_{\mathrm{smp,Mn}} = 1519\,\mathrm{K}$, $T_{\mathrm{smp,Si}} = 1687\,\mathrm{K}$ [4]

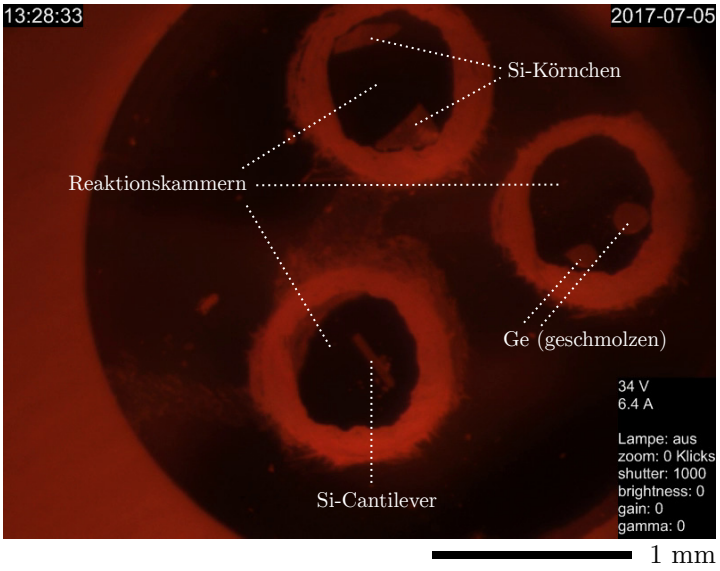


Abbildung 2.10: Reaktionszelle nach Abbildung 2.9.

Spreizbewegungen im Zuge reaktiver Benetzung zeigen.[8] Abhilfe hätte hier ein Anritzen des oberen MgO-Substrats oder die Verwendung von inerten Abstandshaltern schaffen können. Auch ein vorsichtiges Ansintern der Metallproben und anschließende Entfernung des Deckels wären denkbar.

Aufgrund des hohen Präparationsaufwands wurden die Experimente mit Reaktionszellen nicht weiterentwickelt. Eine Vorbereitung nach Abbildung 2.5 lässt sich hingegen zügig wiederholen, falls ein Experiment fehlschlägt. Der Einsatz von Cantilevern gewährleistete die Ortsfestigkeit der Probe auf dem Substrat. Die Probentemperatur konnte pyrometrisch bestimmt werden (Kapitel 5).

2.4.4 Eingeklemmte Proben

Es wurde eine Reihe von Experimenten mit Silicium-Proben (Körnchen und Cantilever) durchgeführt, bei denen diese zwischen zwei MgO-Substratplättchen eingeklemmt wurden. Der transiente Emissionseffekt wurde in keinem dieser Experimente beobachtet.

[8] Weiß u. a.diskutieren in „The Influence of Crystallographic Orientation on the Wetting of Silicon on Quartz Single Crystals“ die reaktive Benetzung von Si-Schmelzen auf SiO_2-Substraten [32]. Dobbe u. a. beschreiben in „Reactive Wetting Controlled by Very Small Vertical Temperature Gradients in a Chemical Transport Mini Reactor“ das reaktive Benetzungsverhalten von Metallschmelzen in kleinen, öffen- und verschließbaren Zellenreaktoren [33].

2.4.5 Angeraute Substratoberfläche

Um den Einfluss des Kontakts der Probe zur Substratoberfläche einzuschätzen, wurden Experimente mit MgO-Substraten durchgeführt, deren Oberflächen mit einem Diamantkopfbohrer angeraut wurden (Abbildung 2.11). Es zeigte sich, dass Cantilever auf rauen Oberflächen bereits bei kleinen Erschütterungen des Probentischs dazu tendieren, vom Substrat zu springen. Daher wurden diese Experimente nicht weitergeführt. In insgesamt zwei Versuchen wurde kein transienter Emissionseffekt beobachtet.

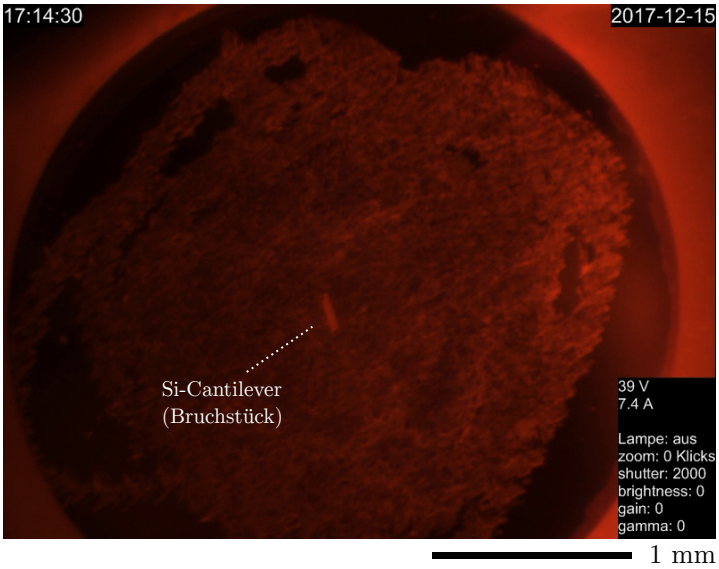


Abbildung 2.11: Experiment mit Si-Cantilever auf angerautem MgO-Substrat.

2.4.6 Cantilever auf Substratunterseite

Im Gegensatz zu Silicium-Körnchen haften Cantilever am Substratplättchen und fallen nicht herunter, wenn dieses umgedreht wird. Um die Hypothese zu untersuchen, dass es sich beim transienten Emissionseffekt lediglich um ein kurzzeitiges Abheben des Cantilevers (und damit Verlust von thermischem Kontakt) vom Substrat in Folge einer Gasentwicklung handelt, wurden zwei Experimente mit dem Cantilever auf der Substratunterseite durchgeführt (Abbildung 2.12). Bei vollständigem Kontaktverlust zum Substrat würde in diesem Fall das Herunterfallen der Probe erwartet werden, sobald der Effekt einsetzt. In beiden Versuchen blieb die Probe ortsfest und zeigte das typische Blinken.

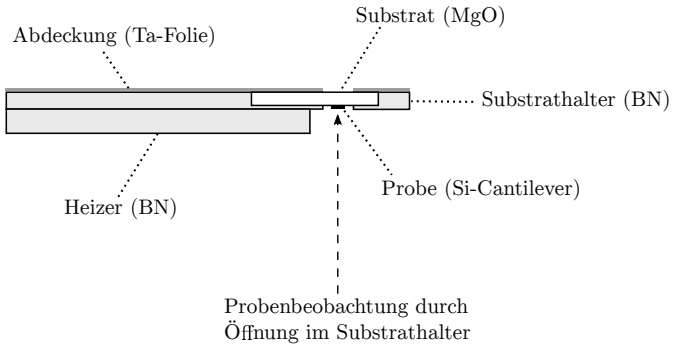


Abbildung 2.12: Darstellung des Probenhalters analog zu Abbildung 2.1, aber mit Si-Cantilever auf der Unterseite statt der Obserseite des Substrats.

2.4.7 Hohe Heizleistungen

Wird das Hauptexperiment beim Erreichen des Schmelzpunktes nicht abgebrochen oder die Heizleistung sogar weiter erhöht, kommt es häufig vor, dass sich der Cantilever teilweise von der Substratoberfläche ablöst. Ein Beispiel ist in Abbildung 2.13 gezeigt. Meistens kommen die abgelösten Teilstücke des Cantilevers wieder auf dem Substrat zum Liegen oder brechen in der Abkühlphase ab und gehen verloren.

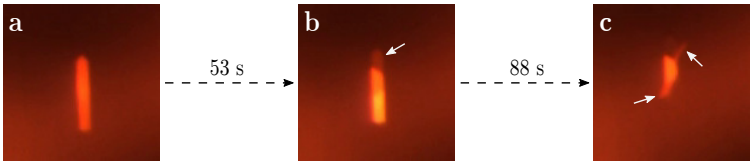


Abbildung 2.13: Ablösen des Cantilevers bei hohen Heizleistungen. **a** Cantilever liegt auf. **b** Teilweise Ablösung vom Substrat im oberen Bereich (weißer Pfeil). **c** Ablösung und Verdrehung im oberen und unteren Bereich.

Bemerkenswert ist, dass die abgelösten Teilstücke des Cantilevers den Blinkeffekt zeigen, obwohl sie keinen direkten Kontakt zum Substrat besitzen. Anscheinend ist die Wärmeleitfähigkeit innerhalb der Probe so hoch, dass sich eine Temperaturänderung an den Kontaktstellen mit dem Substrat über den gesamten Körper der Probe beobachten lässt.

Aus Experimenten mit Siliciumkörnchen war bereits bekannt, dass Proben auch im angeschmolzenen Zustand den transienten Emissionseffekt zeigen können (meistens mit hoher Frequenz, d. h. Blinken mehrmals pro Sekunde).

2.4.8 Siliciumstaub

Beim Einsatz von Siliciumstaub (sehr feine Körnchen, 200 Mesh) konnten die Partikel auf dem Substrat nicht sauber getrennt werden. Der transiente Emissionseffekt wurde nicht beobachtet.

2.4.9 In situ Massenspektrometrie

Durch die Montage eines Quadrupol-Massenspektrometers (QMS) über der Probe in einer Vorkammer zum AES wurde versucht, eine mögliche Korrelation des Blinkeffekts mit der Entstehung der gasförmigen Produkte der Reaktion

$$\mathrm{MgO\,(s)} + \mathrm{Si\,(s)} \longrightarrow \mathrm{Mg\,(g)} + \mathrm{SiO\,(g)} \tag{2.1}$$

zu überprüfen. Der Messrechner wurde dafür so konfiguriert, dass eine gleichzeitige Beobachtung der Probe mittels Videomikroskopie und des QMS-Signals in Echtzeit während des Experiments möglich ist.

Es ist bekannt, dass die Entstehung von $\mathrm{SiO\,(g)}$ mit dem Umfang der Dreiphasengrenze bei der reaktiven Benetzung von flüssigem Silicium auf Quarzsubstraten korreliert und mithilfe von Massenspektrometrie nachgewiesen werden kann [34]. Dies verlangt allerdings flüssiges Silicium und größere Proben. Eine Korrelation der fest-fest Reaktion 6.3 mit dem spontanen Abdunkeln oder Aufleuchten einer Silicium-Probe konnte mit dem QMS leider nicht beobachtet werden. Es ist möglich, dass im Zuge eines Blinkvorgangs zu geringe Mengen an Reaktionsprodukten freigesetzt werden.

In einem verbesserten Versuchsaufbau für zukünftige Massensprektrometrie-Experimente wäre die Montage eines dünnen Keramikröhrchens direkt über der Probe denkbar, um Reaktionsprodukte aufzufangen und Störkomponenten abzuschirmen.

2.4.10 Gekreuzte Cantilever

Es wurde ein Experiment mit zwei gekreuzt aufeinander liegenden Cantilevern durchgeführt (Abbildung 2.14). Es kommt vor, dass beide Cantilever in etwa zur gleichen Zeit an Intensität verlieren und dann wieder aufleuchten, aber es wurden auch viele unkorrelierte Blinkvorgänge beobachtet. Letzteres könnte an einem schlechten thermischen Kontakt an der Kreuzungsstelle liegen. Das Experiment wurde nicht wiederholt. Die Präparation ist sehr aufwendig und mit einem hohen Verbrauch an Cantilevern verbunden.

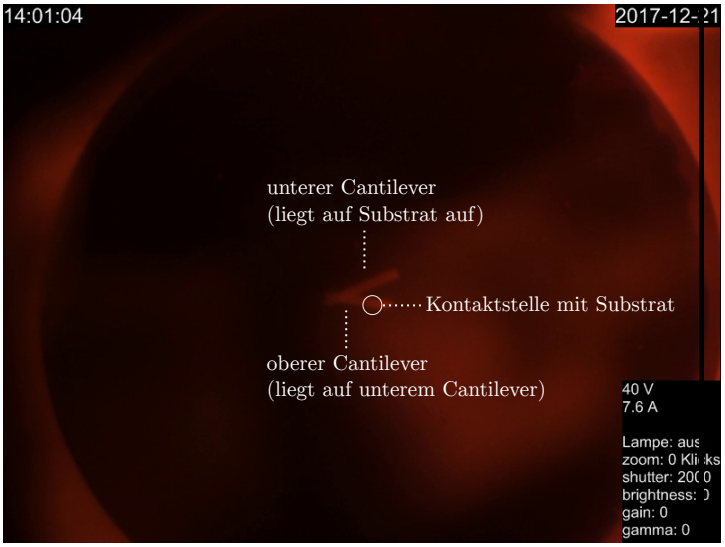


Abbildung 2.14: Experiment mit gekreuzten Cantilevern.

3 Elektronenmikroskopie und EDX-Spektroskopie

Die Arbeit am Rasterelektronenmikroskop JEOL JSM-6700F geschah mit freundlicher Unterstützung der Arbeitsgruppe von Herrn Professor Feldhoff des Instituts für Physikalische Chemie und Elektrochemie der Leibniz Universität Hannover. Die Messungen am Transmissionselektronenmikroskop JEOL JEM-2100F-UHR wurden von Herrn Professor Feldhoffs Arbeitsgruppe vorgenommen. Die Arbeit mit dem Gerät ZEISS AURIGA wurde vom Institut für Werkstoffkunde der Leibniz Universität durchgeführt.

3.1 Rasterelektronenmikroskopie

3.1.1 Silicium-Körnchen

Die Aufnahmen 3.1 bis 3.6 zeigen ein Silicium-Körnchen nach einem Experiment gemäß Abschnitt 2.3. Das Körnchen wurde nach dem Experiment vom Substrat entfernt und auf einen Probenträger für das Mikroskop aufgeklebt.

Die beiden sichtbaren Flächen sind mit **A** und **B** gekennzeichnet. Fläche **A** zeigt noppenartige Kristallitstrukturen, auf Fläche **B** finden sich Löcher, feine Körnerpartikel und kleine Kristallite. Aus Experimenten mit Cantilevern ist bekannt, dass sich auf der dem Substrat abgewandten Seite der Probe ähnliche Kavitäten mit sehr glatten Oberflächen bilden. In den Folgeabschnitten wird dieser Effekt mithilfe weiterer REM-Aufnahmen verdeutlicht.

Auf beiden Flächen wurden EDX-Messungen aller relevanten Merkmale (Oberfläche, Löcher, Kristallite) durchgeführt. Hierbei wurde jeweils ausschließlich Silicium mit Spuren von Sauerstoff gefunden. Magnesium konnte nicht nachgewiesen werden.

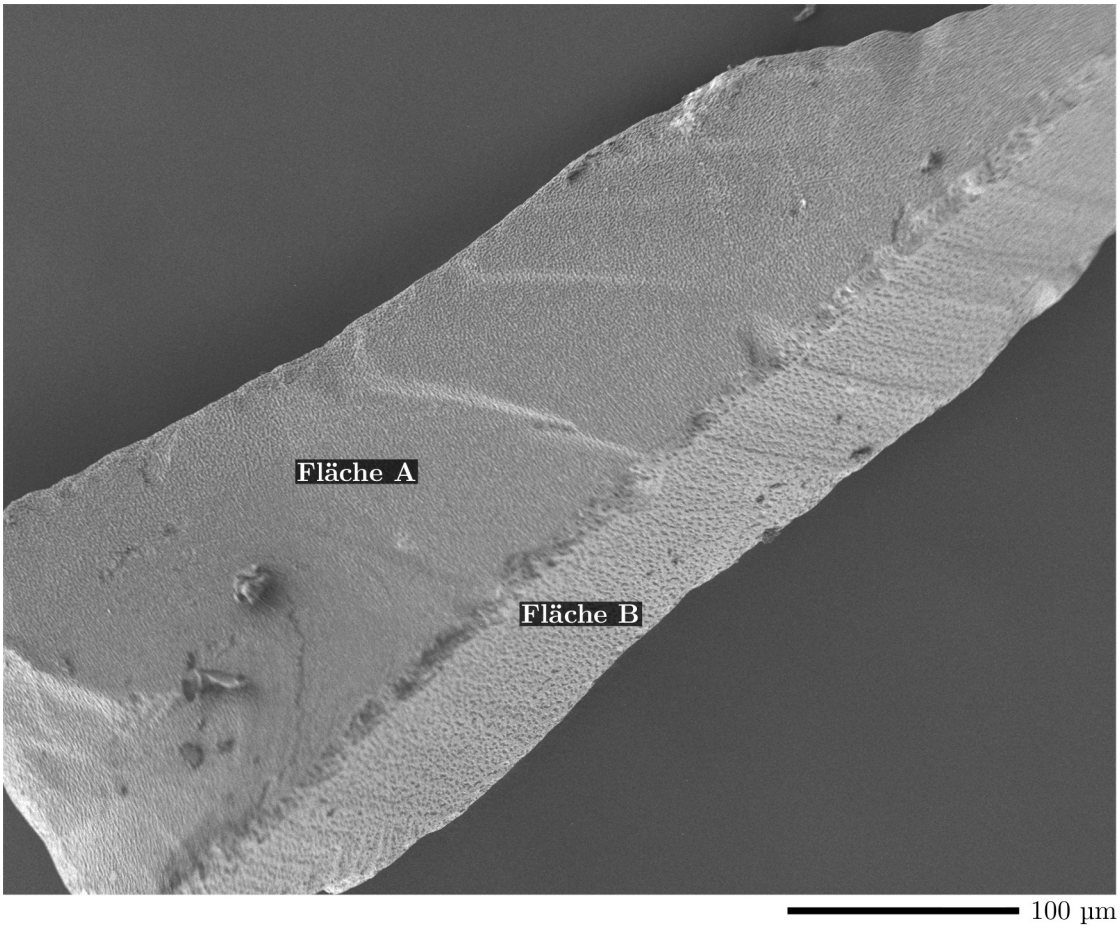


Abbildung 3.1: REM-Aufnahme (JEOL JSM-6700F). Silicium-Körnchen nach dem Experiment. Die Flächen **A** und **B** werden in den Folgebildern genauer betrachtet.

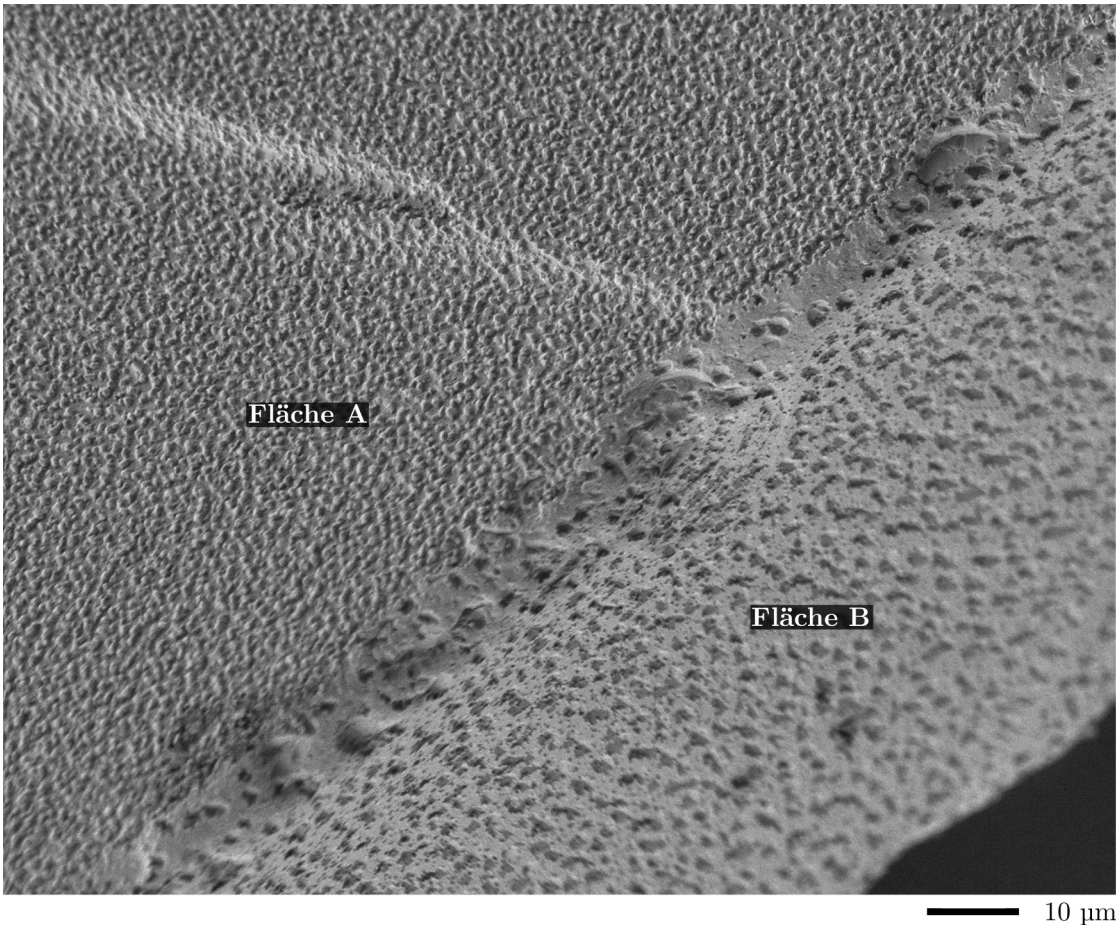


Abbildung 3.2: REM-Aufnahme (JEOL JSM-6700F). Übergangskante zwischen den Flächen **A** und **B**. Bei der gegebenen Auflösung sind bereits körnerartige Erhöhungen auf Fläche **A** zu erkennen. Bei den dunkel erscheinenden „Flecken“ auf Fläche **B** handelt es sich um im Zuge des Experiments gebildete Löcher.

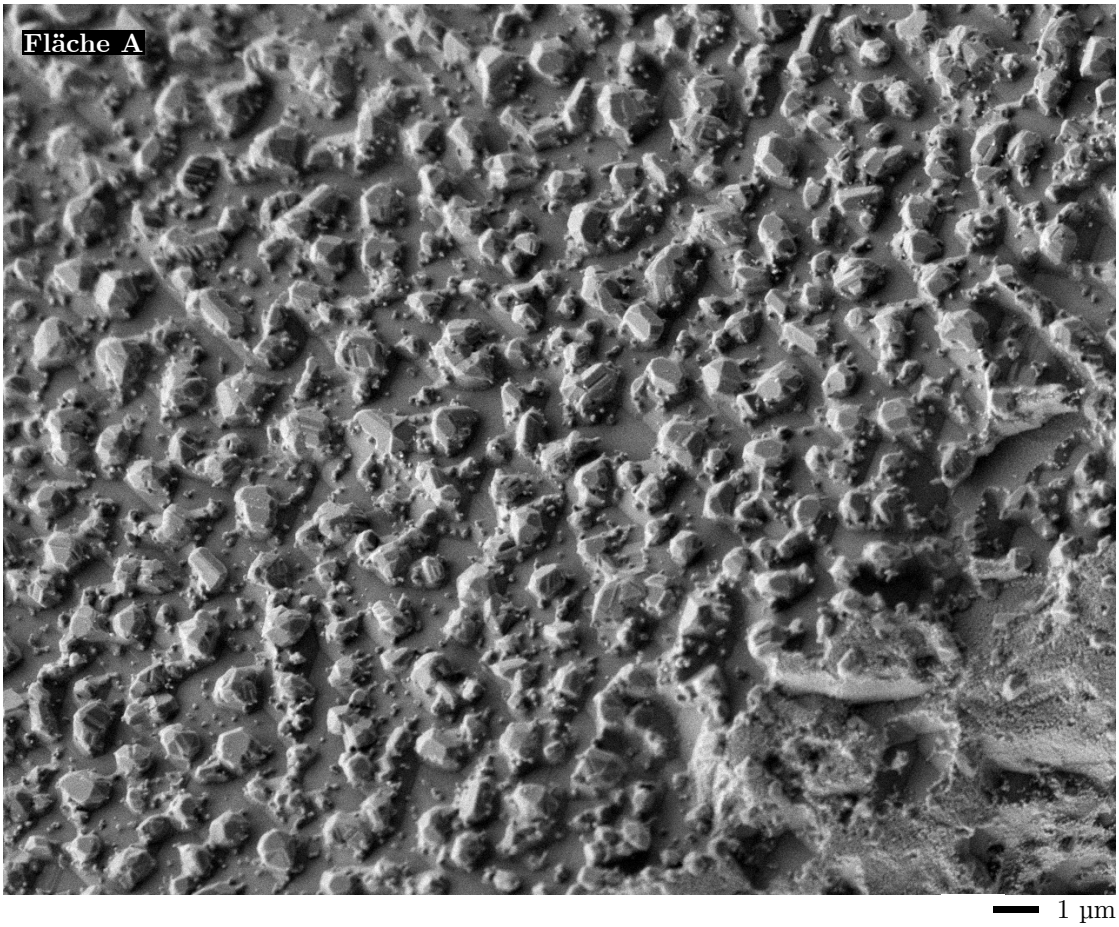


Abbildung 3.3: REM-Aufnahme (JEOL JSM-6700F). Fläche **A** bei höherer Auflösung. Die Oberfläche ist mit kleinen, auf ihr angewachsenen Kristalliten übersäht. Die Zwischenräume erscheinen glatt.

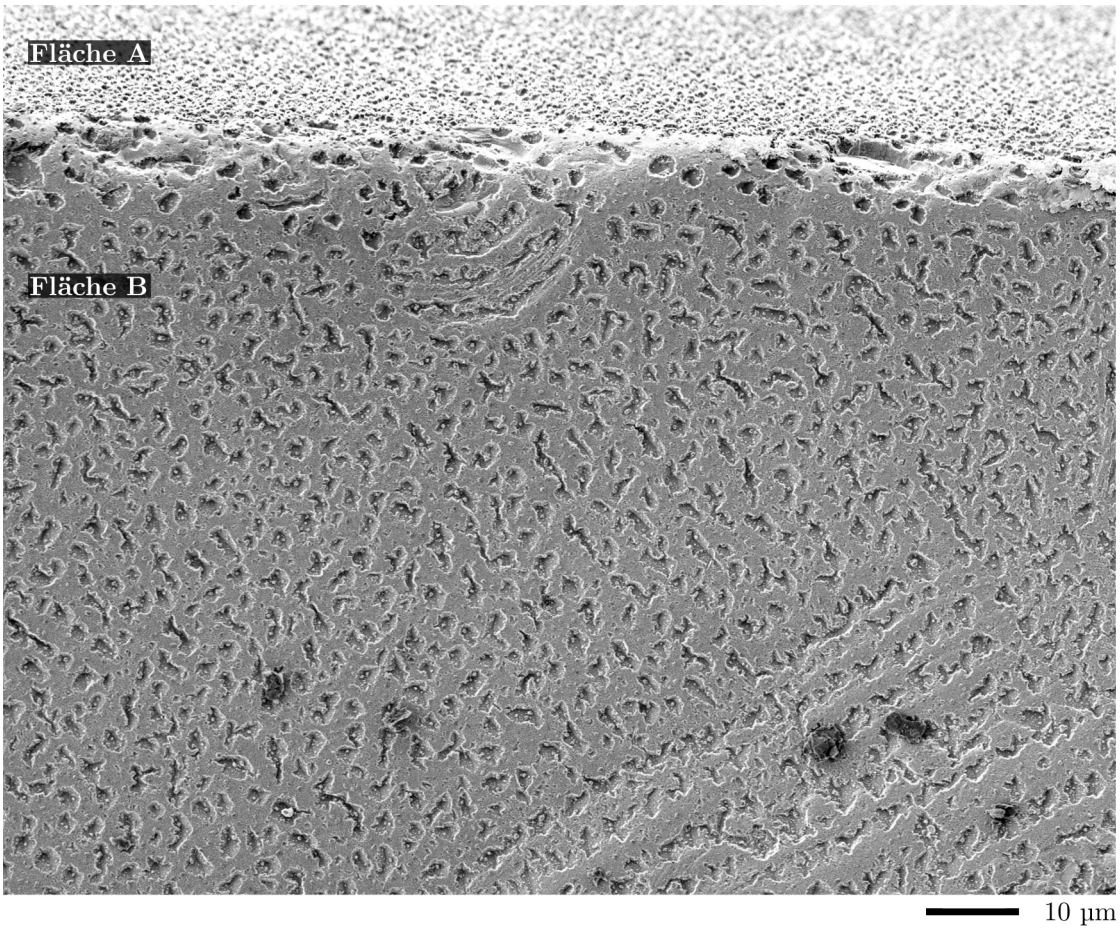


Abbildung 3.4: REM-Aufnahme (JEOL JSM-6700F). Aufsicht auf Fläche **B**.

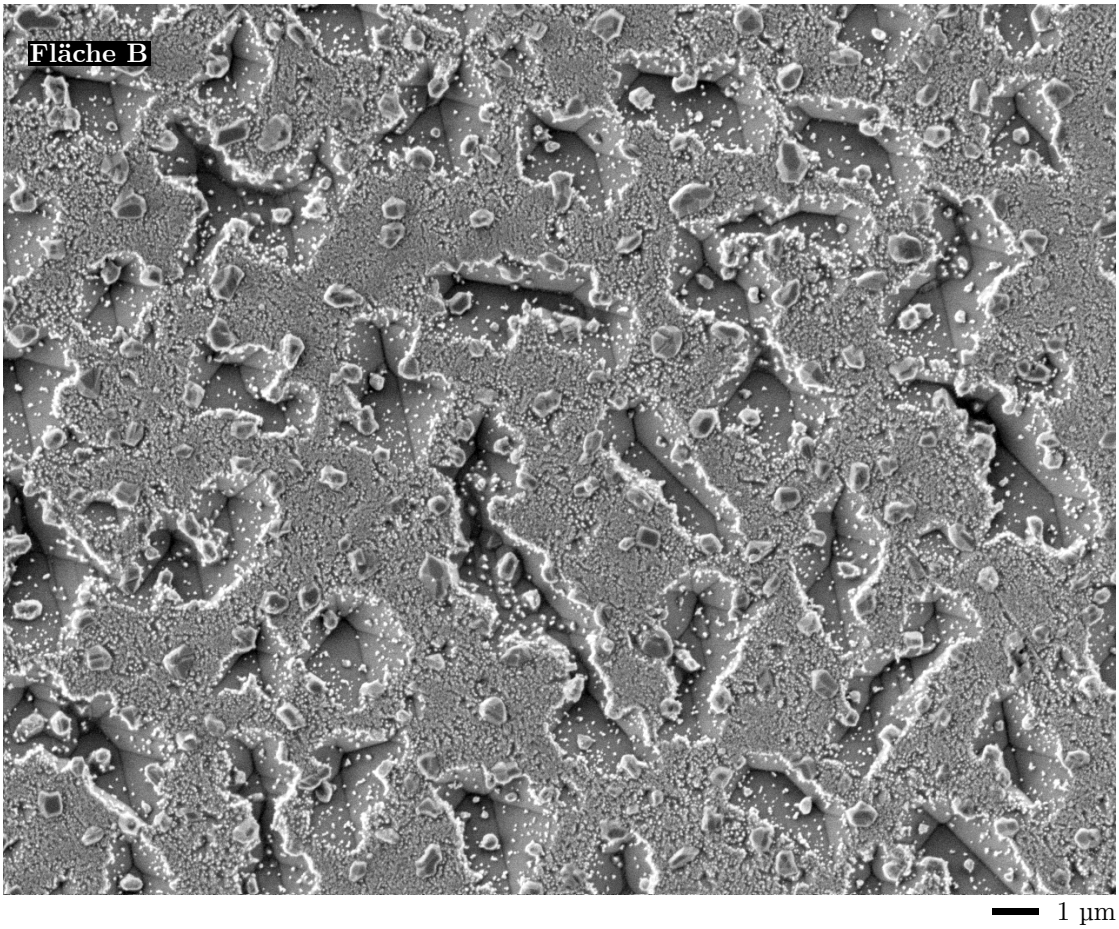


Abbildung 3.5: REM-Aufnahme (JEOL JSM-6700F). Kavitäten in Fläche **B**. Im Vergleich zu Fläche **A** ist die Oberfläche rau und körnig.

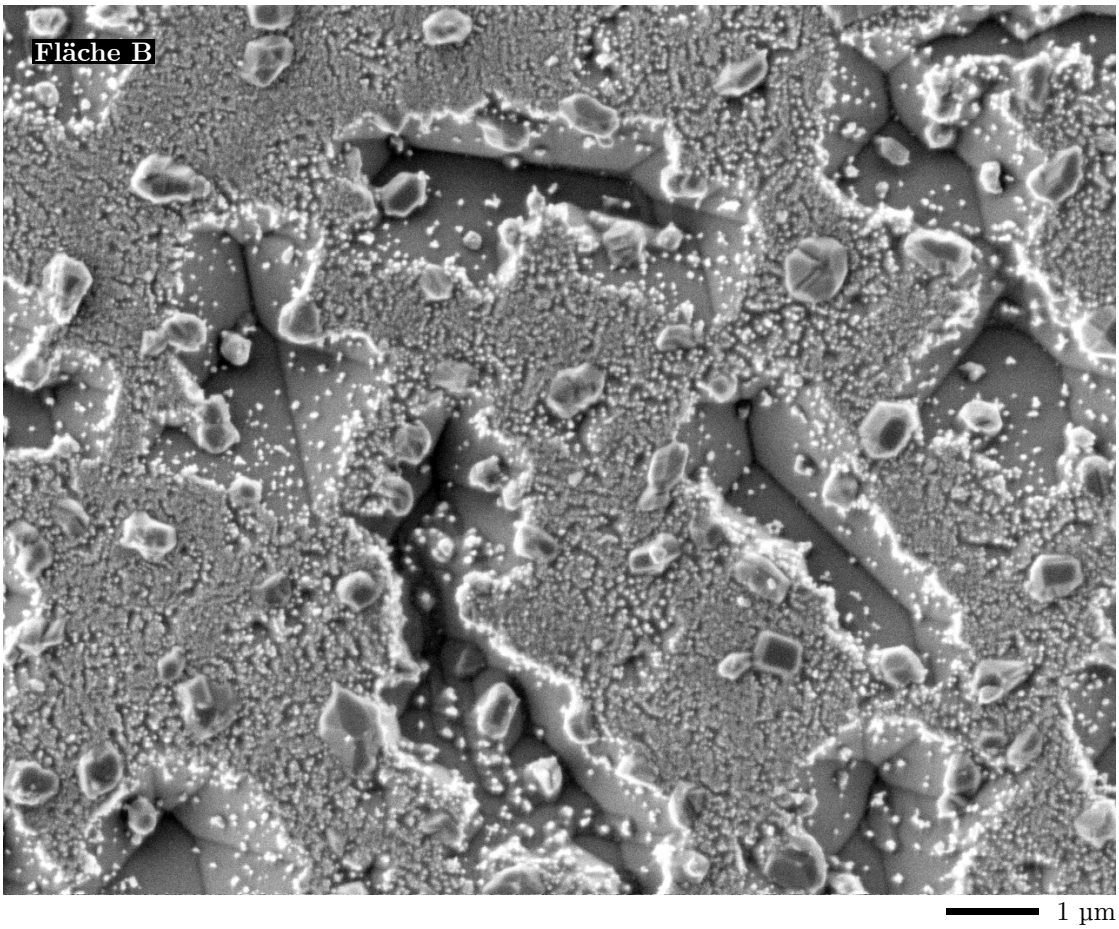


Abbildung 3.6: REM-Aufnahme (JEOL JSM-6700F). Fläche **B** bei höherer Auflösung.

In den folgenden Abschnitten werden REM-Aufnahmen von Cantilevern und deren Rückstände auf MgO-Substraten betrachtet. Hierbei wird sich unter anderem zeigen, dass die Bildung von Löchern auf der dem Substrat abgewandten Seite erfolgt und sehr wahrscheinlich nicht mit der chemischen Reaktion an der Probe-Substrat-Grenzfläche zusammenhängt.

3.1.2 Temperproben

Die Abbildungen 3.7 bis 3.16 zeigen REM-Aufnahmen einer Temperprobe (Versuchsbeschreibung siehe Abschnitt 2.4.1). Bei diesem Experiment hat die Cantilever-Probe eine Ätzgrube im Substrat hinterlassen, aus welcher sie wahrscheinlich im Zuge des Abkühlvorgangs gegen Ende des Experiments herausgeplatzt ist. In der Nähe der Reaktionszone wurden Cantilever-Bruchstücke gefunden. Durch vorsichtiges Umdrehen konnten die Bruchstücke von jeweils beiden Seiten untersucht werden.

Die Aufnahmen 3.7 bis 3.10 beschäftigen sich mit der Ätzgrube, die Cantilever-Bruchstücke werden durch die Abbildungen 3.11 bis 3.14 charakterisiert. Außerdem wurde mithilfe eines Focused Ion Beam (FIB) ein Querschnitt der Reaktionszone am Gerät ZEISS AURIGA angefertigt (Abbildungen 3.15 bis 3.16).

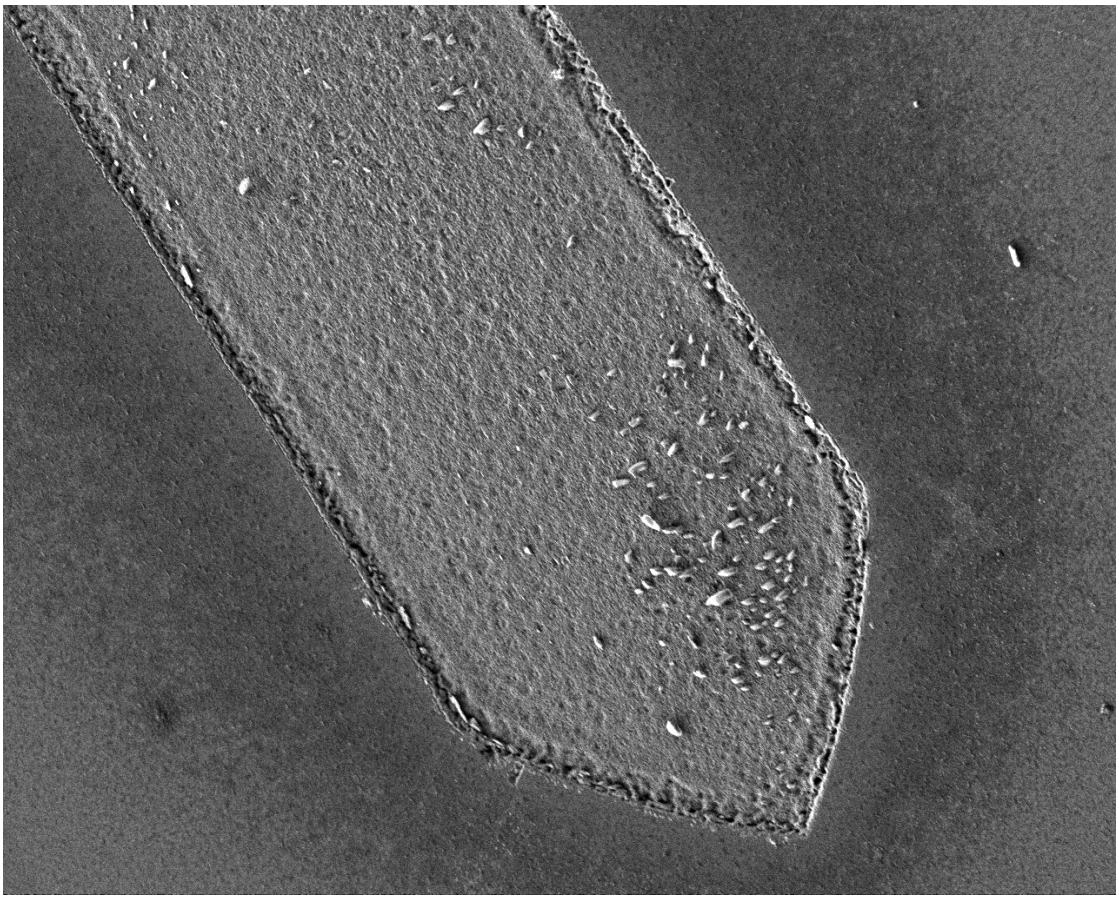


Abbildung 3.7: REM-Aufnahme (JEOL JSM-6700F). Freigelegte Ätzgrube im Bereich der Auflagefläche des Cantilevers während des Temperns. Die fein strukturierte Oberfläche der Reaktionszone und die hell erscheinenden Partikel werden in den Folgebildern näher betrachtet. Die Abbildung wurde bereits in ähnlicher Form veröffentlicht [28].

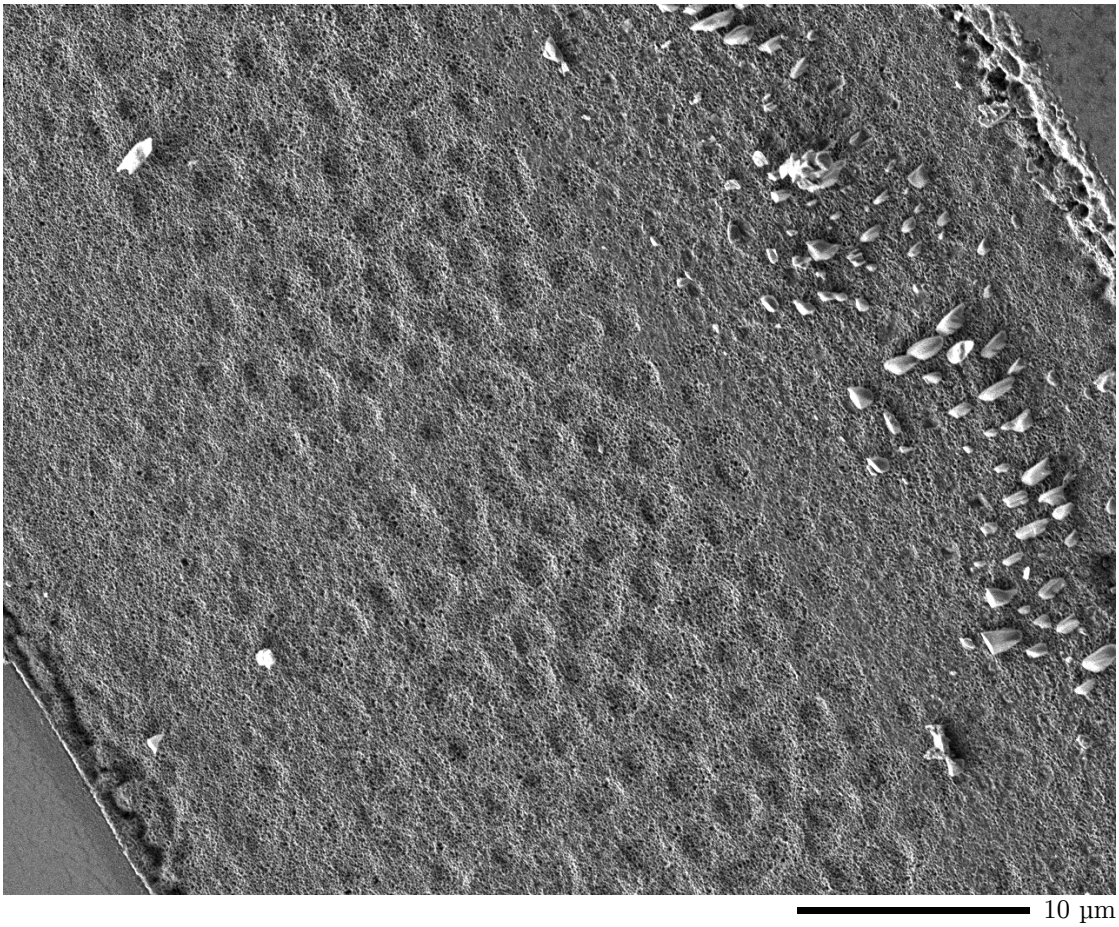


Abbildung 3.8: REM-Aufnahme (JEOL JSM-6700F). Ausschnitt aus dem mittleren Bereich der Ätzgrube. Im rechten Teil des Bildes befinden sich körnerartige Rückstände. Auffällig ist, dass die länglichen Körnchen größtenteils in die gleiche Richtung ausgerichtet zu sein scheinen. Im Mittelteil des Bildes zeigt sich die fein strukturierte Oberfläche der Reaktionszone. Im Zentrum der Ätzgrunbe ist außerdem ein Muster kleiner, rundlicher Strukturelemente auf der Oberfläche zu erkennen.

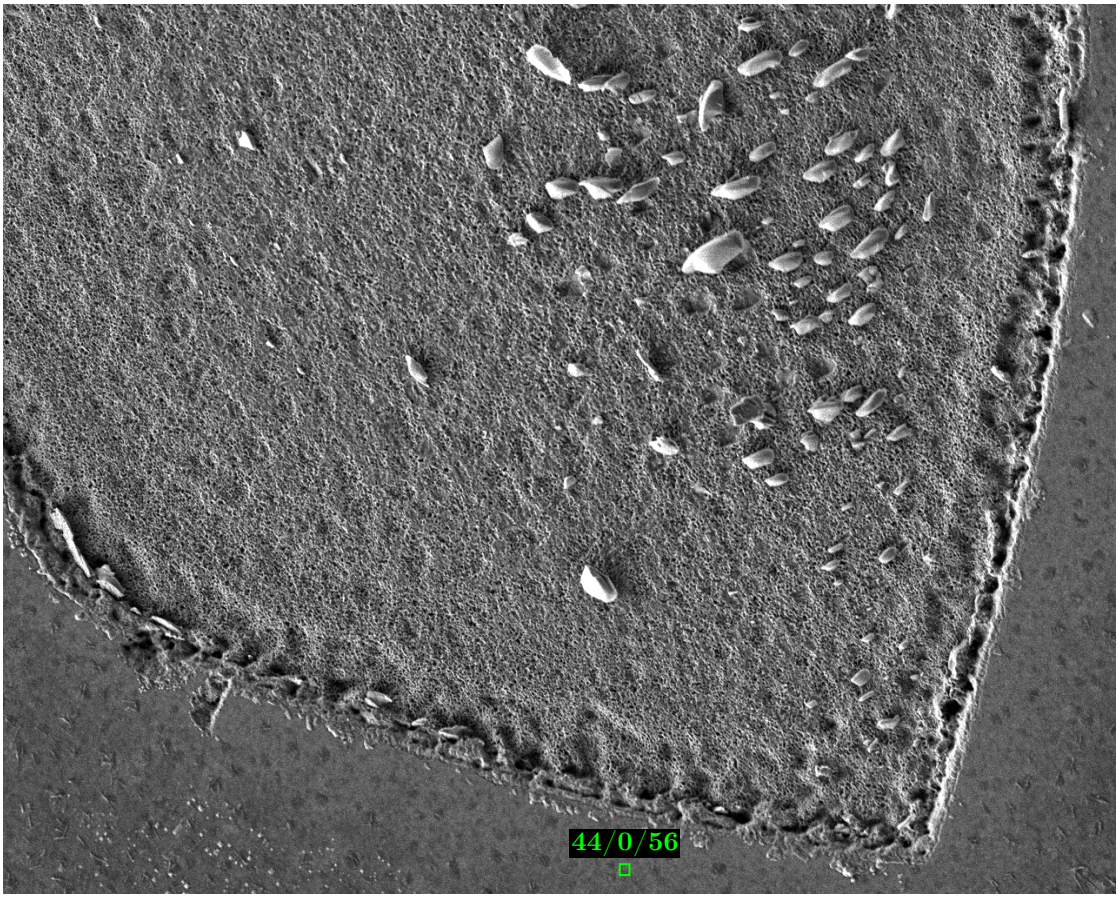


Abbildung 3.9: REM-Aufnahme (JEOL JSM-6700F). Vorderer Bereich der Ätzgrube. Es zeigt sich ein relativ scharfer Übergang zwischen der Reaktionszone und der Oberfläche des MgO-Substrats. Eine EDX-Analyse (grünes Rechteck) des Substratbereichs in unmittelbarer Nähe des Auflagebereichs des Cantilevers lieferte die Oberflächenzusammensetzung 44/0/56 (Mg/Si/O Atomprozent). Es handelt sich also um Magnesiumoxid, vermutlich mit leichtem Sauerstoffüberschuss an der Oberfläche.

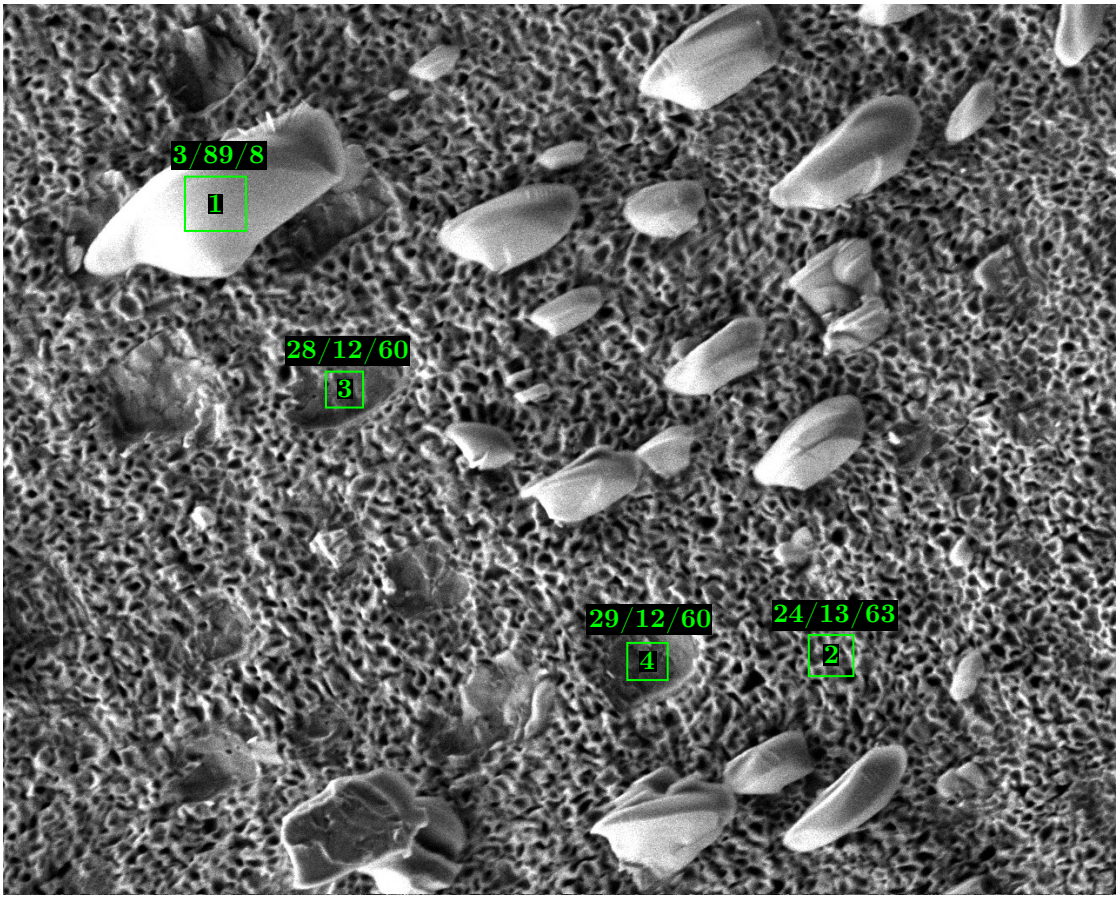


Abbildung 3.10: REM-Aufnahme (JEOL JSM-6700F). Oberfläche der Ätzgrube und Silicium-Körnchen. Grüne Rechtecke: EDX-Messungen, beschriftet mit der Zusammensetzung Mg/Si/O in Atomprozent. Für diese und alle weiteren derartigen Angaben gilt, dass jeweils ganzzahlig gerundet wurde, sodass die Summe der drei Werte knapp von 100 abweichen kann. **1**: Die gemessene Zusammensetzung des Körnchens wird als Silicium mit leichtem Sauerstoffanteil (wahrscheinlich an der Oberfläche) gedeutet. Der geringe Magnesiumanteil ist vermutlich auf Störsignale aus der Umgebung zurückzuführen. In den Folgebildern werden sich mit den Körnchen korrespondierende Kavitäten in der Auflageseite des Si-Cantilevers zeigen. **2**: Die Zusammensetzung der fein struktuierten Oberfläche entspricht formal der Verbindung $Mg_{1,8}Si_1O_{4,8}$ und deutet somit auf Magnesiumsilikat (Mg_2SiO_4) hin. **3-4**: In diesen Bereichen erscheint die Oberfläche glatter als in der Umgebung. Vielleicht wurde die Oberfläche an diesen Stellen beim Abplatzen des Cantilevers beschädigt. Die EDX-Messungen zeigen einen leicht höheren Magnesiumanteil im Vergleich zu Stelle **2**. Erfahrungsgemäß trägt bei dieser Auflösung auch die unmittelbare Umgebung des ausgewählten Messbereichs zum Signal bei, sodass die Angabe einer formalen chemischen Zusammensetzung für die Bereiche **3** und **4** nicht sinnvoll ist. Die Abbildung wurde bereits in ähnlicher Form veröffentlicht [28].

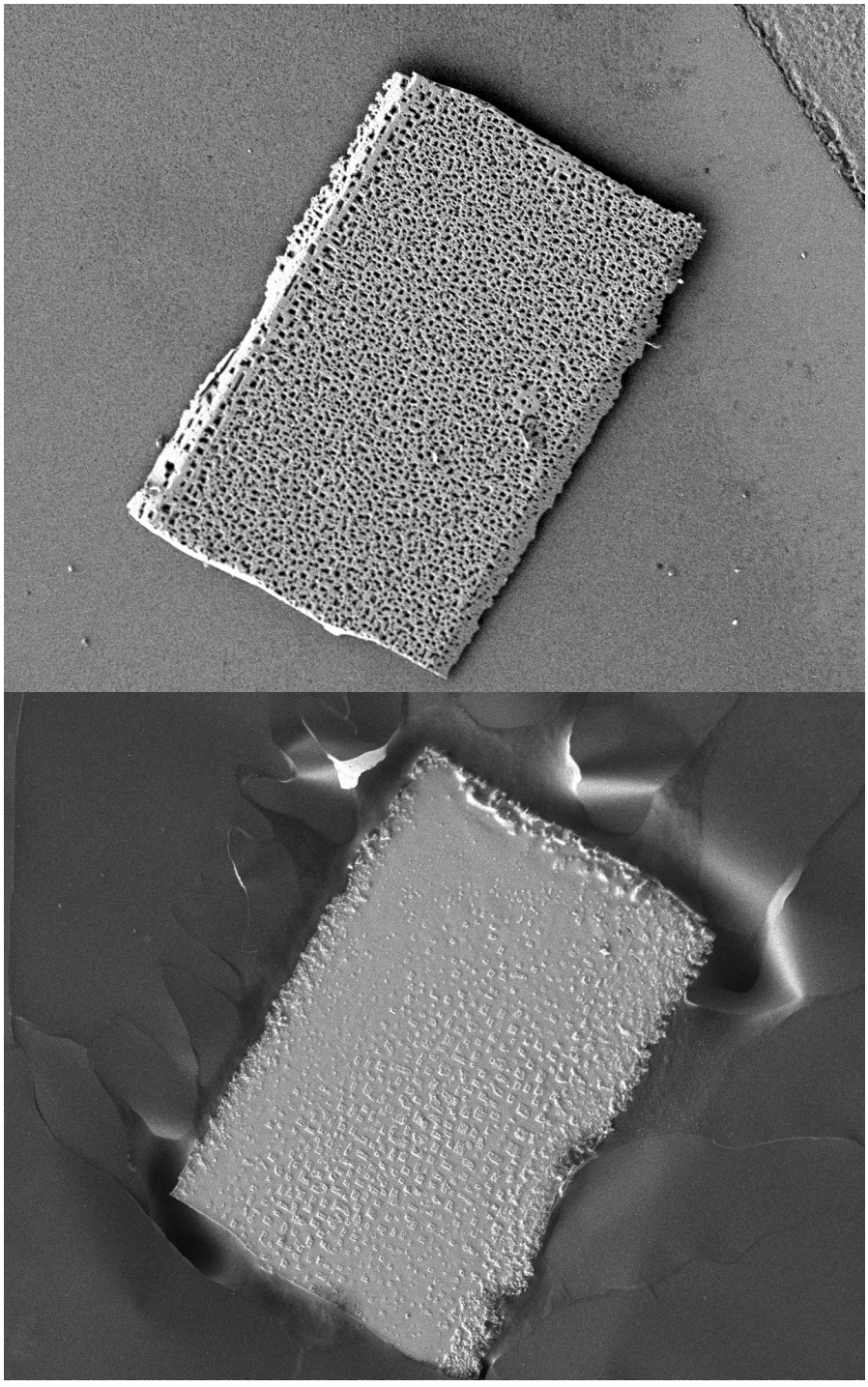


Abbildung 3.11: REM-Aufnahmen (JEOL JSM-6700F). Cantilever-Bruchstück. Oben: Seite, die beim Tempern nicht im Kontakt mit dem Substrat stand (Vakuumseite). Unten: Seite desselben Bruchstücks, welche beim Tempern im Kontakt mit dem Substrat stand (Reaktionsseite). Nach dem Umdrehen wurde das Bruchstück auf ein Klebepad aufgebracht, dessen Oberfläche im unteren Bild zu sehen ist.

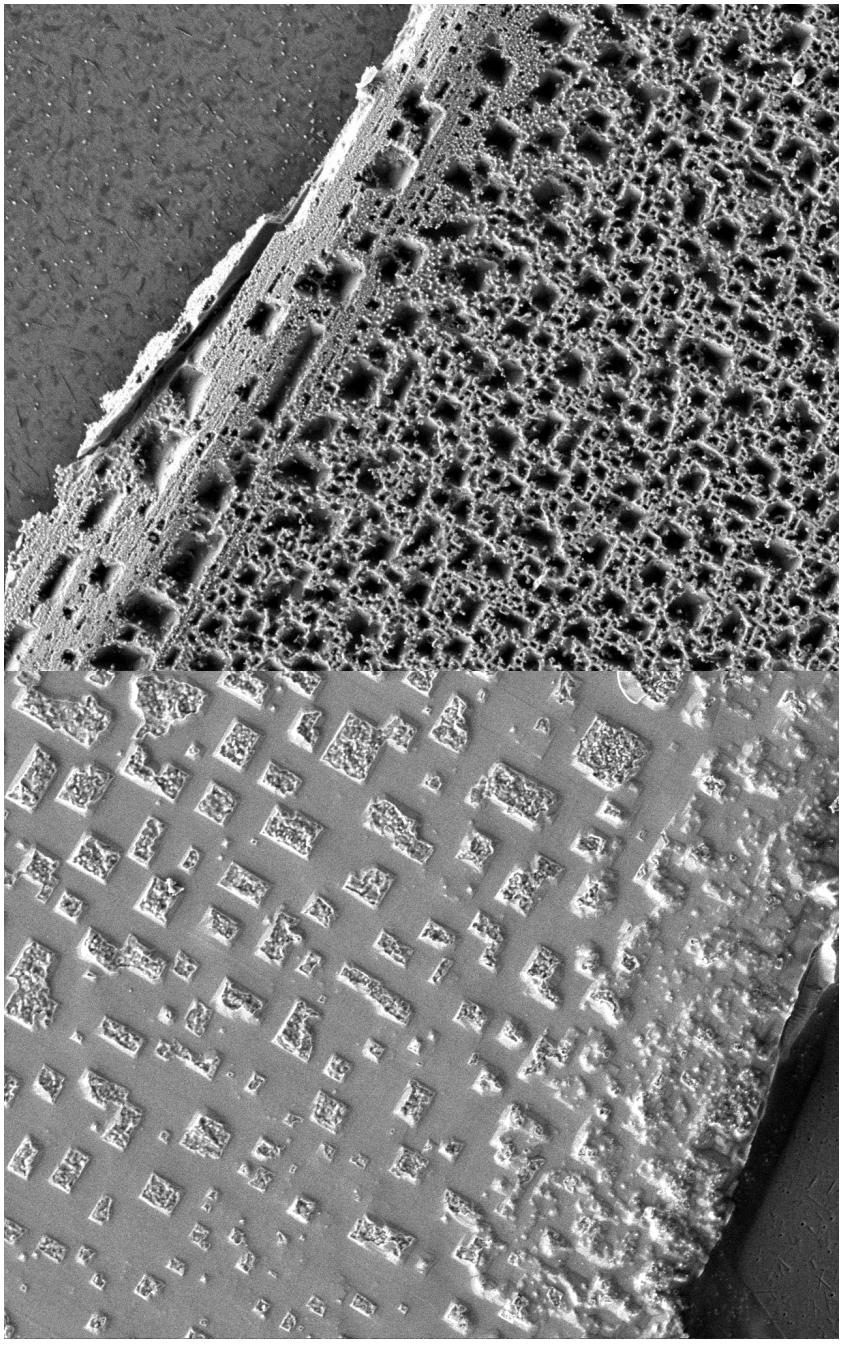


Abbildung 3.12: REM-Aufnahmen (JEOL JSM-6700F). Cantilever-Bruchstück. Oben: Vakuumseite. Die Oberfläche ist rau und von Löchern durchzogen. Unten: Reaktionsseite. Die Oberfläche zeigt rechteckige Erhebungen und ist in den Zwischenräumen glatt. Möglicherweise standen die Erhebungen in Kontakt zum Substrat und erklären Oberflächenmuster wie in Bild 3.8. Auf beiden Seiten zeigten EDX-Messungen nur Silicium mit Spuren von Sauerstoff.

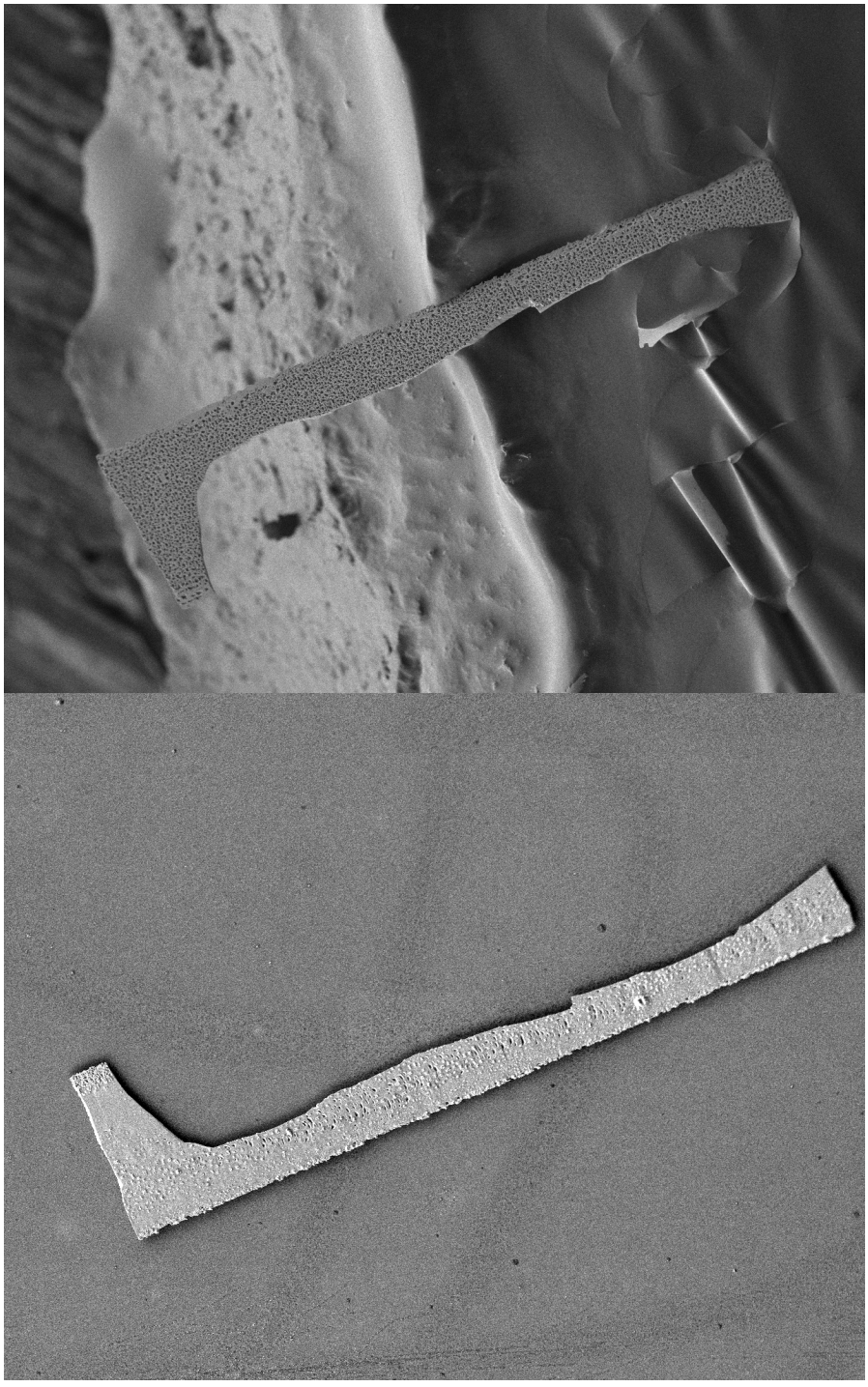


Abbildung 3.13: REM-Aufnahmen (JEOL JSM-6700F). Cantilever-Bruchstück. Oben: Vakuumseite, auf Klebepad (linke Hälfte steht über). Unten: Reaktionsseite auf MgO-Substrat.

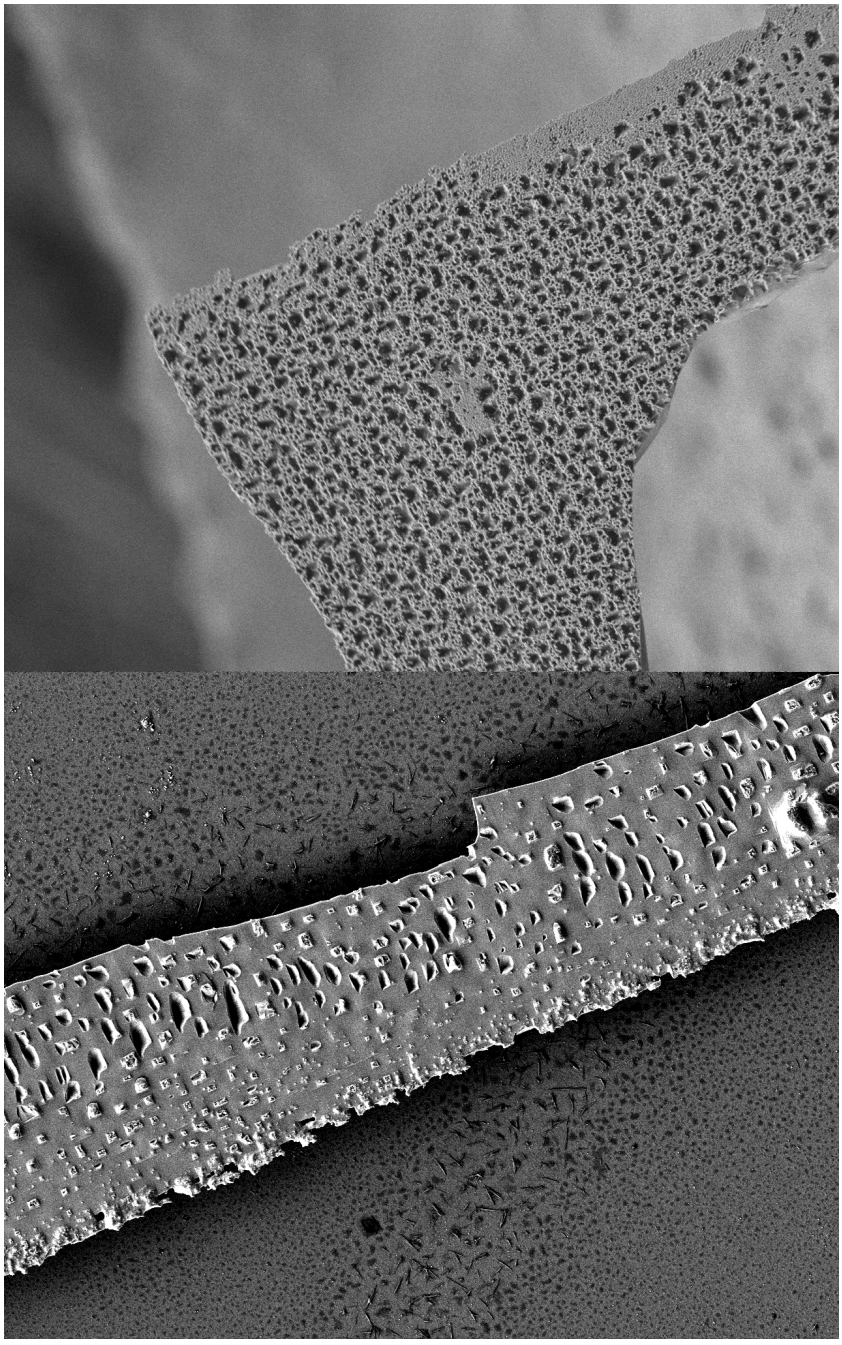


Abbildung 3.14: REM-Aufnahmen (JEOL JSM-6700F). Cantilever-Bruchstück. Oben: Vakuumseite. Unten: Reaktionsseite. Die Oberfläche gleicht der Reaktionsseite in Abbildung 3.12. Außerdem zeigen sich längliche Vertiefungen. An diesen Stellen scheint Material abgeplatzt zu sein. Sie bilden die Gegenstücke der länglichen Siliciumkörnchen in der Ätzgrube (vergleiche Abbildung 3.8).

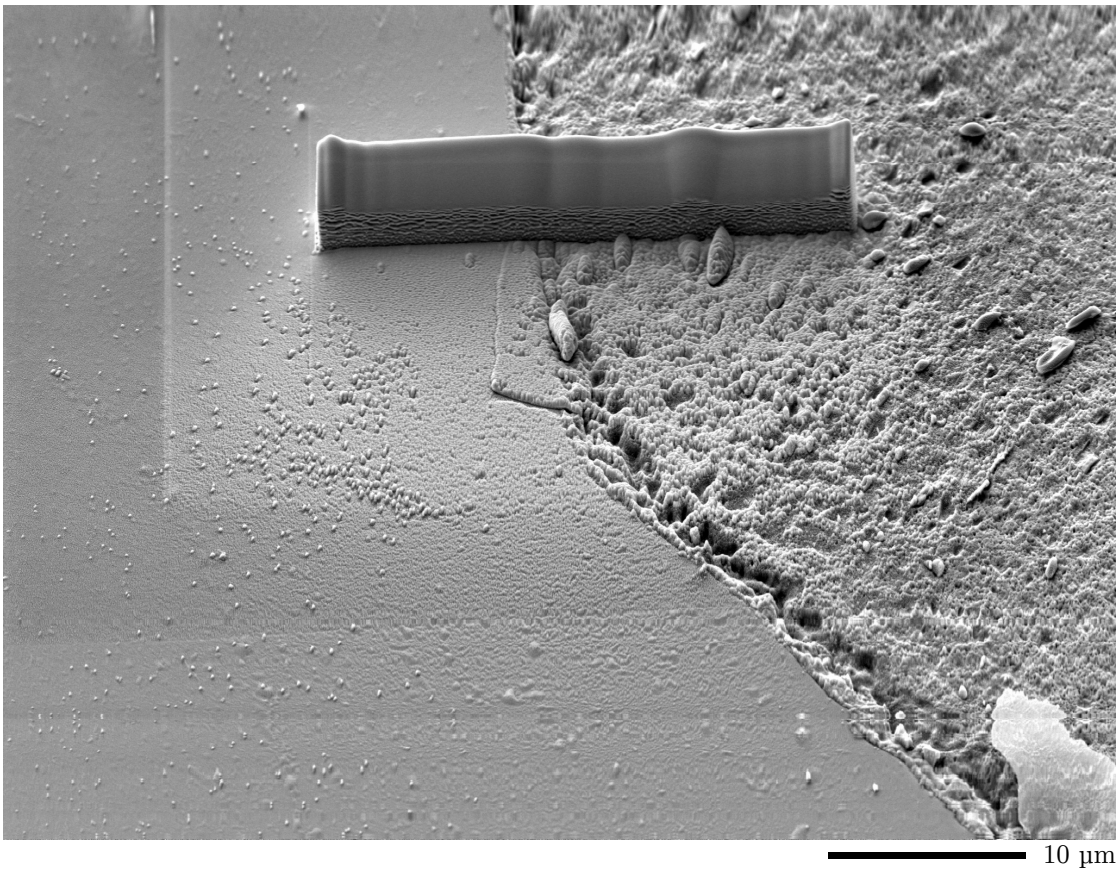


Abbildung 3.15: REM-Aufnahme (ZEISS AURIGA). Als Opfermaterial auf die Substratoberfläche aufgebrachter Platinbalken. In die Probenkammer wird ein gasförmiger Platinkomplex eingesprüht, welcher dann an der zu beschichtenden Stelle durch Beschuss mit Gallium-Ionen aus dem Focused Ion Beam aufgebrochen wird. Das elementare Platin setzt sich auf der Probe ab.

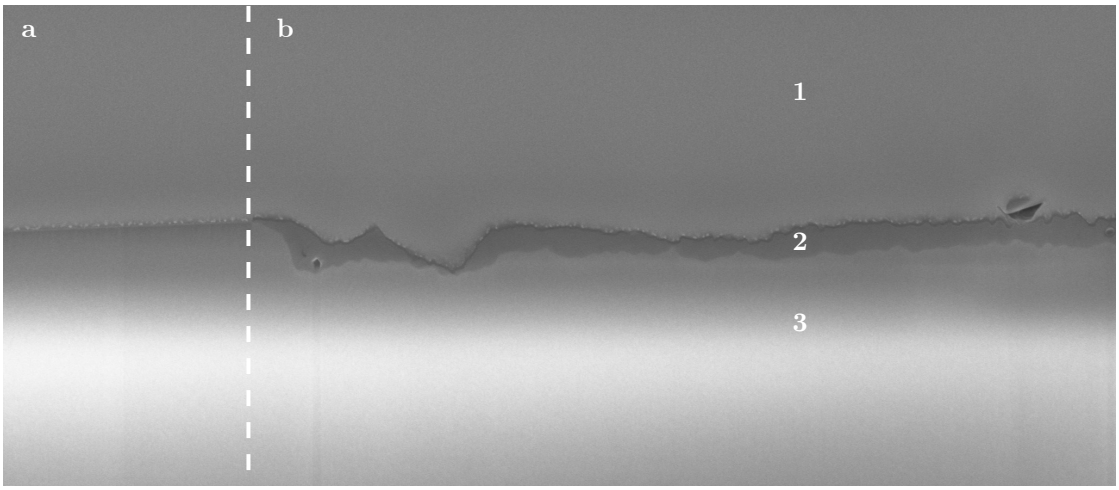


Abbildung 3.16: REM-Aufnahme (ZEISS AURIGA). Mit Hilfe des Focused Ion Beam freigelegter Querschnitt der Substratoberfläche, parallel zum Platinbalken aus Abbildung 3.15. **a** Bereich außerhalb der Ätzgrube. **b** Bereich innerhalb der Ätzgrube. **1** Platin-Opferschicht. **2** Schicht mit Reaktionsprodukten in der Ätzgrube, hier ungefähr 250 nm dick. Mit Hilfe des Simulationsprogramms CASINO („monte CArlo SImulation of electroN trajectory in sOlids“) wurde mit der bei den EDX-Messungen verwendeten Beschleunigungsspannung von 5 kV eine etwa 250 nm tiefe Anregungsbirne in Mg_2SiO_4-Bulkmaterial berechnet [35]. Die Messung an Stelle **2** in Abbildung 3.10 weißt demnach deutlich auf Mg_2SiO_4 als Reaktionsprodukt an der Oberfläche der Ätzgrube hin. **3** MgO-Substrat. Der Isolator lädt sich elektrostatisch auf und erscheint daher heller. Die Abbildung wurde bereits in ähnlicher Form veröffentlicht [28].

3.1.3 Hauptexperiment

Die Abbildungen 3.17 bis 3.21 zeigen REM-Aufnahmen eines Cantilevers auf einem MgO-Substrat nach der Durchführung eines Hauptexperiments gemäß Abschnitt 2.3. Die Auswertung der Videodaten in Kapitel 5 erfolgte auf Basis dieses Experiments.

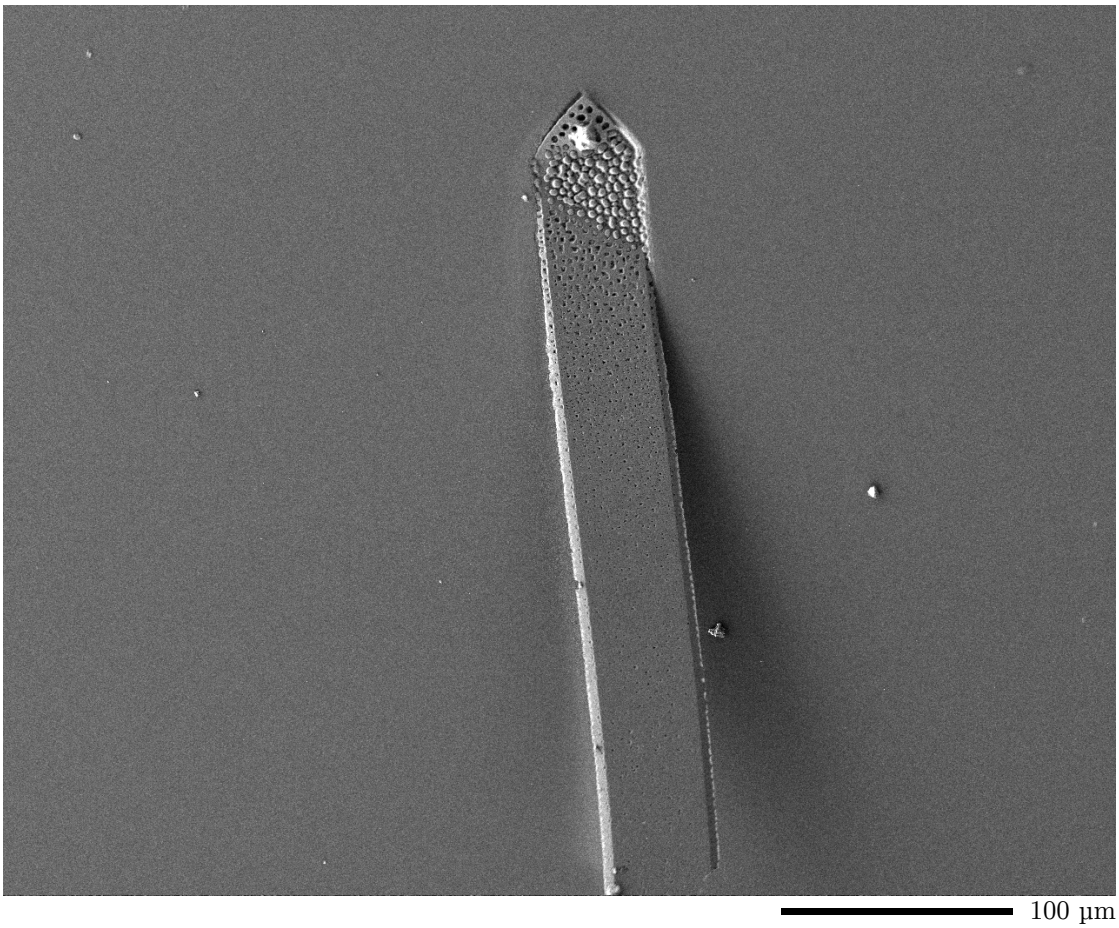


Abbildung 3.17: REM-Aufnahme (JEOL JSM-6700F). Cantilever auf MgO-Substrat. Der Kopfbereich ist auf dem Substrat angeschmolzen. Wahrscheinlich hat sich im Zuge dieses Prozesses der übrige Probenkörper leicht aufgestellt und den Kontakt zum Substrat verloren.

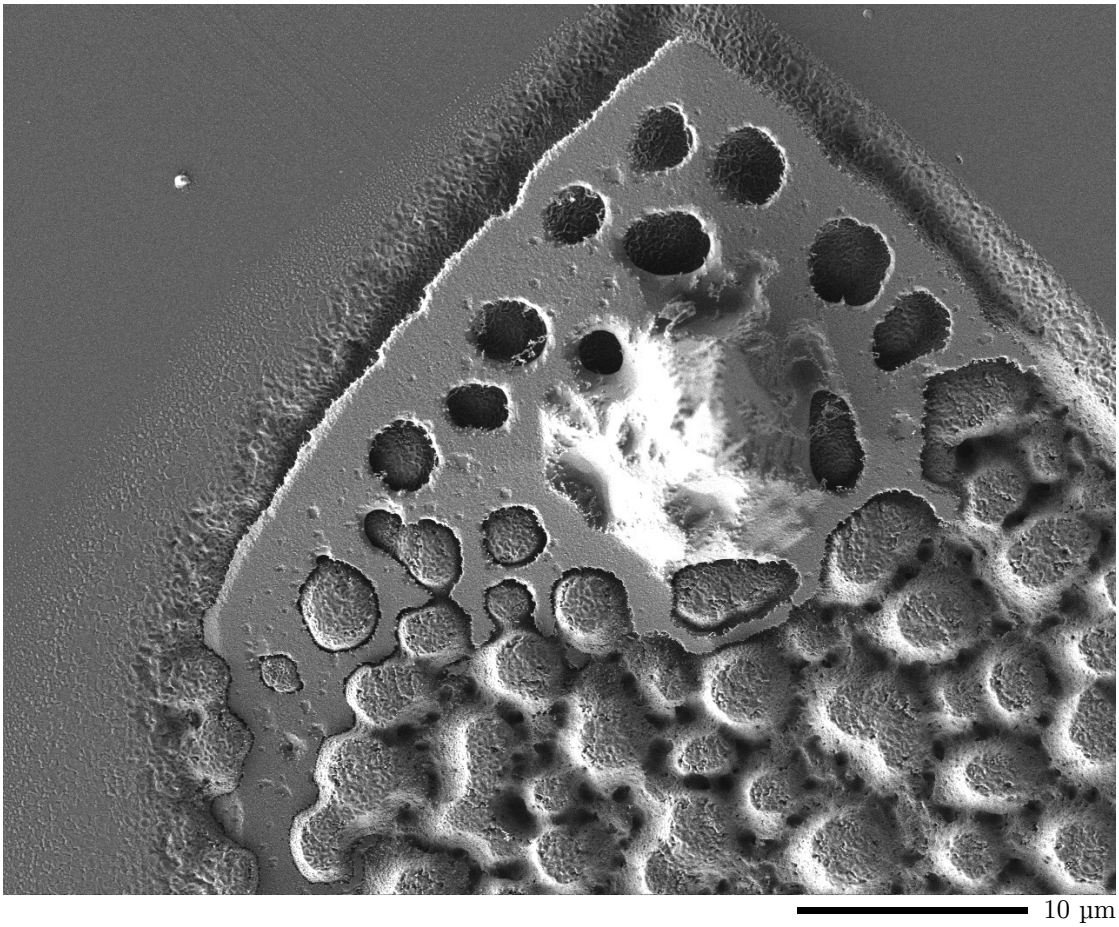


Abbildung 3.18: REM-Aufnahme (JEOL JSM-6700F). Kopfbereich des Cantilevers. Beim hellen Bereich im Zentrum des Bildes handelt es sich um die Messspitze (siehe Abbildung 2.4). Im Gebiet oberhalb der Messspitze ist die Probe von Löchern durchzogen und hat den Kontakt zum Substrat verloren. Im unteren Bereich sind noppenartige Erhebungen entstanden. Diese zeigen zusammen mit dem Randbereich der Probe eine fein strukturierte Oberfläche, die stark an die Oberfläche der Ätzgrube in den REM-Aufnahmen der Temperprobe erinnert. Links der Messspitze sind einzelne Noppen zu erkennen, die den Cantilever durchbrechen, und somit den Entstehungsprozess der Löcher zeigen.

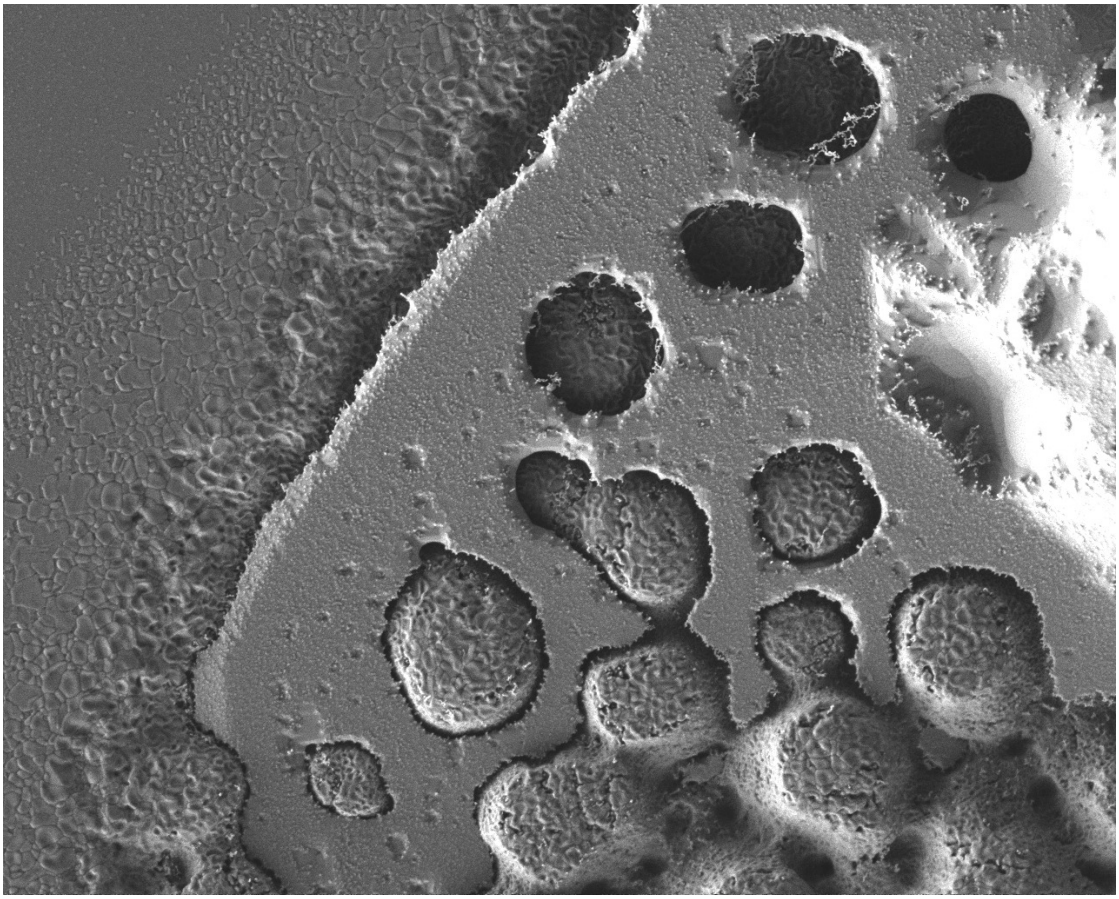


Abbildung 3.19: REM-Aufnahme (JEOL JSM-6700F). Bereich links der Messspitze bei hoher Auflösung.

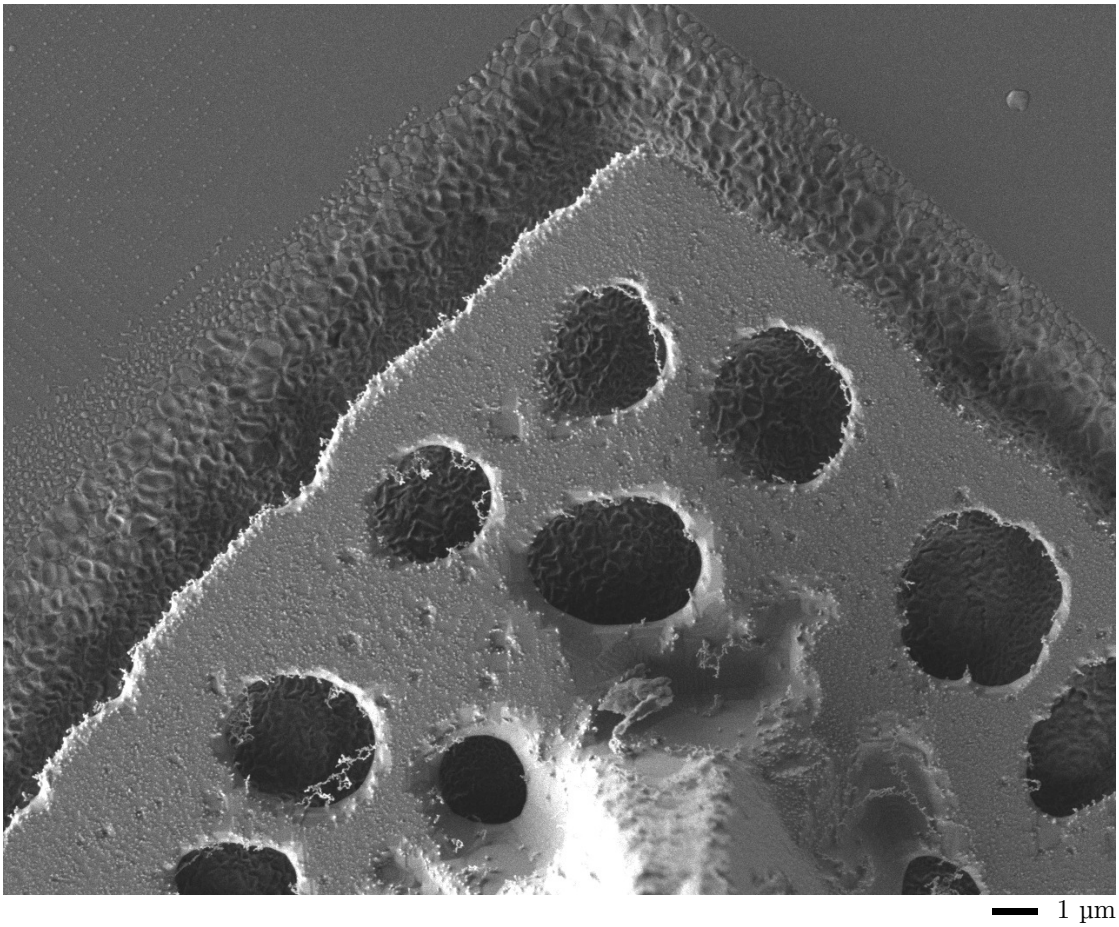


Abbildung 3.20: REM-Aufnahme (JEOL JSM-6700F). Bereich oberhalb der Messspitze bei hoher Auflösung. Im Hintergrund ist die Ätzgrube und deren strukturierte Oberfläche zu erkennen.

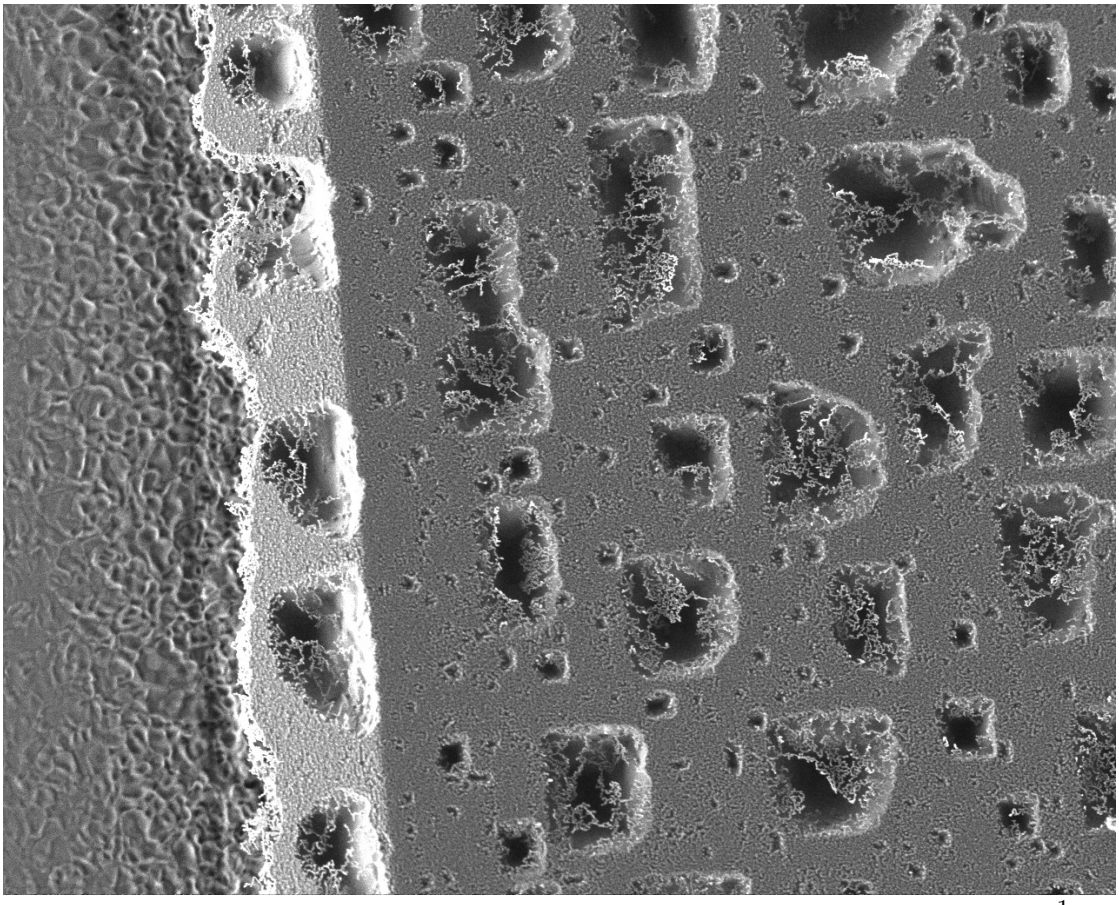


Abbildung 3.21: REM-Aufnahme (JEOL JSM-6700F). Körper des Cantilevers und Randbereich. Die Oberfläche der Probe zeigt rechteckige Kavitäten, die zum Teil von feinen, gewebeartigen Netzen bedeckt sind. Da ein Hauptexperiment nur etwa ein Fünftel der Zeit eines fünfstündigen Temperexperiments andauert, ist die Oberfläche noch nicht komplett mit Löchern durchsetzt (vergleiche Abbildung 3.12). Bei der Entstehung der Löcher handelt es sich wahrscheinlich um einen Abdampfprozess, der nicht mit der chemischen Reaktion an der Cantilever-Substrat-Grenzfläche zusammenhängt (zur Begründung siehe nächster Abschnitt).

3.1.4 PBN-Substrate

Um die Effekte auf der Vakuumseite des Cantilevers zu klären, wurden Vergleichsexperimente zum Hauptexperiment durchgeführt, bei denen die MgO-Substrate durch chemisch inerte PBN-Substrate ersetzt wurden. Die typischen rechteckigen Kavitäten aus den vorangegangenen Abbildungen zeigen sich auch auf diesen Substraten. Bei der Entstehung handelt es sich demnach wahrscheinlich um einen Abdampfungsprozess.

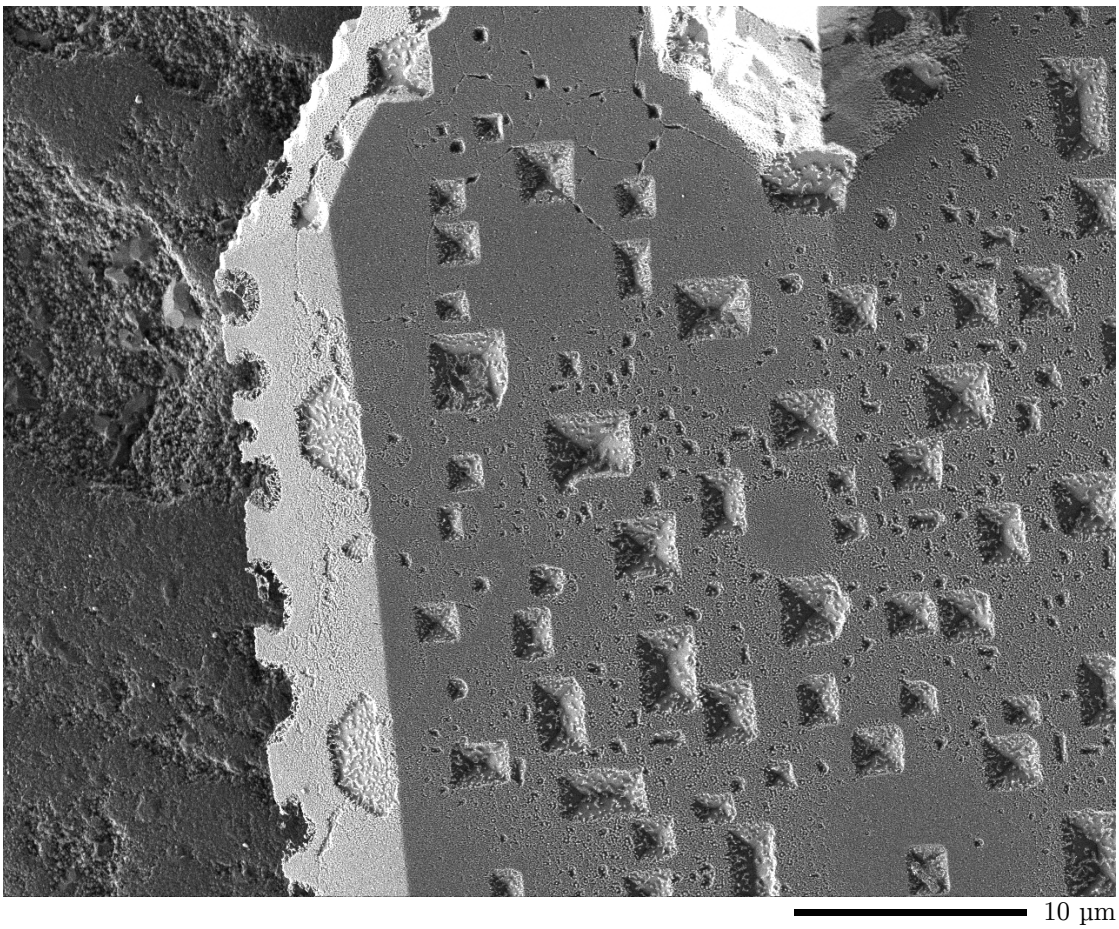


Abbildung 3.22: REM-Aufnahme (JEOL JSM-6700F). Cantilever auf PBN-Substrat.

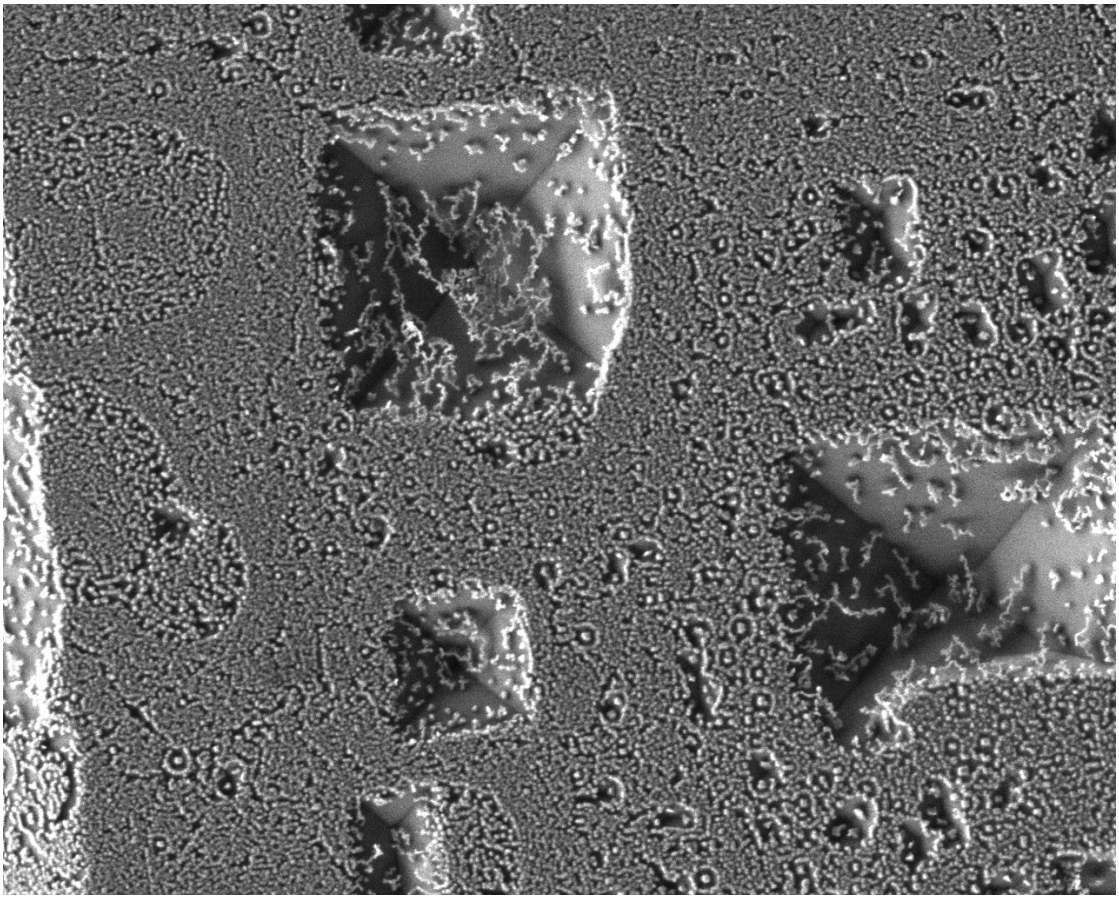


Abbildung 3.23: REM-Aufnahme (JEOL JSM-6700F). Cantilever auf PBN-Substrat. In dieser Abbildung ist gut die leicht körnige Oberflächenstruktur zu erkennen. Wahrscheinlich handelt es sich bei den „Netzen“ in den Vertiefungen um zusammenhängende Überreste dieser Körnchen.

3.1.5 Rückstand

Die nächste Bilderserie zeigt den Rückstand eines Cantilevers auf einem MgO-Substrat nach einem Experiment, bei dem der Schmelzvorgang nicht abgebrochen wurde. Von dieser Probe wurde außerdem eine TEM-Lamelle angefertigt, deren Entstehung durch REM-Aufnahmen dokumentiert ist.

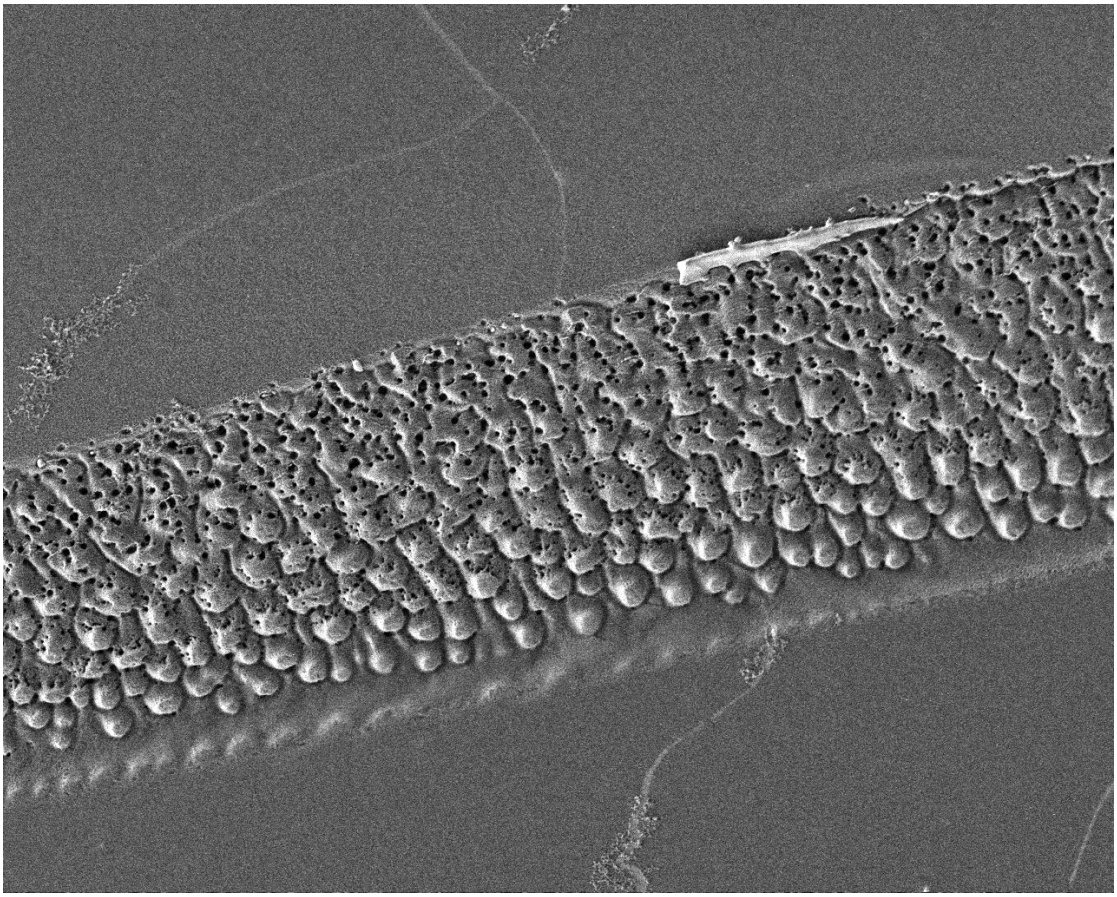


Abbildung 3.24: REM-Aufnahme (JEOL JSM-6700F). Rückstand der abreagierten Probe auf dem MgO-Substrat. Es sind die gleichen noppenartigen Erhebungen wie in Abbildung 3.18 zu erkennen. Am oberen Rand der Reaktionszone befindet sich ein länglicher Rückstand von Silicium (EDX siehe nächste Abbildung).

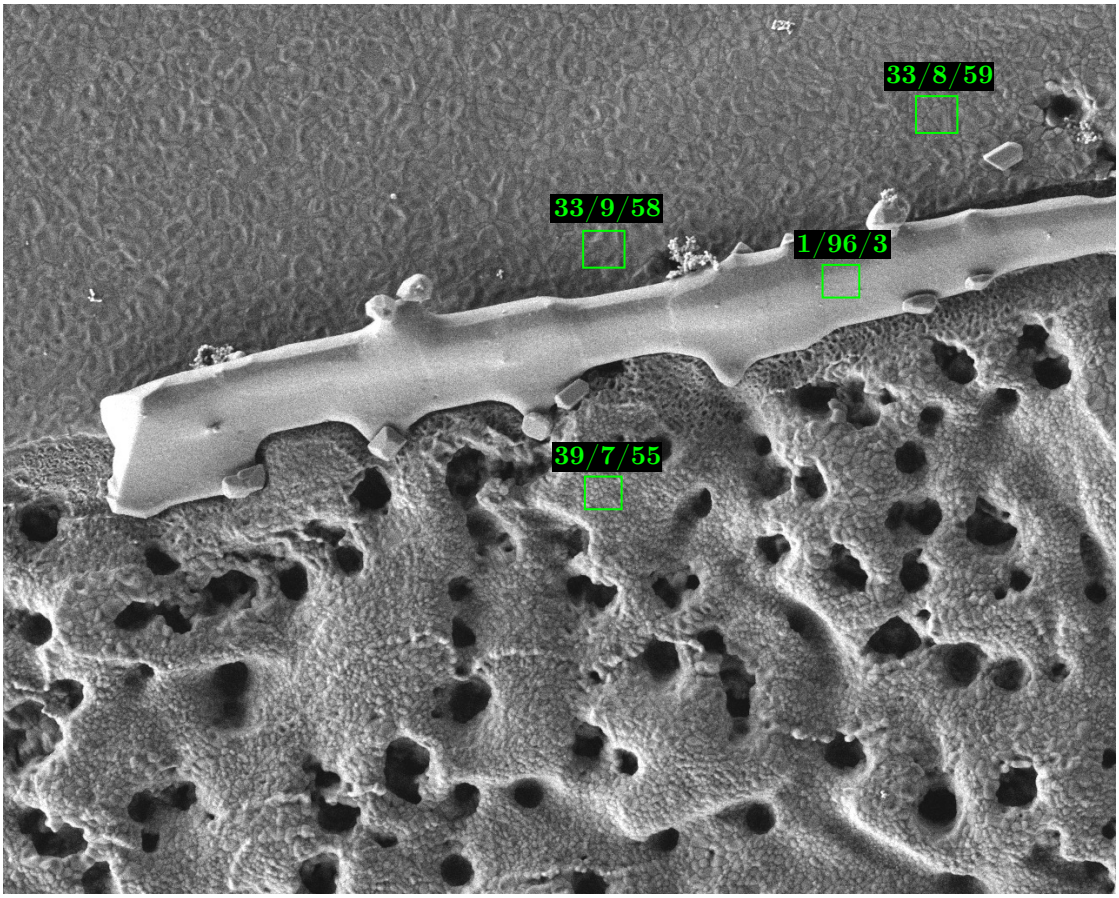


Abbildung 3.25: REM-Aufnahme (JEOL JSM-6700F). Reaktionszone, Randbereich und Rückstand. Grüne Rechtecke: EDX-Messungen, beschriftet mit der Zusammensetzung Mg/Si/O in Atomprozent. Der Rückstand und das Substrat in unmittelbarer Nähe der Reaktionszone besitzen ähnliche EDX-Signale. Eine Zusammensetzung von Mg/Si/O von 39/7/55 könnte sich formal ungefähr durch vier Teile MgO und einen Teil Mg_2SiO_4 ergeben. In den nachfolgenden Betrachtungen wird sich zeigen, dass die Eindringtiefe der Elektronen bei den EDX-Messungen für dieses Experiment wahrscheinlich über die Reaktionsschicht an der Oberfläche hinaus reichte. Der längliche Balken in der oberen Bildhälfte ist ein Silicium-Rückstand.

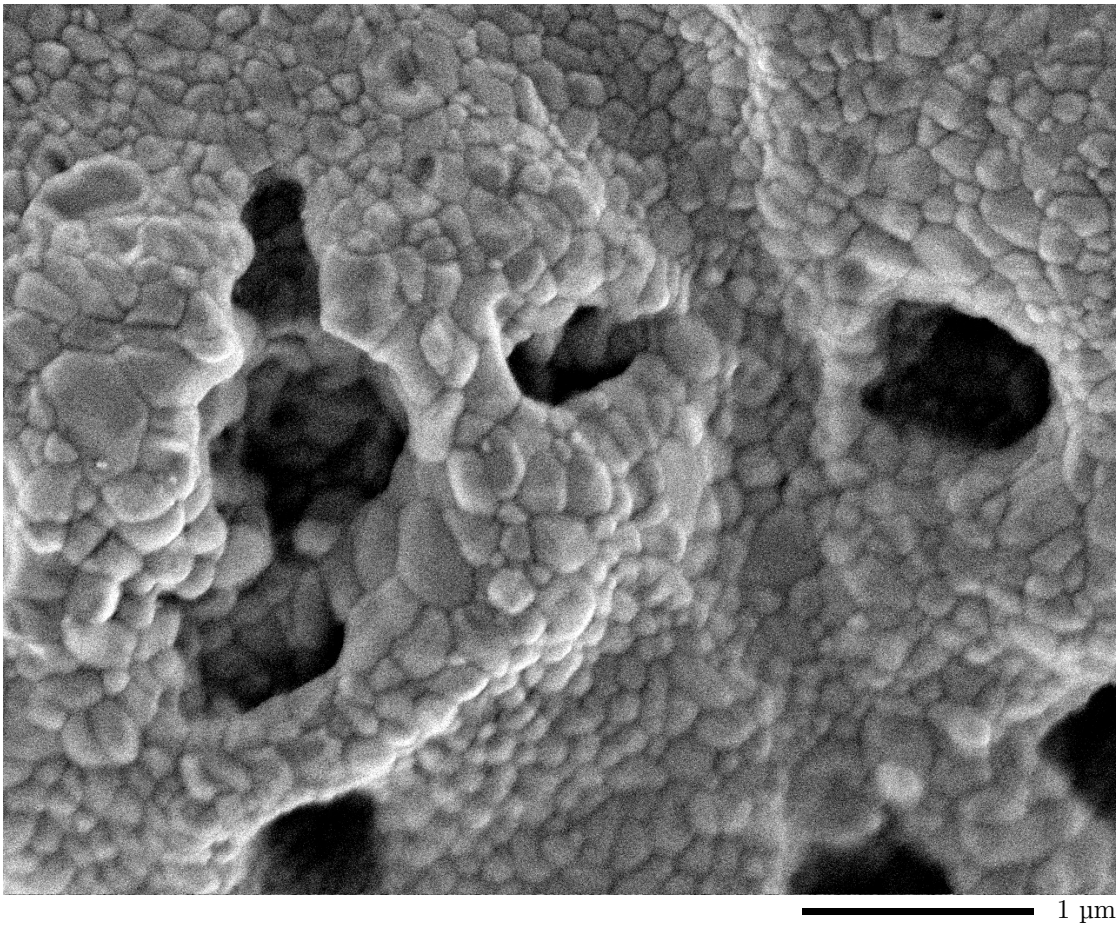


Abbildung 3.26: REM-Aufnahme (JEOL JSM-6700F). Bei hoher Auflösung sind Korngrenzen an der Oberfläche der Reaktionszone zu erkennen.

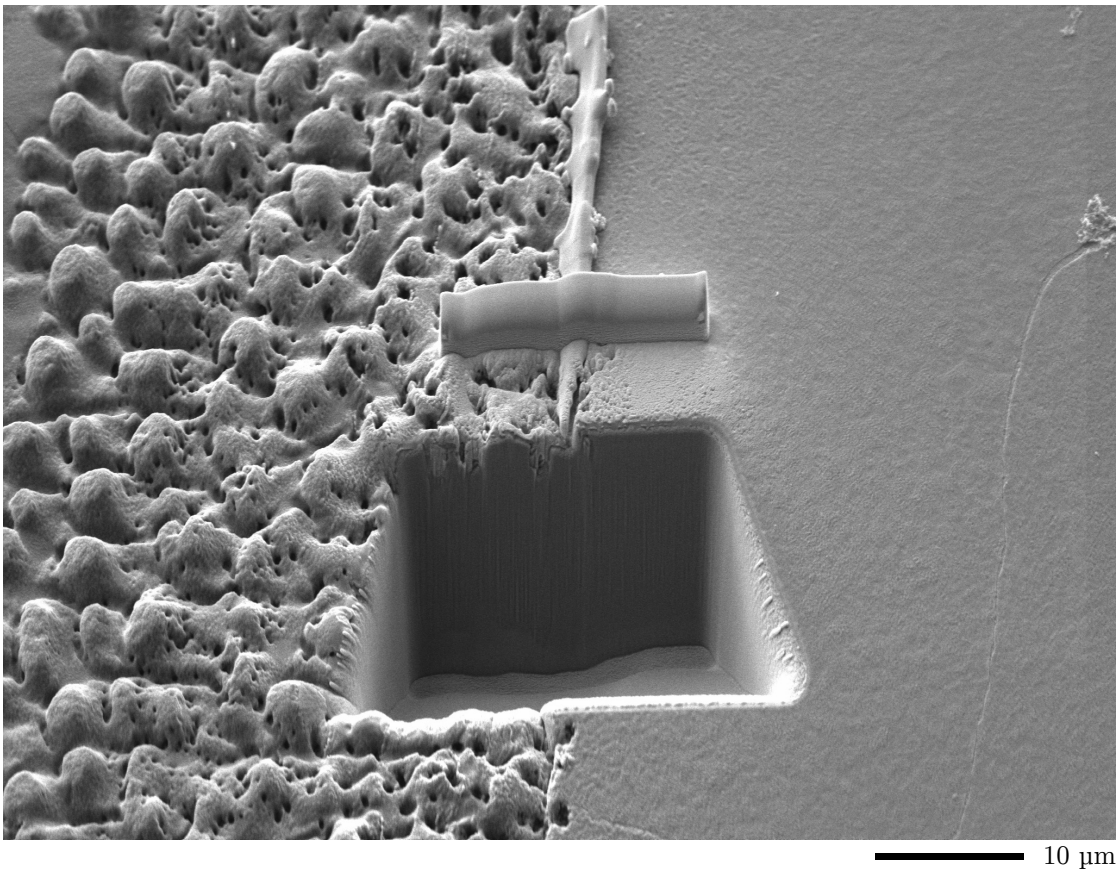


Abbildung 3.27: REM-Aufnahme (JEOL JSM-6700F). Herstellung einer TEM-Lamelle: Herausschneiden eines Keils mit dem Focused Ion Beam. Der Keil wird bis in die Platin-Opferschicht (länglicher Balken im rechten Winkel zum Si-Rückstand) hinein geschnitten.

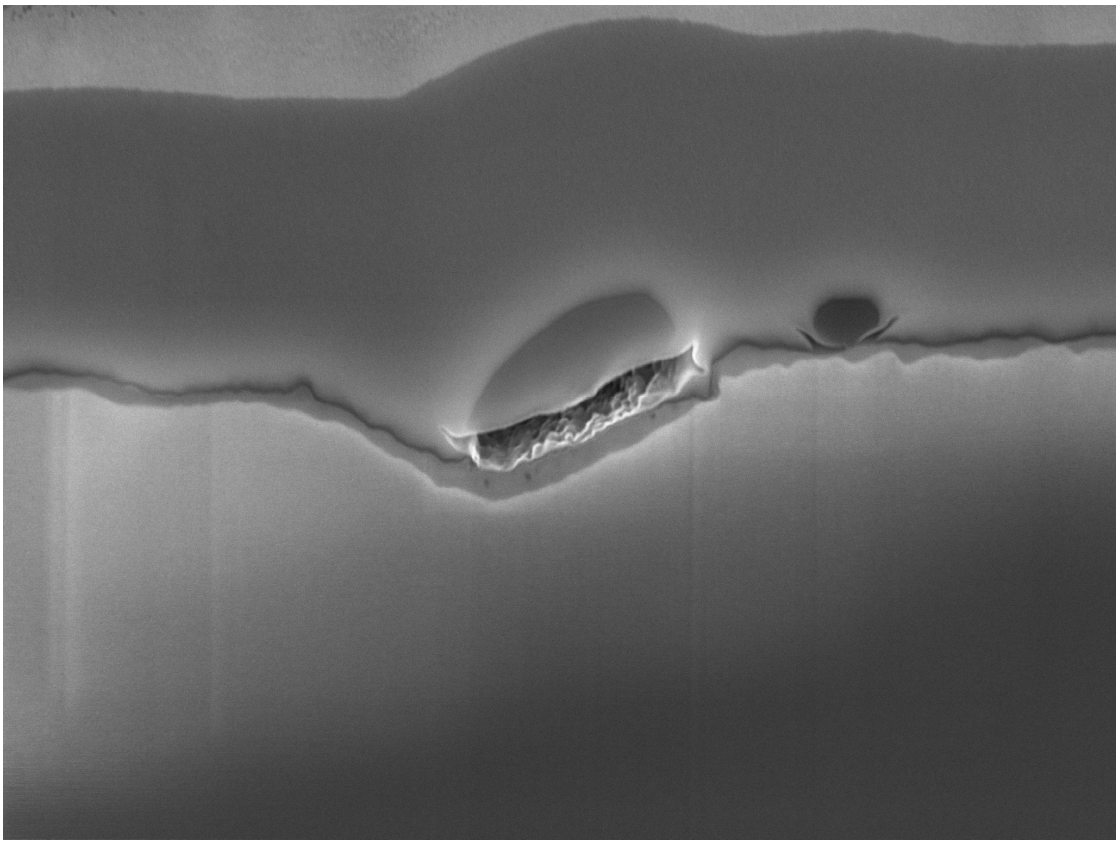


Abbildung 3.28: REM-Aufnahme (JEOL JSM-6700F). Blick auf den Querschnitt, dessen Anfertigung im vorherigen Bild gezeigt wurde. In der Mitte befindet sich der Si-Rückstand und direkt darunter ein Hohlraum. Links davon befindet sich die Reaktionszone. Rechts davon befindet sich der Bereich des MgO-Substrats direkt neben der Reaktionszone und der Querschnitt eines kleinen Silicumkörnchens. Beim dünnen, dunklen Streifen der zwischen dem linken und dem rechten Bildrand verläuft, handelt es sich um die Reaktionsschicht, beziehungsweise den Rückstand an Reaktionsprodukten. Die Dicke des Streifens beträgt etwa 100–200 nm. Damit setzen sich die in Abbildung 3.25 diskutierten EDX-Signale der Oberfläche wahrscheinlich aus Reaktionsprodukt und darunter liegendem MgO zusammen. (Mit dem Programm CASINO wurde eine etwa 250 nm tiefe Anregungsbirne für die verwendeten Geräteparameter für Mg_2SiO_4 simuliert [35].)

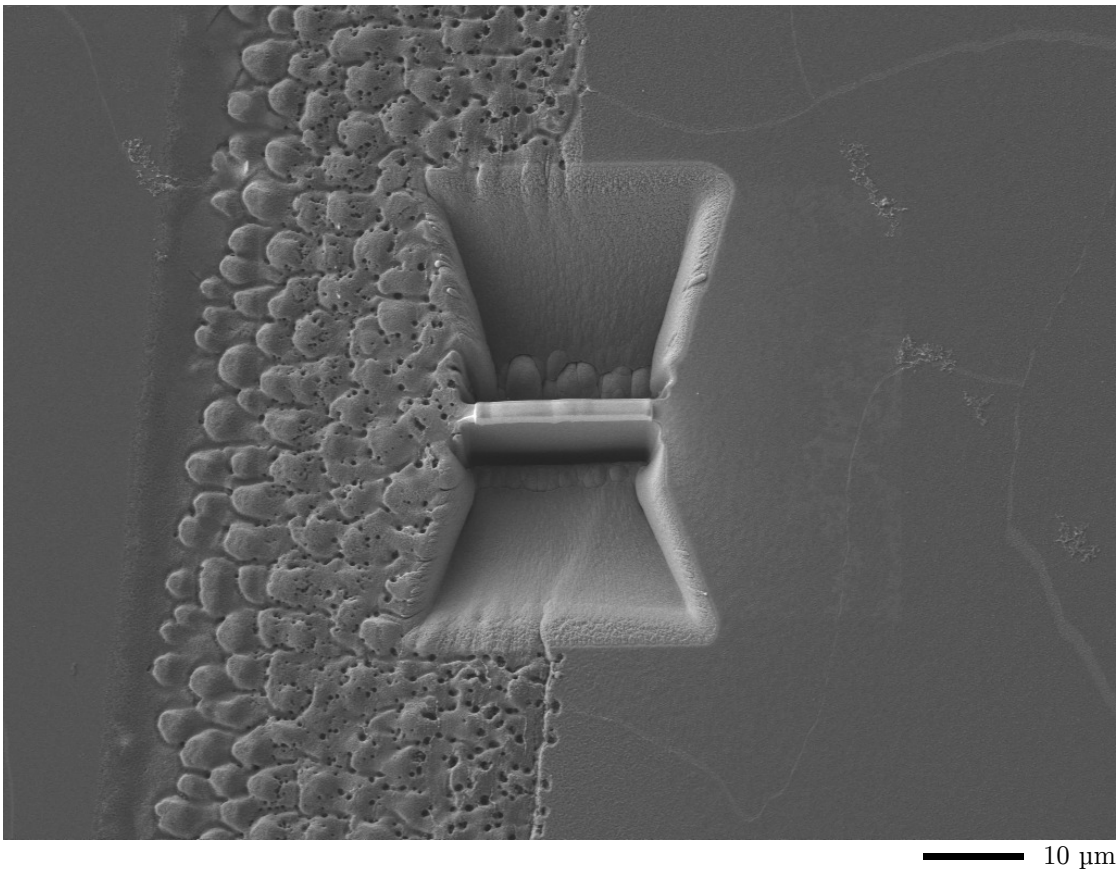


Abbildung 3.29: REM-Aufnahme (ZEISS AURIGA). Herstellung einer TEM-Lamelle: die Lamelle entsteht durch das Herausschneiden von zwei Keilen.

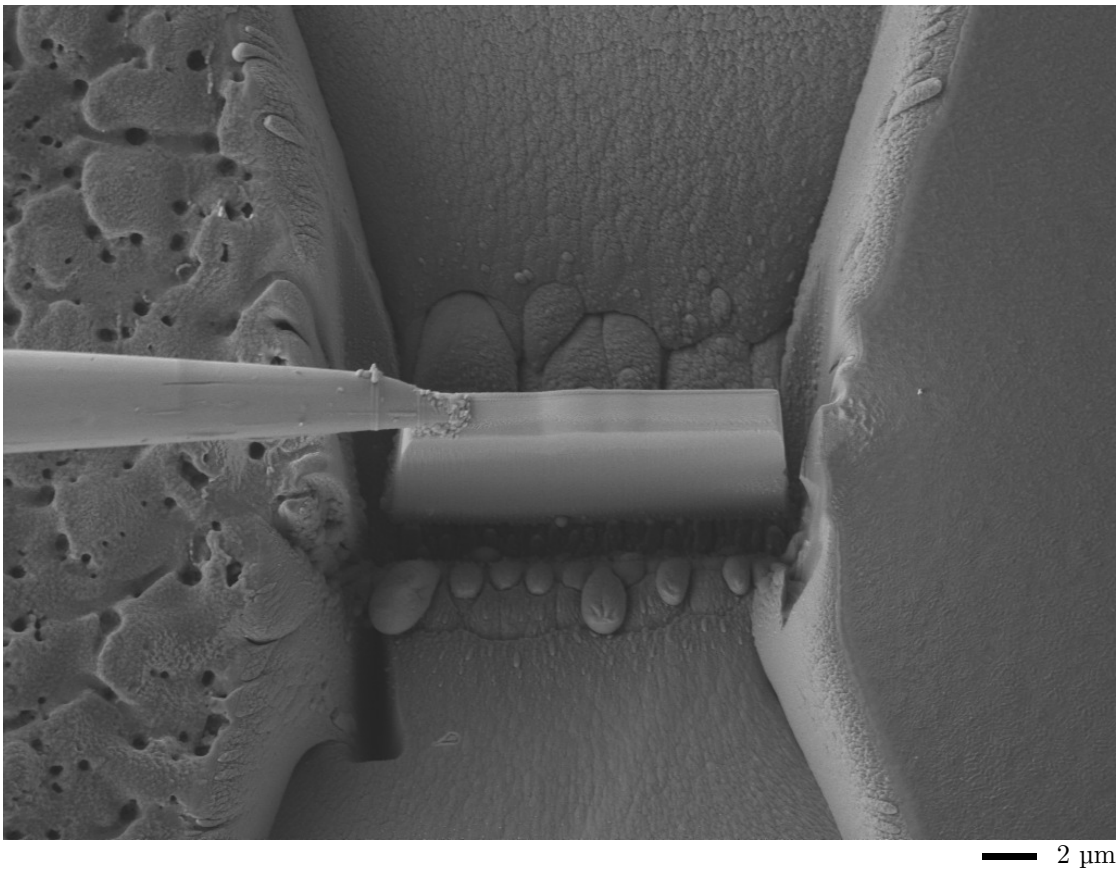


Abbildung 3.30: REM-Aufnahme (ZEISS AURIGA). Herstellung einer TEM-Lamelle: Eine Wolframnadel wird an die Lamelle herangeführt und mit Hilfe eines Platin-Verbindungsstücks an der Lamelle befestigt. Das Platin-Verbindungsstück wird nach dem gleichen Prinzip wie die Platin-Opferschicht erzeugt. Mit Hilfe des Focused Ion Beams wird im Anschluss die Lamelle vollständig vom Substrat gelöst und an der Nadel herausgehoben.

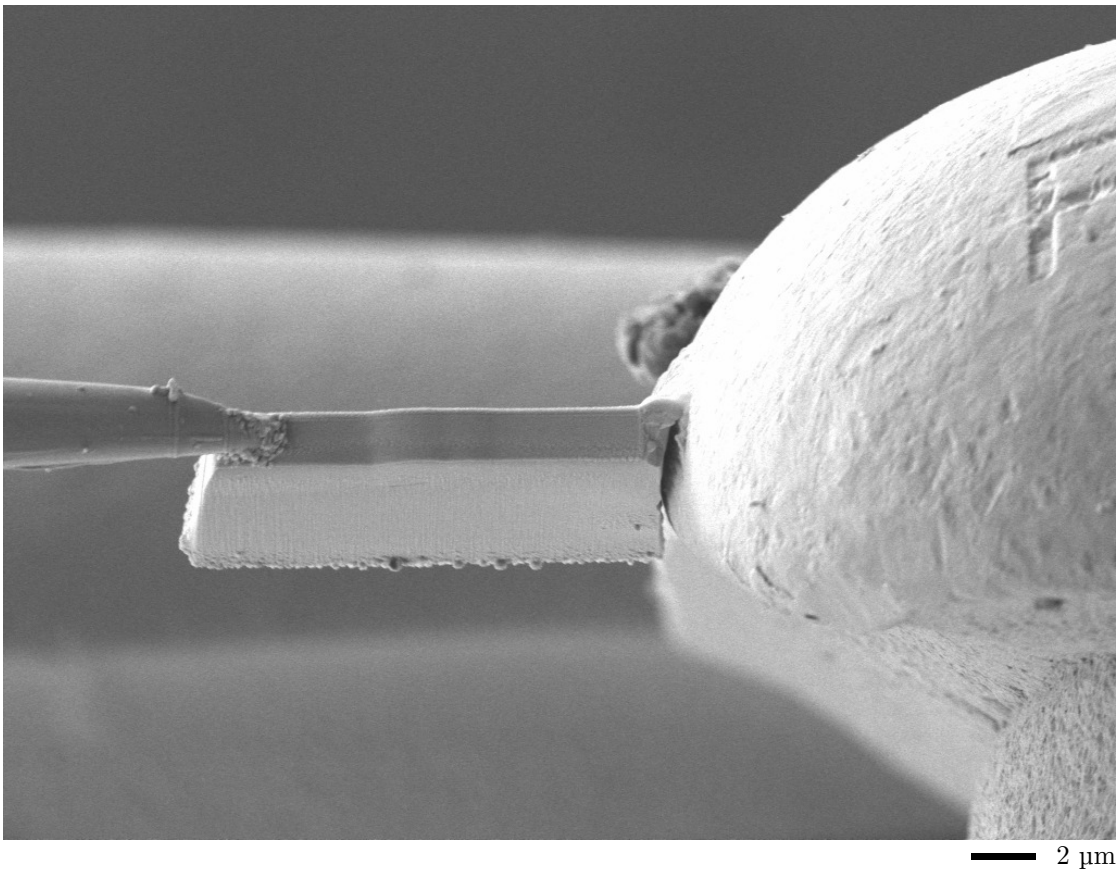


Abbildung 3.31: REM-Aufnahme (ZEISS AURIGA). Anbringung der Lamelle an einem Kuper-Grid mit Hilfe eines weiteren Platin-Verbindungsstücks. Mit dem Focused Ion Beam wird die Verbindung zur Wolframnadel als nächstes wieder gelöst und die Lamelle weiter ausgedünnt.

3.2 Transmissionselektronenmikroskopie

Die Lamelle, deren Anfertigung im vorherigen Abschnitt gezeigt wurde, ist im Transmissionselektronenmikroskop untersucht worden. Mittels EDX-Spektroskopie wurde ein qualitatives Mapping der Elemente Magnesium, Silicium und Sauerstoff erzeugt. Außerdem wurde für ausgewählte Bereiche die Zusammensetzung der Probe quantifiziert.

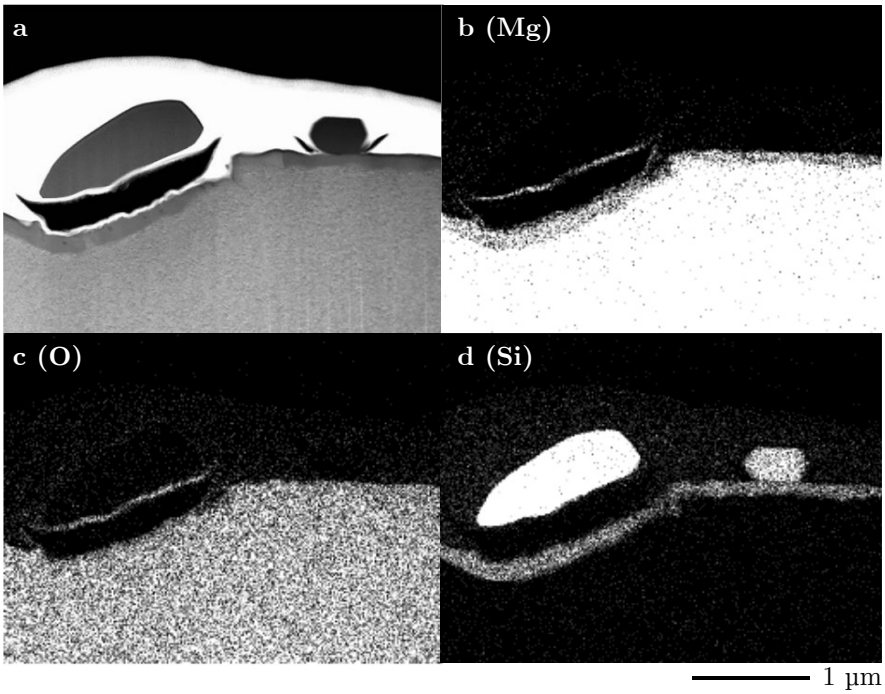


Abbildung 3.32: TEM-Aufnahme (JEOL JEM-2100F-UHR). **a** Übersichtsbild. Zur Orientierung vergleiche mit Abbildung 3.28. Der sehr helle Bereich in der oberen Bildhälfte ist die Platin-Opferschicht. Die Teilbilder **b** bis **c** zeigen qualitativ das Vorkommen der Elemente Magnesium (**b**), Sauerstoff (**c**) und Silicium (**d**). Bereiche mit erhöhter K_α-Strahlung werden hell dargestellt. Die dünne Schicht der Reaktionsprodukte ist mit Silicium angereichert.

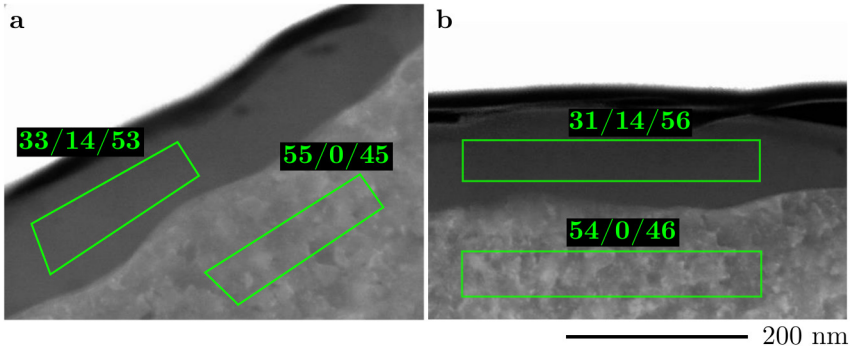


Abbildung 3.33: TEM-Aufnahmen (JEOL JEM-2100F-UHR). Hochauflösende Abbildungen der Schicht mit Reaktionsprodukten und dem unmittelbar darunter liegenden Substrat für zwei Stellen **a** und **b**. Grün markierte Bereiche: EDX-Messungen, beschriftet mit dem Verhältnis Mg/Si/O. Im Bereich des Substrats scheint leichter Überschuss an Mg vorzuliegen. Die mittlere Zusammensetzung der oberen Schicht entspricht formal etwa $Mg_{2,3}Si_1O_{3.9}$. Sowohl beim Tempern (vergleiche Abbildung 3.10) als auch für das Endprodukt der Reaktion geben die EDX-Messungen demnach einen starken Hinweis auf Mg_2SiO_4. Die EDX-Messungen der Oberfläche aus Abbildung 3.25 liefern hingegen wie bereits diskutiert wahrscheinlich ungenaue Ergebnisse, da die Elektronen die Reaktionsschicht durchdringen.

4 Programmierung

Der Hauptteil der Software besteht aus einer Programmbibliothek zur Planung, Platzierung und Auswertung von Beobachtungszonen (Regions of Interest, ROIs) in Videodateien. Der Anspruch war hierbei ein erweiterbares Programm zu entwickeln, mit welchem Videodaten von Experimenten einfach und auch von Nutzern ohne oder mit geringen Programmierkenntnissen ausgewertet werden können. Die folgenden Abschnitte beschreiben die grundsätzlichen Ideen mit Anwendungsbeispielen und sind als Kurzdokumentation zu verstehen. Der vollständige Programmcode ist ab Seite 100 einsehbar.

Die Software enthält außerdem ein Programm zur Umrechnung von BGR-Farbkanal-Intensitäten (dimensionslose Werte zwischen 0 und 255) zu Temperaturwerten. Das Vorgehen und die verwendeten Formeln werden in Kapitel 5 beschrieben. Kurze Hinweise zur Verwendung und der Programmcode beginnen auf Seite 163.

Die Bibliothek enthält einige kleine Hilfsprogramme (zum Beispiel Installationsbefehle oder Skripte zum Schneiden und Rotieren von Videos), die nicht angehängt sind.

4.1 Grundidee

Die Programmstruktur orientiert sich am sogenannten *Observer* Entwurfsmuster (Design Pattern), welches formal im Standardwerk *Design Patterns. Elements of Reusable Object-Oriented Software* von Gamma u. a. beschrieben wird [36]. Das Ziel dieses Entwurfsmusters ist die Definition einer „einer-zu-vielen“ Abhängigkeit zwischen Datenstrukturen. Genau ein zu beobachtendes Objekt (das *Observable*) informiert bei einer Zustandsänderung alle als Beobachter registrierten Objekte (die *Observer*).[1] Jedem einzelnen Beobachter wird es selbst überlassen, ob und wie auf die Zustandsänderung reagiert wird. Zwischen den Beobachtern besteht keine Kopplung, d. h. es gibt keine gegenseitige Beeinflussung. Ein Beobachter besitzt keine Information über die Existenz anderer Beobachter.

[1] Objekte sind konkrete Instanzen von sogenannten Klassen. Beispiel: die Klasse `CircRoi` definiert allgemein die Struktur einer kreisförmigen Region of Interest und existiert genau einmal. Konkrete Instanzen dieser Klasse können dann zum Beispiel verschiedene Zentren und Radien besitzen. Der Sammelbegriff Datenstruktur kann für Klassen und Objekte verwendet werden.

Im konkreten Fall ist das beobachtbare Objekt eine Video-Datenstruktur, welche Einzelbilder (Frames) generiert. Spezielle ROI-Objekte sind die Beobachter, welche immer dann vom Video über eine Zustandsänderung informiert werden, nachdem ein neuer Frame generiert wurde. Die ROIs werten dann den aktuellen Frame innerhalb ihres Beobachtungsgebiets aus und speichern intern die gesammelten Daten. Erst nachdem dieser Vorgang für alle Beobachter abgeschlossen ist, kann das Video zum nächsten Frame voranschreiten. ROIs können zu jedem Zeitpunkt hinzugefügt und wieder entfernt werden. Auf diese Weise lässt sich für jede ROI der Beobachtungszeitraum festlegen. Abbildung 4.1 zeigt die Beziehung zwischen den beteiligten Objekten am Beispiel eines Videos, auf dem drei Beobachtungszonen registriert sind.

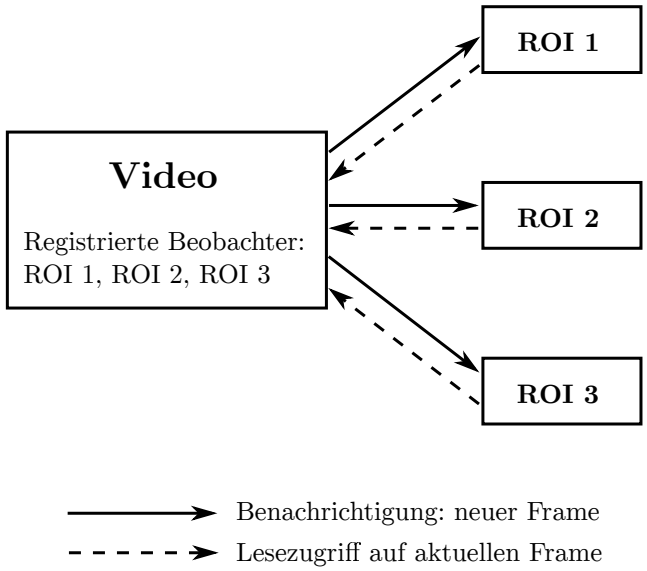


Abbildung 4.1: Video mit drei registrierten ROIs. Wann immer das Video einen neuen Frame generiert, werden alle ROIs benachrichtigt. Diese besitzen nun die Gelegenheit, den aktuellen Frame innerhalb des jeweiligen Beobachtungsgebiets auszuwerten.

4.2 Kurzdokumentation

Das Python-Paket `roitools` besteht aus drei Hauptkomponenten.

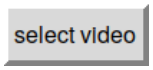
1. Einer grafischen Benutzeroberfläche zum Planen von Beobachtungszonen (Modul `roi_planner_gui.py`).
2. Der Klasse `PyCap`, einer Fassade für die Video-Datenstruktur `VideoCapture` der Bibliothek OpenCV (Modul `vidtools.py`) [37].[2]
3. Der Klasse `RoiCap`, einer Erweiterung von `PyCap`, welcher ROIs als Beobachter hinzugefügt werden können (Modul `roitools.py`).

Die folgenden Abschnitte geben einen kurzen Überblick der essenziellen Werkzeuge, können aber nicht alle Funktionen im Detail beschreiben. Weitere Informationen befinden sich in den Dokumentationskommentaren im Quelltext (den sogenannten „Docstrings“) und können auch mit der Funktion `help` beim Arbeiten mit der Python-Kommandozeile angezeigt werden (Beispiel: `help(PyCap)`). Längere Beispiele für Funktionen, die auf diesen Grundbausteinen aufbauen, können im Modul `examples.py` ab Seite 155 eingesehen werden.

4.2.1 Planung von Beobachtungszonen

Die Planung von ROIs erfolgt über eine grafische Benutzeroberfläche (Graphical User Interface, GUI), welche in Abbildung 4.2 gezeigt ist. Die erste Zeile mit Bedienelementen dient dem Einlesen und Auswerfen von Videodateien (Tabelle 4.1). Rechts daneben wird der Dateiname angezeigt.

Tabelle 4.1: Bedienelemente zur Dateiauswahl.

Schaltfläche	Funktion
	Videodatei laden
	Videodatei auswerfen

Darunter folgt die Fortschrittsanzeige des Videos. In der nächsten Zeile befinden sich Bedienelemente zur Kontrolle des Bildes. Im Bereich „player controls“ sind

[2] Eine Fassade (Facade) ist ein weiteres von Gamma u. a. beschriebenes Entwurfsmuster [36]. Fassaden bieten eine kontextspezifische, vereinfachte Abstraktion eines komplexen Systems und unterdrücken Details. Ein klassisches Beispiel ist die Benutzer-Schnittstelle eines Betriebssystems.

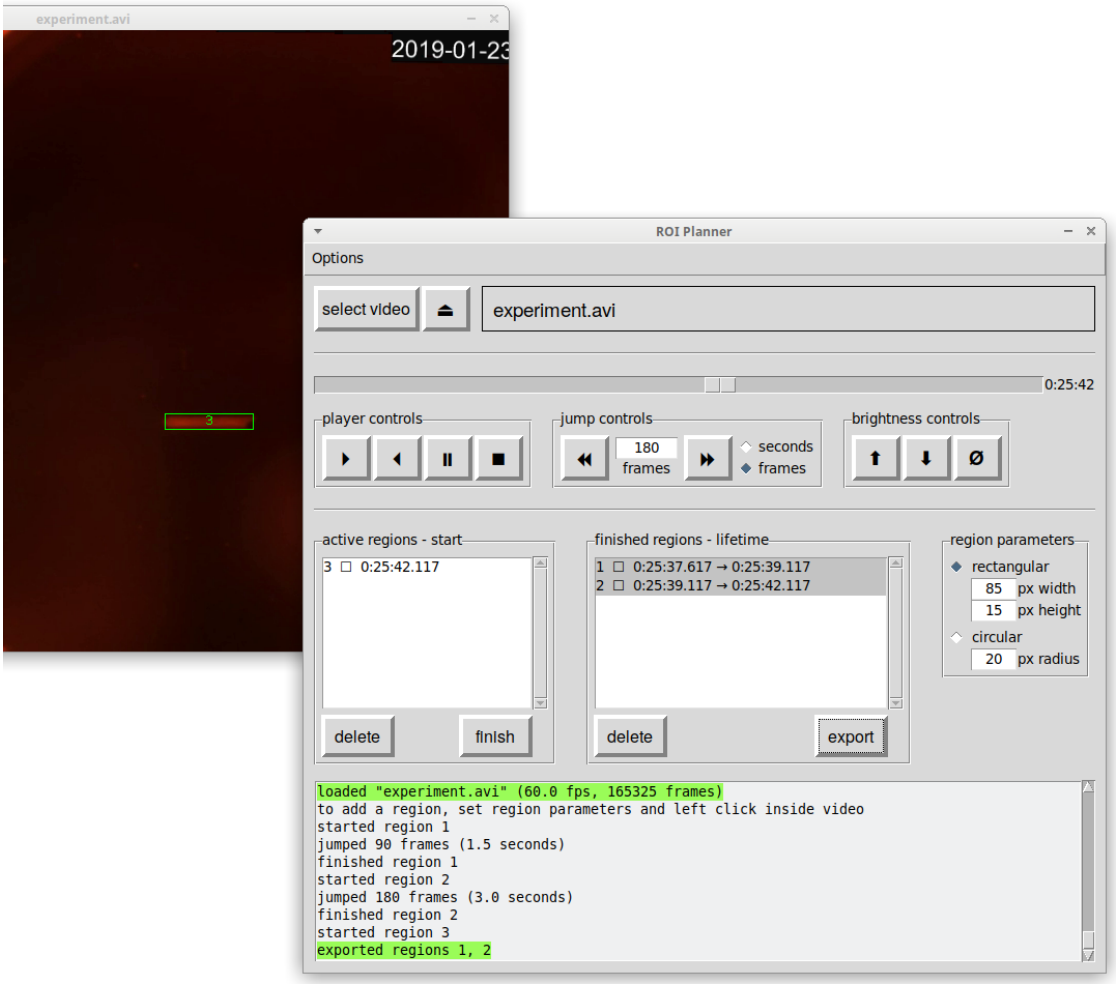


Abbildung 4.2: Vordergrund: Grafische Benutzeroberfläche zur Planung von ROIs. Hintergrund: Laborvideo mit aktiver ROI. Die Bedienelemente werden im Text beschrieben.

Schaltflächen zum Abspielen, Pausieren und Stoppen des Videos gruppiert (Tabelle 4.2). Das Vorwärtsabspielen funktioniert für viele Dateien mit nativer oder nahezu nativer Geschwindigkeit. Das Rückwärtsabspielen ist rechenintensiv und erfolgt für Videos mit hoher Auflösung verlangsamt. Für das Planen von Beobachtungszonen ist die Abspielgeschwindigkeit unwichtig.

Rechts neben dieser Gruppe befinden sich Bedienelemente für exakte Zeitsprünge in der Einheit Sekunden oder Frames („jump controls“). Für Sekunden werden Zahlen mit Nachkommastellen im Eingabefeld zwischen den beiden in Tabelle 4.3 gezeigten Schaltflächen akzeptiert. Die Möglichkeit der Bewegung in Einzelbildern ist hilfreich, falls sehr schnelle Prozesse beobachtet werden müssen.

Ebenfalls nützlich ist die Funktion zum Aufhellen oder Abdunkeln des Videos, falls das zu beobachtende Gebiet mit bloßem Auge schlecht zu erkennen ist. Die

Tabelle 4.2: Bedienelemente zur Abspielkontrolle.

Schaltfläche	Funktion
	Vorwärts abspielen
	Rückwärts abspielen
	Video pausieren
	Video stoppen

Tabelle 4.3: Bedienelemente für Zeitsprünge.

Schaltfläche	Funktion
	Vorwärts springen
	Zurück springen

entsprechenden Schaltflächen sind unter „brightness controls“ gruppiert und in Tabelle 4.4 zusammengefasst.

Tabelle 4.4: Bedienelemente zur Helligkeitskontrolle.

Schaltfläche	Funktion
	Video aufhellen
	Video abdunkeln
	Helligkeit zurücksetzen

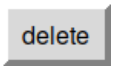
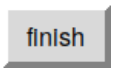
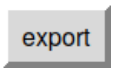
Unter den Schaltflächen zur Bedienung des Videobildes befinden sich zwei Boxen für aktive Regionen („active regions - start“) und beendete Regionen („finished regions - lifetime“). Rechts daneben können ROI-Parameter vorgegeben werden. Es besteht die Wahl zwischen rechteckigen und kreisförmigen Regionen.[3] Für recht-

[3]Mit Programmierkenntnissen lässt sich die Form von Regionen noch spezifischer beeinflussen (siehe Abschnitt 4.2.3).

eckige Beobachtungsregionen müssen die Breite und die Höhe in Pixeln eingegeben werden, für kreisförmige Regionen der Radius.

Neue ROIs können durch die Selektion eines Rechtecks im Videobild mit der linken Maustaste hinzugefügt werden. Ein Rechtsklick fügt eine ROI mit fixen Parametern (Breite und Höhe oder Radius) hinzu. Eine Region erscheint dann als nummerierter farbiger Umriss im Bild und als Zeile zusammen mit dem Anfangszeitpunkt in der Box für aktive Regionen. (In Abbildung 4.2 ist gerade ROI Nummer drei aktiv). Jetzt kann die Position des Videos durch Abspielen, Sprünge, oder durch Anklicken der Fortschrittsleiste verändert werden. Eine oder mehrere aktive Regionen lassen sich durch Anklicken ihrer jeweiligen Zeile auswählen und dann durch Aktivierung der Schaltfläche „finish“ finalisieren. Finalisierte Regionen speichern ihren Beobachtungszeitraum mit einem Zeitstempel für die Start- und die Endzeit und werden von der Box für aktive Regionen in die Box für beendete Regionen überführt. (In Abbildung 4.2 sind Region Nummer eins und zwei bereits finalisiert.) Von hier aus können beendete ROIs dann mit der Schaltfläche „export“ als Tabellendatei exportiert werden, in der alle wichtigen Informationen zu den Regionen (Typ, Ort, Lebensdauer) gespeichert sind. Sowohl aktive als auch bereits beendete Regionen können über den jeweiligen „delete“-Knopf gelöscht werden. Die Schaltflächen zum Arbeiten mit Beobachtungszonen sind noch einmal in Tabelle 4.5 zusammengefasst.

Tabelle 4.5: Bedienelemente für ROIs.

Schaltfläche	Funktion
	Ausgewählte ROIs löschen
	Ausgewählte ROIs beenden
	Ausgewählte ROIs exportieren

Am unteren Rand der Benutzeroberfläche befindet sich ein Textbereich mit Statusmeldungen. Warnungen und Fehler werden ebenfalls hier angezeigt. Ein Warnhinweis wird beispielsweise ausgegeben, wenn einer vom Nutzer angeforderten Sprung-Zeit in Sekunden keine ganze Anzahl von Einzelbildern entspricht. (Beispiel: Sprungzeit 0.05 Sekunden für ein Video mit Framrate 25 fps.) Ein Fehler wird unter anderem dann angezeigt, wenn der Nutzer keine Zugriffsrechte für den zum Exportieren ausgewählten Dateiordner besitzt oder versucht wird, eine defekte Datei einzulesen.

Falls die standardmäßig grünen Umrisse der ROIs schwer zu erkennen sind, kann die Farbe über das in Abbildung 4.3 gezeigte Optionsmenü angepasst werden.

Außerdem existieren die folgenden Einstellungen.

- Automatisches Löschen von exportierten ROIs.
- Automatischer Tausch der Zeitstempel, falls eine Region zu einem Zeitpunkt (bezüglich der Position des Videos) finalisiert wird, der vor ihrem Startzeitpunkt liegt.
- Das Laden von Dateien erlauben, die nicht im AVI-Format vorliegen.

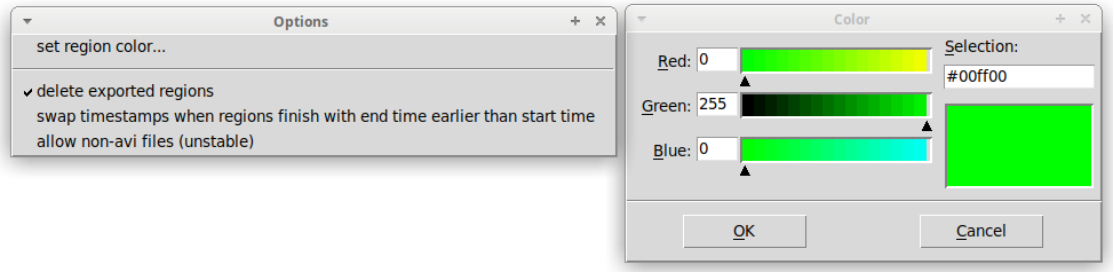


Abbildung 4.3: Links: Optionsmenü der GUI. Rechts: Farbauswahl für die Umrisse von ROIs.

Das Einlesen von anderen Dateiformaten als AVI muss manuell aktiviert werden und ist mit dem Warnhinweis „unstable“ versehen. Formal garantiert die als Grundlage verwendete `VideoCapture` Datenstruktur der Bibliothek OpenCV nur die plattformunabhängige Unterstützung von AVI-Dateien. In den durchgeführten Praxistests konnten allerdings viele unterschiedliche Videoformate (zum Beispiel MP4 und WMV) und sogar Einzelbilder (JPG, PNG, BMP, ...) problemlos eingelesen werden. Als einzige Beeinträchtigung wurde beobachtet, dass Zeitprünge das Ziel gelegentlich um einige Millisekunden verfehlen. Falls sehr exakt gearbeitet werden muss, wird empfohlen, die zu untersuchenden Dateien in das AVI-Format zu konvertieren.

4.2.2 Datenstrukturen für Videos und Frames

Die Klasse `PyCap` ist eine einfach zu bedienende Fassade für die `VideoCapture` Datenstruktur der Bibliothek `OpenCV`. Eine Instanz von `PyCap` repräsentiert ein Video, aus dem beliebige Frames ausgelesen werden können. Es gelten die folgenden Besonderheiten, welche gleich anhand eines Beispiels erklärt werden.

- Attribute können über die Punkt-Notation ausgelesen und gesetzt werden.
- Instanzen von `PyCap` sind iterierbar.

- Die Iteration liefert `Frame` Objekte. Dies sind Arrays mit Zusatzfunktionen.

Die folgenden zwei Programme sind äquivalent und verdeutlichen den Unterschied zwischen der Verwendung der Klassen `VideoCapture` und `PyCap`. Ein Video (sample.avi) wird eingelesen und zwischen Sekunde zehn und zwölf ausgewertet (mit einer nicht weiter spezifizierten Beispielfunktion `analyze`). Außerdem werden die Einzelbilder mit einer Verzögerung von 25 Millisekunden angezeigt.

Beispiel A: Verwendung von `VideoCapture`

```
1  import cv2
2
3  video = cv2.VideoCapture('sample.avi')
4  video.set(cv2.CAP_PROP_POS_MSEC, 10000)
5
6  while True:
7      success, frame = video.read()
8
9      if not success:
10         break
11
12     msec = video.get(cv2.CAP_PROP_POS_MSEC)
13
14     if msec > 12000:
15         break
16
17     analyze(frame)
18
19     cv2.imshow('sample.avi', frame)
20     cv2.waitKey(25)
```

Beispiel B: Verwendung von `PyCap`

```
1  from vidtools import PyCap, wait
2
3  video = PyCap('sample.avi')
4  video.pos_msec = 10000
5
6  for frame in video:
7      if video.pos_msec > 12000:
8          break
9
10     analyze(frame)
11
12     frame.show()
13     wait(25)
```

`PyCap` Objekte unterstützen komfortablen Lese- und Schreibzugriff auf Attribute wie `pos_msec` (Position in Millisekunden) direkt über die Punkt-Notation. Attribute von `VideoCapture` müssen über die Funktionen `cv2.get` und `cv2.set` gelesen und gesetzt werden. Der Unterschied wird beim Vergleich von Zeile 4 in Beispiel A mit Zeile 4 in Beispiel B (Attribut setzen) und von Zeile 12 in Beispiel A mit Zeile 7 in Beispiel B (Attribut auslesen) deutlich. Tabelle 4.6 zeigt eine kleine Auswahl der zur Verfügung stehenden Attribute. Einige Attribute (zum Beispiel die Gesamtzahl der Frames) unterstützen aus offensichtlichen Gründen nur den Lese-, aber keinen Schreibzugriff.

Tabelle 4.6: Ausgewählte Video-Attribute.

`VideoCapture`	`PyCap`	Bedeutung
`cv2.CAP_PROP_POS_MSEC`	`pos_msec`	Position in Millisekunden
`cv2.CAP_PROP_POS_FRAMES`	`pos_frames`	Position in Frames
`cv2.CAP_PROP_FRAME_COUNT`	`frame_count`	Gesamtanzahl Frames
`cv2.CAP_PROP_FPS`	`fps`	Framerate
`cv2.CAP_PROP_FRAME_HEIGHT`	`frame_height`	Horizontale Auflösung
`cv2.CAP_PROP_FRAME_WIDTH`	`frame_width`	Vertikale Auflösung

`PyCap` Objekte sind iterierbar, was sich in Beispiel B dadurch zeigt, dass eine `for` Schleife bei der Iteration über die Frames des Videos benutzt werden darf. Des Weiteren kann mit Hilfe des Befehls `next(video)` explizit der nächste Frame erzeugt werden (im Beispiel nicht verwendet). Die Schleife bricht automatisch ab, falls kein weiterer Frame generiert werden kann (Zeile 6). Bei der Verwendung eines `VideoCapture` muss hingegen innerhalb einer „ewigen“ `while True` Schleife manuell jeder Frame über `video.read()` ausgelesen werden. Schlägt der Leseprozess fehl, wird die Schleife nicht automatisch abgebrochen (Zeile 6 bis 10 in Beispiel A). Objekte der Klasse `PyCap` verfügen außerdem über die Methode `play`. Soll das Video nur abgespielt werden, vereinfacht sich die Logik in Beispiel B nach der Definition von `video` zu einer einzigen Zeile.

```
video.play(delay=25, start=10000, stop=12000)
```

Bei Frames handelt es sich formal um dreidimensionale Arrays mit der Größe `frame_height` × `frame_width` ×3. Abstrakt können sich diese Arrays als zweidimensionale Matrizen vorgestellt werden. Die Anzahl von Spalten und Zeilen entsprechen der Breite und Höhe des Videobildes in Pixeln. Jedes Matrixelement repräsentiert einen Pixel durch die Speicherung von drei Zahlen für die BGR-Farbkanal-Werte. Die von `PyCap` generierten Frames verfügen über nützliche Zu-

satzfunktionen wie `show`. (Vergleiche Zeile 19 in Beispiel A mit Zeile 12 in Beispiel B.)

4.2.3 Beobachtungsregionen und beobachtbare Videos

Die ROIs sind in der in Abbildung 4.4 gezeigten Klassenhierarchie implementiert. Nur die Klassen `MeanRectRoi` und `MeanCircRoi` sammeln Daten, wenn sie auf einem Video registriert sind. Sie bieten Beispielimplementierungen und werden zur Auswertung der Laborvideos in Abschnitt 5 verwendet. Die anderen Klassen dienen als Basis für Erweiterungen durch den Nutzer mit selbst definierten Auswertungsalgorithmen. Eine mögliche Weiterentwicklung des Programms ist die Implementierung von kreisförmigen ROIs als Spezialfall einer allgemeinen, elliptischen ROI.

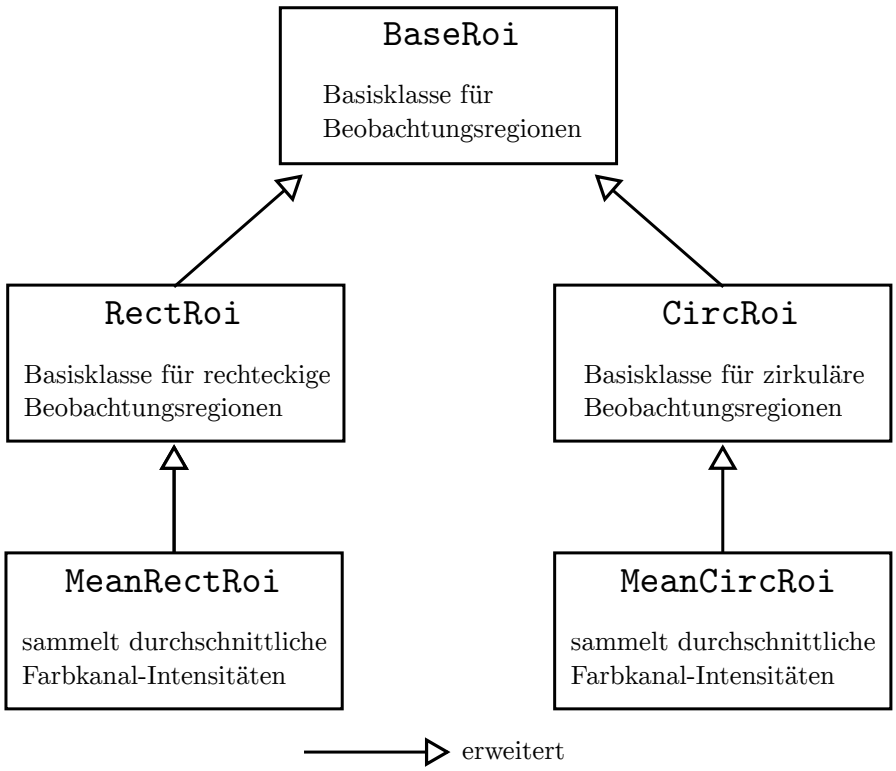


Abbildung 4.4: Klassenhierarchie für ROIs.

Die Klasse `RoiCap` ist eine Erweiterung von `PyCap` und stellt ein Video dar, dem Beobachtungszonen hinzugefügt werden können. Wann immer ein Frame generiert wird, werden alle registrierten ROIs automatisch benachrichtigt. Danach werden die Umrisse der ROIs auf das aktuelle Bild gezeichnet. (Das Sammeln von

Daten durch die ROIs erfolgt vor dem Zeichnen.) Ein neuer Frame wird durch Aufruf der Methode `RoiCap.next` erzeugt. Dies kann explizit erfolgen oder implizit durch Verwendung einer `for` Schleife wie in Beispiel B in Abschnitt 4.2.2. Der folgende Code registriert eine kreisförmige ROI mit den Mittelpunkts-Koordinaten $(100, 100)$ und Radius 20 (in Pixeln).

```
1 from roitools import RoiCap, MeanCircRoi
2
3 video = RoiCap('sample.avi')
4 roi = MeanCircRoi((100, 100), 20)
5 video.add_roi(roi)
```

Um Daten über die gesamte Laufzeit des Videos zu sammeln, reicht es aus, alle Frames mit einer `for` Schleife zu generieren. (Für lange Videos kann dies eine Weile dauern.)

```
6 for frame in video:
7     pass
```

Alternativ kann auch die Methode `play` verwendet werden, um das Video langsamer, aber für den Nutzer sichtbar und mit eingezeichneten Umrissen der Beobachtungszonen abzuspielen.

Sind nur bestimmte Zeitintervalle von Interesse, kann das Attribut `pos_msec` oder `pos_frames` analog zu den Beispielen in Abschnitt 4.2.2 verwendet werden. Achtung: das Setzen dieser Attribute generiert keine Frames! Die für jedes Einzelbild gesammelten Daten werden an die interne Liste `roi.collected` angehängt und können mit der Methode `to_file` in Tabellenform exportiert werden.

```
8 with open('results.csv') as results:
9     roi.to_file(results)
```

Für Abfolgen von Standardaufgaben, wie das Hinzufügen und rechtzeitige Löschen mehrerer, zeitlich versetzter ROIs, den anschließenden Export der gesammelten Daten und die grafische Darstellung derselbigen, existieren bereits die Komfortfunktionen `get_roidata` und `collect_plot_export` im Modul `examples.py`. Diese Funktionen verlangen lediglich den Pfad zur zu untersuchenden Videodatei und eine Liste von ROIs zusammen mit der jeweiligen Start- und Endzeit der Beobachtung. Es wird auf die Dokumentationskommentare im Quelltext verwiesen.

Die Erstellung eigener ROIs mit benutzerdefinierter Auswertung von Frames ist sehr einfach. Es reicht aus, eine der zur Verfügung gestellten Basisklassen durch Implementierung der Methode `collect` zu erweitern. Das folgende Beispiel definiert eine rechteckige ROI. Es soll der Median-Wert der Intensitäten des roten Farbkanals innerhalb des Beobachtungsbereichs berechnet werden. Dieser wird dann zusammen mit der Nummer des aktuellen Frames abgespeichert. Dies ist keine besonders nützliche Auswertung, zeigt aber das generelle Vorgehen.

```
1  from roitools import RectRoi
2  import numpy as np
3
4  class MedianRedRectRoi(RectRoi):
5      def collect(self):
6          '''called when observed video advances a frame
7          computes median of red channel, returns (frame number, median) tuple'''
8
9          roi_area = self.get_rect()
10         red_median = np.median(roi_area.red)
11         frame_number = self.cap.pos_frames
12
13         record = (frame_number, float(red_median))
14         return record
```

Die Methode `collect` wird *automatisch* immer genau dann aufgerufen, wenn das beobachtete Video ein Einzelbild generiert. In Zeile 8 wird die rechteckige Beobachtungsregion der ROI aus dem aktuellen Frame des beobachteten Videos ausgeschnitten. Die Operation erzeugt ein neues `Frame` Objekt, welches dann ausgewertet wird. Das beobachtete Video (ein `RoiCap`) ist unter `self.cap` erreichbar. Der Rückgabewert der Funktion ist ein Tupel aus Frame-Nummer und Medianwert und wird automatisch an eine interne Liste (namens `collected`) der ROI angehängt, welche über die Punkt-Notation zugänglich ist. Falls der Export der Daten gewünscht ist, wird empfohlen ebenfalls eine `to_file` Methode zu implementieren. Für ein längeres Beispiel sei auf den Code der Methoden `collect` und `to_file` der Klassen `MeanRectRoi` und `MeanCircRoi` verwiesen.

Durch Anwendung einer sogenannten boolschen Maske auf das mit `get_rect` ausgeschnittene Rechteck ist außerdem eine Implementierung von ROIs mit beliebiger Form möglich. Dies verlangt allerdings fortgeschrittene Programmierkenntnisse. Als Beispiel kann die Implementierung der kreisförmigen ROI `CircRoi` herangezogen werden.

5 Auswertung der Videodaten und Temperaturbestimmung

Die Auswertung eines gemäß Abschnitt 2.3 durchgeführten Experiments gliedert sich in die folgenden Teilschritte.

1. Übersichtsdiagramm: zeitliche Auftragung der mittleren Farbkanal-Intensitäten im Bereich der Probe.

2. Umfeld-Korrektur: Berücksichtigung des von zu großen Regionen abgedeckten Probenumfelds.

3. Pyrometrie: Zuordnung von Intensitäten zu Temperaturwerten.

4. Farbkanal-Vergleich: Überprüfung, ob berechnete Temperaturänderungen konsistent bezüglich aller Farbkanäle sind.

5. Fehlerbetrachtung: Abschätzung der Auswirkungen getroffener Annahmen und Vereinfachungen auf die Temperaturwerte.

6. Zuordnung von Temperaturen: Darstellung von Intensitäten mit zugehöriger Temperatur.

5.1 Übersichtsdiagramm

Nach der Aufzeichnung der Videodaten gemäß des in Abschnitt 2.3 beschriebenen Vorgehens wurde zunächst ein Übersichtsdiagramm angefertigt. Dieses zeigt den zeitlichen Verlauf der mittleren Farbkanal-Intensitäten im Bereich der Probe. Es handelt sich hierbei um dimensionslose Werte zwischen 0 und 255. Mit Hilfe dieses Graphen kann sicherer als mit dem bloßen Auge beurteilt werden, ob und bei welchen Heizleistungen der Blinkeffekt eingetreten ist. Außerdem kann überprüft werden, ob sich im Laborjournal vermerkte Beobachtungen zum Emissionsverhalten in den Intensitäten der drei Farbkanäle widerspiegeln.

Abbildung 5.1 zeigt diesen Graphen für ein Experiment mit einem Silicium-Cantilever, bei dem die Heizleistung etwa alle fünf Minuten erhöht wurde. In

der Darstellung werden fünf Stufen mit jeweils konstanter Heizleistung durchlaufen (Spannung und Stromfluss sind im Diagramm vermerkt). Der charakteristische, transiente Emissionseffekt zeigt sich auf allen Stufen durch Einbrüche in den Farbkanal-Intensitäten. Die Frequenz und Intensität der Einbrüche steigt mit der Heizleistung. Die verwendeten ROIs sind in Abbildung 5.2 dargestellt. Bei der Beschreibung der folgenden Auswertungsschritte wird sich auf dasselbe Experiment bezogen.

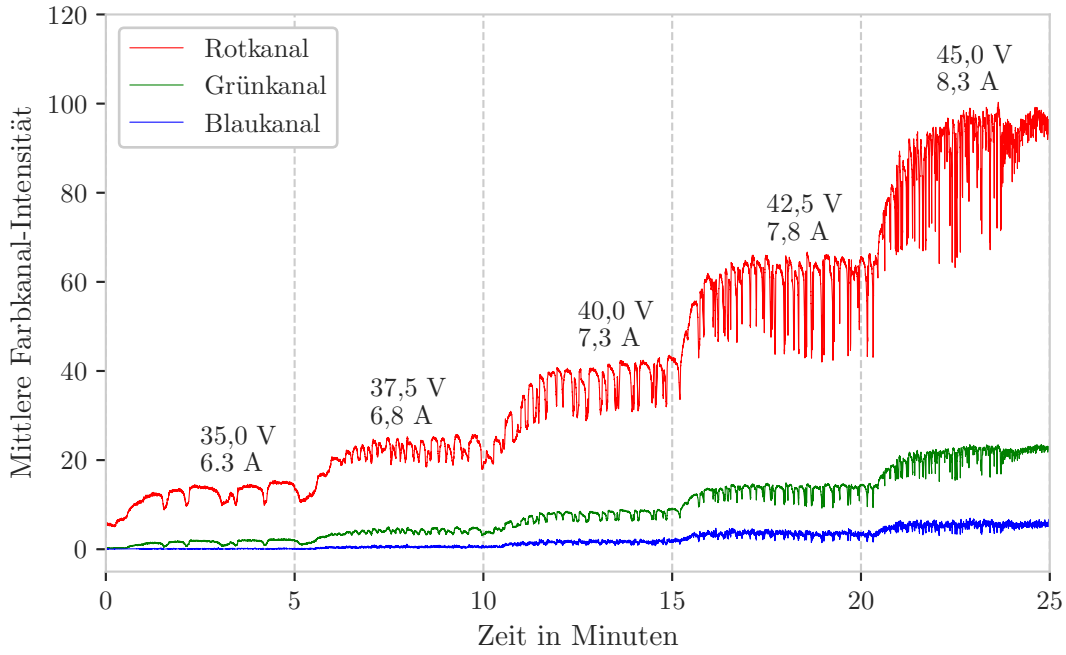


Abbildung 5.1: Zeitlicher Verlauf der Farbkanal-Intensitäten im Bereich der Probe für ein Cantilever-Experiment mit fünf Heizstufen.

Die zeitliche Auftragung der Kanalintensitäten wurde mit Hilfe der in Kapitel 4 vorgestellten Software angefertigt. Es wurden rechteckige Beobachtungsregionen verwendet, welche die mittleren Farbkanal-Intensitäten für jedes Einzelbild berechnen. Der Probentisch ist über die Dauer eines Experiments nicht absolut ortsfest und bewegt sich typischerweise mit einer mittleren Geschwindigkeit von etwa 2 mm pro Stunde. Aus diesem Grund musste die Probe mit Beobachtungsregionen „verfolgt“ werden. Für Abbildung 5.1 wurden 50 Regionen verwendet, die jeweils im Abstand von etwa 30 Sekunden platziert wurden (siehe Abbildung 5.2). Der Cantilever ist zu Beginn des Videos kaum zu erkennen. Die Planung von Beobachtungsregionen ist dennoch möglich, indem das Bild mit Hilfe der grafischen Benutzeroberfläche stark aufgehellt wird.

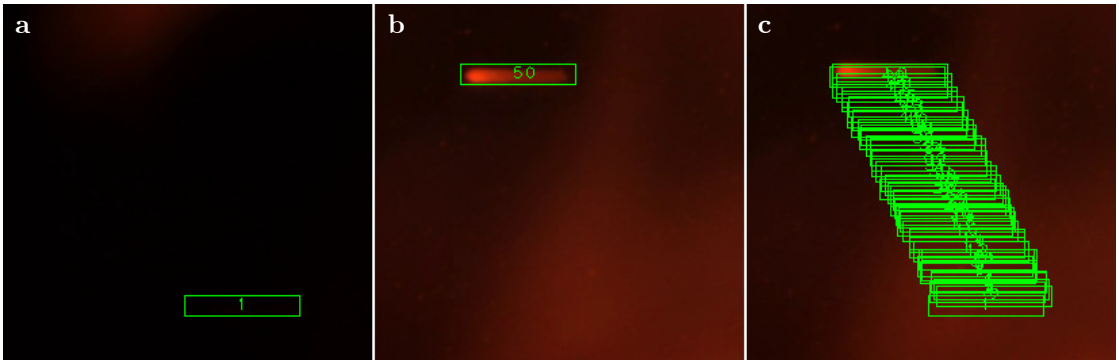


Abbildung 5.2: Platzierung von Beobachtungsregionen. **a** Erste ROI. **b** Letzte ROI. **c** Übersicht aller ROIs. Die Regionen waren zu verschiedenen Startzeitpunken für jeweils etwa 30 Sekunden aktiv.

5.2 Umfeld-Korrektur

Bei der Verwendung von Beobachtungsregionen gemäß Abbildung 5.2 ergeben sich zwei Schwierigkeiten. Zum einen können zum Zeitpunkt eines ROI-Wechsels Artefakte in Form von kleinen, abrupten Sprüngen in der zeitlichen Auftragung der Kanalintensitäten entstehen. Diese sind in Übersichtsdarstellungen wie Abbildung 5.1 kaum zu erkennen, da ein langes Zeitintervall und ein großer Intensitätsbereich abgebildet sind. Die Artefakte zeigen sich allerdings bei höherer zeitlicher Auflösung. Zum anderen sind die Regionen etwas größer als der Cantilever und schließen daher einen Teil des Probenumfelds mit ein. Da die mittleren Farbkanal-Intensitäten nur für den Probenbereich ausgewertet werden sollen, berechnen zu große Beobachtungsregionen einen verfälschten Wert. Eine „perfekte“ Platzierung der Regionen ist aufgrund der Bewegung des Cantilevers bezüglich der Bildkoordinaten allerdings nicht praktikabel, da nach wenigen Einzelbildern bereits ein (manueller) Wechsel der Region erfolgen müsste.

Aus diesen Gründen wurde ein Auswertungverfahren verwendet, welches den Regionswechsel in Zeitabschnitten mit gleicher Heizleistung vermeidet. Die Berechnung der mittleren Intensitäten der Probe aus den mittleren Kanalintensitäten einer größeren, die Probe umschließenden Beobachtungsregion ist dann mit angemessenem Aufwand möglich. Die zu den Heizspannungen 35 bis 42,5 V gehörenden Stufen wurden jeweils ohne Wechsel der ROI über einen Zeitraum von 90 Sekunden beobachtet. (Bei 45 V setzt der Schmelzprozess ein, diese Stufe wird gesondert behandelt.) Die Beobachtungszeit stellt einen Kompromiss dar: zum einen sollen möglichst viele Einbrüche der Kanalintensitäten aufgezeichnet werden, zum anderen soll die ROI möglichst wenig Probenumfeld einschließen. Lange Beobachtungszeiten ohne ROI-Wechsel erfordern aufgrund der Bewegung des Bildes eine große ROI, damit die Probe die Beobachtungszone nicht verlässt. Die vier verwendeten Beobachtungsregionen sind in Abbildung 5.3 gezeigt.

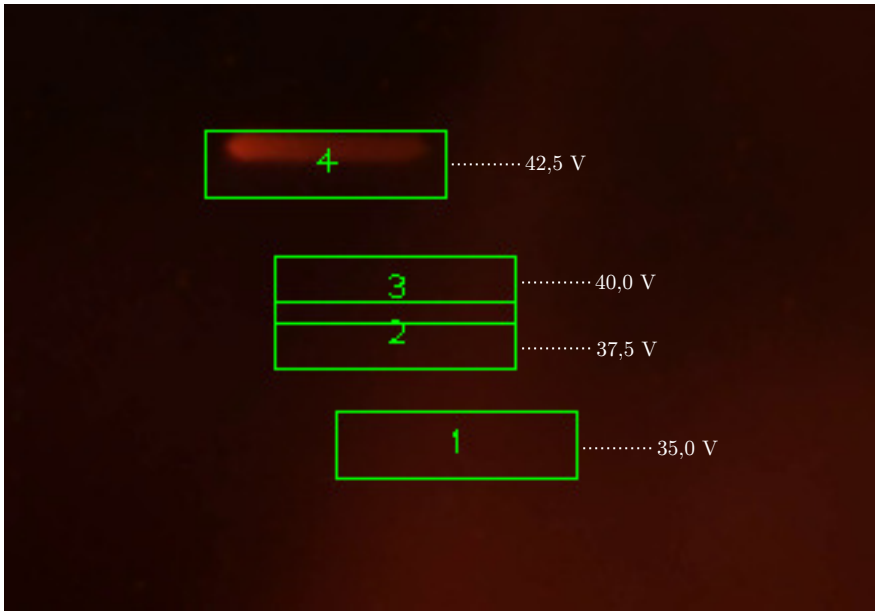


Abbildung 5.3: Übersicht der ROIs für die Stufen mit den Heizspannungen 35,0 bis 42,5 V. Die ROIs waren zu verschiedenen Startzeitpunkten über einen Zeitraum von jeweils 90 Sekunden aktiv.

Als nächstes erfolgte die Berechnung der mittleren Proben-Intensitäten aus den mit Hilfe der Beobachtungsregionen gemessenen Mittelwerten. Für jede ROI gilt

$$\bar{I}_{\mathrm{ROI}}^{\mathrm{Kanal}} = x_{\mathrm{Umfeld}} \cdot \bar{I}_{\mathrm{Umfeld}}^{\mathrm{Kanal}} + x_{\mathrm{Canti}} \cdot \bar{I}_{\mathrm{Canti}}^{\mathrm{Kanal}} . \tag{5.1}$$

$\bar{I}_{\mathrm{ROI}}^{\mathrm{Kanal}}$ Mittlere Intensität in der Beobachtungsregion. Wurde mit der entwickelten Software für jedes beobachtete Einzelbild berechnet. Kanal $\in \{\mathrm{rot}, \mathrm{grün}, \mathrm{blau}\}$ bezeichnet den Farbkanal.

$\bar{I}_{\mathrm{Canti}}^{\mathrm{Kanal}}$ Mittlere Intensität des Cantilevers. Das Ziel der Umfeld-Korrektur ist die Berechnung dieser Größe.

$\bar{I}_{\mathrm{Umfeld}}^{\mathrm{Kanal}}$ Mittlere Intensität des Umfelds (ROI ohne Cantilever). Wird einmal pro ROI bestimmt und innerhalb deren Lebensdauer als konstant angenommen, da sich die Heizleistung nicht ändert.

x_{Canti} Anteil des Cantilevers an der Fläche der ROI (Konstante).

x_{Umfeld} Anteil des Umfelds an der Fläche der ROI (Konstante).

Umstellen von Gleichung 5.1 liefert

$$\bar{I}_{\mathrm{Canti}}^{\mathrm{Kanal}} = \frac{\bar{I}_{\mathrm{ROI}}^{\mathrm{Kanal}} - x_{\mathrm{Umfeld}} \cdot \bar{I}_{\mathrm{Umfeld}}^{\mathrm{Kanal}}}{x_{\mathrm{Canti}}}. \tag{5.2}$$

Mit Hilfe dieser Formel ist die Berechnung der Cantilever-Intensität für jedes auszuwertende Einzelbild möglich. Der Wert von $\bar{I}_{\mathrm{ROI}}^{\mathrm{Kanal}}$ ist für jeden Frame bekannt. Die Fläche der ROI A_{ROI} (in px^2) kann aus den bei der Planung der ROIs exportierten Daten abgelesen werden. Für rechteckige ROIs werden die (x, y)-Koordinaten zweier gegenüberliegender Ecken gespeichert. Alternativ kann A_{ROI} mit einem Bildbearbeitungsprogramm bestimmt werden (siehe Abbildung 5.4). Die Anteile x_{Canti} und x_{Umfeld} berechnen sich mit der Cantilever-Fläche A_{Canti} dann über

$$x_{\mathrm{Canti}} = \frac{A_{\mathrm{Canti}}}{A_{\mathrm{ROI}}} \tag{5.3}$$

$$x_{\mathrm{Umfeld}} = 1 - x_{\mathrm{Canti}}. \tag{5.4}$$

Die Flächen A_{ROI} und A_{Canti} sind konstant, da alle ROIs gleich groß gewählt wurden und sich die Fläche des Cantilevers im Beobachtungszeitraum nicht ändert. Somit sind auch x_{Canti} und x_{Umfeld} Konstanten, zu deren Bestimmung es ausreicht, die Länge L_{Canti} und Breite B_{Canti} des Cantilevers in Pixeln mit einem Bildbearbeitungsprogramm zu messen. Der Wert von A_{Canti} ergibt sich dann aus $L_{\mathrm{Canti}} \cdot B_{\mathrm{Canti}}$. Hierbei wurde darauf geachtet, dass das Verhältnis von L_{Canti} und B_{Canti} mit dem Verhältnis dieser Längen in den zuvor angefertigten Rasterelektronenmikroskopie-Aufnahmen übereinstimmt (siehe Kapitel 3).[1] Das Vorgehen wird mit Hilfe von Abbildung 5.4 veranschaulicht. Es gilt $A_{\mathrm{ROI}} = 91\,\mathrm{px} \cdot 26\,\mathrm{px} = 2366\,\mathrm{px}^2$ sowie $L_{\mathrm{Canti}} = 74\,\mathrm{px}$ und $B_{\mathrm{Canti}} = 11\,\mathrm{px}$. Hieraus ergeben sich $A_{\mathrm{Canti}} = 11\,\mathrm{px} \cdot 74\,\mathrm{px} = 814\,\mathrm{px}^2$ und $x_{\mathrm{Canti}} = 814\,\mathrm{px}^2/2366\,\mathrm{px}^2 \approx 0{,}34$ sowie $x_{\mathrm{Umfeld}} \approx 1 - 0{,}34 = 0{,}66$. Die Auswirkungen der konkreten Wahl der Cantilever-Fläche auf die nachfolgend berechneten Temperaturwerte werden im Abschnitt zur Fehlerbetrachtung diskutiert.

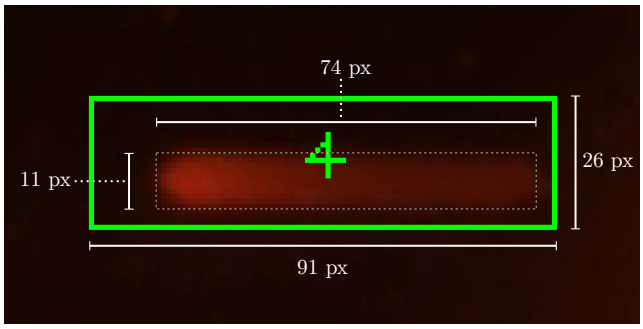


Abbildung 5.4: Längenmessungen zur Bestimmung von x_{Canti} und x_{Umfeld}.

[1] Das sich aus den Herstellerangaben ergebenede Verhältnis von $\frac{L_{\mathrm{Canti}}}{B_{\mathrm{Canti}}} = \frac{450\,\mu\mathrm{m}}{50\,\mu\mathrm{m}} = 9$ kann nur als grober Richtwert herangezogen werden. Die Länge und Breite unterliegen herstellungsbedingten Schwankungen und Cantilever-Proben brechen bei der Präparation oft nicht sauber an der Basis des Silicium-Basiskörpers ab.

Um Formel 5.2 anwenden zu können, müssen außerdem die Werte von $\bar{I}^{\text{Kanal}}_{\text{Umfeld}}$ für jede ROI und jeden Farbkanal bestimmt werden. Die in Abbildung 5.3 gezeigten Beobachtungsregionen sind jeweils über Zeiträume mit konstanter Heizleistung aktiv. Daher reicht es aus, für jede ROI exemplarisch einen Frame aus dem Beobachtungszeitraum herauszugreifen. Mit der Grafiksoftware GIMP wurde eine Selektion erstellt, welche nur das Umfeld des Cantilevers innerhalb der betrachteten ROI abdeckt [38]. Hierfür wurde die Fläche der ROI selektiert und dann eine Selektion mit Länge L_{Canti} und Breite B_{Canti} für den Cantilever von dieser abgezogen. Die Mittelwerte der Farbkanal-Intensitäten des Umfelds wurden anschließend mit der Histogramm-Funktion des Programms ausgelesen. Bei diesem Vorgehen wurden unveränderte Frames ohne eingezeichnete Beobachtungszonen verwendet, da deren Umrisse und Identifikationsnummern das Ergebnis verfälschen würden. Die Werte für $\bar{I}^{\text{Kanal}}_{\text{Umfeld}}$ sind in Tabelle 5.1 zusammengefasst.

Tabelle 5.1: $\bar{I}^{\text{Kanal}}_{\text{Umfeld}}$ für die ROIs aus Abbildung 5.3.

	$\bar{I}^{\text{blau}}_{\text{Umfeld}}$	$\bar{I}^{\text{grün}}_{\text{Umfeld}}$	$\bar{I}^{\text{rot}}_{\text{Umfeld}}$
ROI 1 (35,0 V)	0,1	0,4	8,4
ROI 2 (37,5 V)	0,3	2,2	14,1
ROI 3 (40,0 V)	1,3	5,2	23,6
ROI 4 (42,5 V)	2,4	7,3	31,8

Die Umrechnung der gemessenen Werte für $\bar{I}^{\text{Kanal}}_{\text{ROI}}$ in die gesuchten Werte für $\bar{I}^{\text{Kanal}}_{\text{Canti}}$ über Gleichung 5.2 kann auf einfache Weise mit einem Tabellenkalkulationsprogramm erfolgen. Die Daten für $\bar{I}^{\text{Kanal}}_{\text{ROI}}$ jedes betrachteten Einzelbildes werden von der in Kapitel 4 beschriebenen Software bereits in tabellarischer Form exportiert. Die Anteile x_{Canti} und x_{Umfeld} sind Konstanten und $\bar{I}^{\text{Kanal}}_{\text{Umfeld}}$ ist für jede ROI und jeden Farbkanal konstant. Im konkreten Fall wurde die Python-Bibliothek `pandas` für die Umrechnung verwendet [39].

5.3 Pyrometrie

Die Zuordnung von $\bar{I}^{\text{Kanal}}_{\text{Canti}}$ zu absoluten Temperaturen erfolgt mit Hilfe des planckschen Strahlungsgesetzes in der Form

$$I_\lambda = 2\pi hc^2 \left[\frac{1}{\mathrm{e}^{\frac{hc}{\lambda kT}} - 1}\right] \frac{1}{\lambda^5} \quad [40, \text{S. } 945]. \tag{5.5}$$

I_λ Spektrale Intensitätsdichte (Einheit $\mathrm{W\,m^{-3}}$).

h Plancksches Wirkungsquantum.

c Lichtgeschwindigkeit.

λ Wellenlänge.

k Boltzmann-Konstante.

T Absolute Temperatur.

Der Faktor $2\pi hc^2$ wird zur Konstanten A zusammengefasst

$$I_\lambda = A \left[\frac{1}{\mathrm{e}^{\frac{hc}{\lambda k T}} - 1} \right] \frac{1}{\lambda^5}. \tag{5.6}$$

Die Integration von I_λ und Einführung der unbestimmten Faktoren G (Geometriefaktor) und $U^{\,\mathrm{Kanal}}$ (Umrechnungsfaktor) führen auf einen Ansatz für $\bar{I}^{\mathrm{Kanal}}_{\mathrm{Canti}}$

$$\bar{I}^{\mathrm{Kanal}}_{\mathrm{Canti}} = G U^{\,\mathrm{Kanal}} \int_{400\,\mathrm{nm}}^{700\,\mathrm{nm}} Q^{\mathrm{Kanal}}_\lambda \cdot I_\lambda \,\mathrm{d}\lambda. \tag{5.7}$$

Die wellenlängenabhängige Quanteneffizienz $Q^{\mathrm{Kanal}}_\lambda$ kann den Kenndaten der Kamera entnommen werden.[2,3] Die Emissivität von Silicium wird im betrachteten Temperatur- und Wellenlängenbereich als konstant angenommen [41–44].

Es gelte $C^{\,\mathrm{Kanal}} := AGU^{\,\mathrm{Kanal}}$. Durch Einsetzen von I_λ aus Gleichung 5.6 in Gleichung 5.7 und Überführung des Integrals in eine endliche Summe ergibt sich eine numerische Integration über die Stützstellen λ_i

$$\bar{I}^{\mathrm{Kanal}}_{\mathrm{Canti}}(T) = C^{\,\mathrm{Kanal}} \sum_i Q^{\mathrm{Kanal}}_{\lambda_i} \left[\frac{1}{\mathrm{e}^{\frac{hc}{\lambda_i k T}} - 1} \right] \frac{1}{\lambda_i^5} \Delta\lambda_i. \tag{5.8}$$

Bei bekanntem Wert für $C^{\,\mathrm{Kanal}}$ kann über Formel 5.8 die erwartete Farbkanal-Intensität des Cantilevers bei gegebener Temperatur numerisch berechnet werden.

Der Schmelzpunkt von Silicium liegt bei $T_{\mathrm{smp}} = 1687\,\mathrm{K}$ [4]. Ist $\bar{I}^{\mathrm{Kanal}}_{\mathrm{Canti}}(T_{\mathrm{smp}})$ am Schmelzpunkt auf Basis von experimentellen Daten bekannt, kann die Konstante $C^{\,\mathrm{Kanal}}$ für jeden Farbkanal durch Umstellen von Gleichung 5.8 berechnet werden

$$C^{\,\mathrm{Kanal}} = \bar{I}^{\mathrm{Kanal}}_{\mathrm{Canti}}(T_{\mathrm{smp}}) \left(\sum_i Q^{\mathrm{Kanal}}_{\lambda_i} \left[\frac{1}{\mathrm{e}^{\frac{hc}{\lambda_i k T_{\mathrm{smp}}}} - 1} \right] \frac{1}{\lambda_i^5} \Delta\lambda_i \right)^{-1}. \tag{5.9}$$

[2] Siehe Array `Q` in `signal_to_temp.py` auf Seite 163.

[3] Es ist zu beachten, dass mit $\bar{I}^{\mathrm{Kanal}}_{\mathrm{Canti}}$ keine spektralen Intensitäten, sondern mittlere BGR-Farbkanal-Intensitäten bezeichnet werden – dimensionslose Werte aus dem Intervall $[0, 255]$.

Der Cantilever begann auf der 45,0 V Heizstufe am Messspitzen-Ende (Kopfende) zu schmelzen. Das Einsetzen dieses Vorgangs ist mit einer spontanen Änderung des Emissionsverhaltens verbunden, welche in Abbildung 2.4 deutlich im hinteren Bereich der letzten Heizstufe zu erkennen ist. Nach dem Experiment wurden Rasterelektronenmikroskopie-Aufnahmen der Probe angefertigt, auf denen die Zone des Anschmelzens deutlich zu erkennen ist (siehe Kapitel 3). Die Bestimmung der mittleren Kanal-Intensitäten in dieser Schmelzzone kurz vor dem Phasenübergang lieferte die gesuchten Werte für $\bar{I}^{\text{Kanal}}_{\text{Canti}}(T_{\text{smp}})$ zur Bestimmung von C^{Kanal} über Gleichung 5.9. Hierfür wurde erneut die Histogramm-Funktion der Software GIMP verwendet. Die verwendete Selektion ist in Abbildung 5.5 gezeigt. Auf die Auswirkungen der konkreten Wahl des ausgewählten Bereichs und des verwendeten Einzelbilds wird im Abschnitt zur Fehlerbetrachtung eingegangen. Die gemessenen Intensitäten lauten

$$\begin{aligned}
\bar{I}^{\text{blau}}_{\text{Canti}}(T_{\text{smp}}) &= 12 \\
\bar{I}^{\text{grün}}_{\text{Canti}}(T_{\text{smp}}) &= 56 \\
\bar{I}^{\text{rot}}_{\text{Canti}}(T_{\text{smp}}) &= 236.
\end{aligned}$$

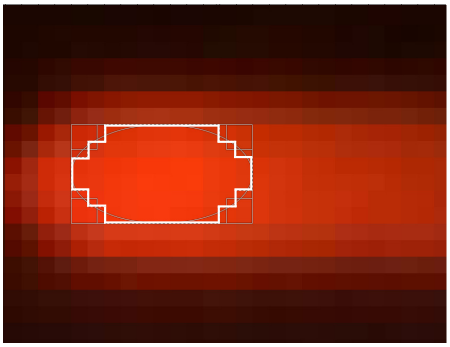


Abbildung 5.5: Kopfende des Cantilevers einige Sekunden vor dem Einsetzen des Schmelzvorgangs. Die weiß angedeutete Selektion markiert den Bereich der gleich entstehenden Schmelzzone und dient der Bestimmung von $\bar{I}^{\text{Kanal}}_{\text{Canti}}(T_{\text{smp}})$.

Die Auswertung von Gleichung 5.8 über den Temperaturbereich 1200–1687 K erfolgte mit Hilfe eines Python-Programms (Seite 163) und ist in Abbildung 5.6 dargestellt. Die abgebildeten Daten stehen in tabellarischer Form zur Verfügung, sodass bei gegebenem Wert für $\bar{I}^{\text{Kanal}}_{\text{Canti}}$ die zugehörige Temperatur abgelesen werden kann.

Streng genommen sind Gleichung 5.8 und deren Auftragung in Abbildung 5.6 nur bei uniformer Farbkanal-Intensität des Cantilevers gültig, d. h. $\bar{I}^{\text{Kanal}}_{\text{Canti}} = I^{\text{Kanal}}_{\text{Pixel}}$ für jeden Bildpunkt. Mit anderen Worten: es darf keinen Temperaturgradienten über den Körper der Probe geben. In diesem Fall kann aus der Farbkanal-Intensität

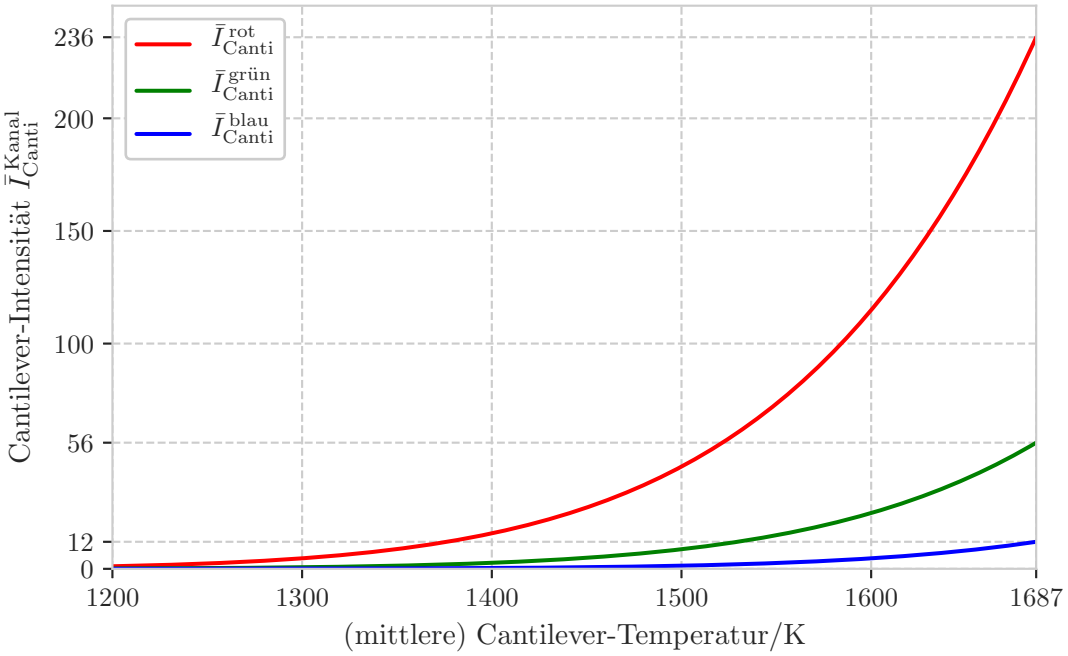


Abbildung 5.6: Cantilever-Intensität als Funktion der Temperatur. Anmerkungen zur Gültigkeit befinden sich im Text.

die Probentemperatur mit den Daten aus Abbildung 5.6 bestimmt werden. Allerdings zeigen Cantilever insbesondere bei hohen Heizleistungen oft hellere und dunklere Bereiche, zum Beispiel weil an einer Stelle der Kontakt der Probe zum Substrat verloren gegangen ist. Da es sich bei den berechneten Werten für $\bar{I}^{\text{Kanal}}_{\text{Canti}}$ um Mittelwerte handelt, kann auch nur die mittlere Temperatur der Probe angegeben werden. Nachfolgend sind alle pyrometrisch bestimmten Probentemperaturen als Mittelwerte zu verstehen. Diese Mittelwerte sind allerdings mit einem Fehler behaftet, da es sich bei der Temperaturabhängigkeit der Farbkanal-Intensität nicht um einen linearen Zusammenhang handelt. Das Problem und dessen Auswirkungen auf die nachfolgend angegebenen Temperaturen werden im Abschnitt zur Fehlerbetrachtung genauer erklärt.

5.4 Farbkanal-Vergleich

Aus Abbildung 5.6 und 5.1 ist ersichtlich, dass der rote Farbkanal die höchste Empfindlichkeit besitzt und im Laufe des Experiments den größten Wertebereich durchläuft. Daher wurde die Berechnung von Temperaturen auf Basis des roten Kanals durchgeführt. Trotzdem ist es sinnvoll, die Korrelation der Farbkanäle bei

hoher Heizleistung zu überprüfen. Liegt tatsächlich ein thermischer Effekt vor, dann werden ähnliche berechnete Temperaturdifferenzen erwartet. Eine Stichprobe wurde auf Basis des in Abbildung 5.7 gezeigten Intensitätssprungs durchgeführt. Die Auswertung ist in Tabelle 5.2 zusammengefasst.[4]

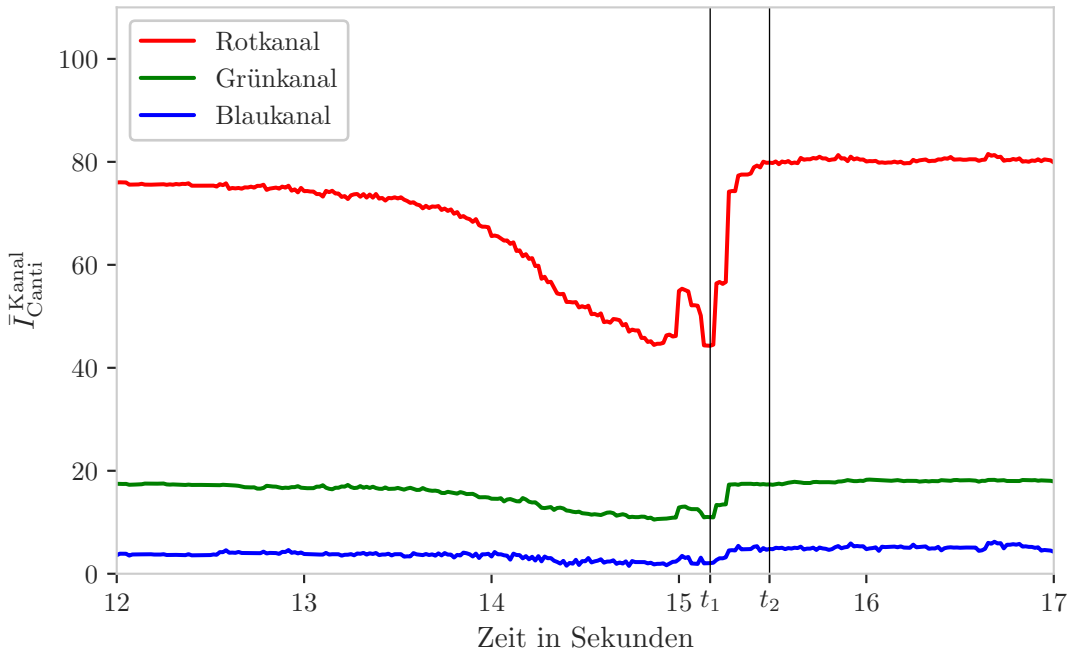


Abbildung 5.7: Für den Farbkanal-Vergleich gewählter Intensitätssprung. Die zugehörigen Temperaturen wurden für die Zeitpunkte t_1 und t_2 berechnet.

Tabelle 5.2: Daten in Bezug auf Abbildung 5.7.

Kanal	$I(t_1)$	$I(t_2)$	$T(t_1)/\mathrm{K}$	$T(t_2)/\mathrm{K}$	$\Delta T/\mathrm{K}$
rot	52,2	80,6	1514	1560	46
grün	11,7	17,6	1527	1566	39
blau	2,8	4,9	1557	1605	48

Es ergibt sich ein Mittelwert von $\overline{\Delta T} = 44{,}3\,\mathrm{K}$ mit einer Standardabweichung

[4] Die in Tabelle 5.2 aufgeführten Intensitätswerte können leicht von der Auftragung in Abbildung 5.7 abweichen. Erstere wurden zwecks höherer Genauigkeit durch manuelle Auswertung der Frames zu den Zeitpunkten t_1 und t_2 bestimmt, d. h. ohne die Fehlerquelle der Umfeld-Korrektur. Die Auswirkung der Umfeld-Korrektur wird im Abschnitt zur Fehlerbetrachtung diskutiert.

von 4,7 K, was etwa zehn Prozent des Mittelwerts entspricht. Die mittlere Abweichung vom Mittelwert beträgt etwa 3,6 K. Die recht gute Korrelation der Farbkanäle hinsichtlich berechneter Temperaturdifferenzen ist ein Indiz, dass es sich beim transienten Emissionsphänomen tatsächlich um einen thermischen Effekt handelt.

5.5 Fehlerbetrachtung

Bei der Berechnung von Temperaturwerten aus gemessenen Intensitäten müssen folgende Einflüsse und Fehlerquellen berücksichtigt werden

1. Wahl des Farbkanals
2. Verwendung der mittleren Cantilever-Intensität zur Temperaturbestimmung
3. Umfeld-Korrektur über Formel 5.2
4. Wahl des Kalibrierungs-Frames am Schmelzpunkt
5. Auswahl der Schmelzzone
6. Kalibrierungstemperatur
7. Abschätzung von Länge und Breite des Cantilevers in Pixeln.

Punkt 1: Die Temperaturbestimmung erfolgt auf Basis des Rotkanals. Aus bereits genannten Gründen wird erwartet, dass eine Mittelwertbildung über mehrere Farbkanäle ein ungenaueres Ergebnis liefert, als nur den empfindlichsten Kanal zu verwenden.

Punkt 2: Da es sich bei Gleichung 5.8 nicht um einen linearen Zusammenhang handelt, ist die Berechnung von mittleren Temperaturen aus mittleren Intensitäten fehlerbehaftet. Die Problematik wird anhand des folgenden Rechenbeispiels deutlich. Angenommen der Cantilever leuchtet im vorderen Viertel seiner Fläche besonders stark (Rotkanal-Intensität 230) und ist ansonsten dunkler (Rotkanal-Intensität 100). Die Situation ist in Abbildung 5.8 gezeigt.

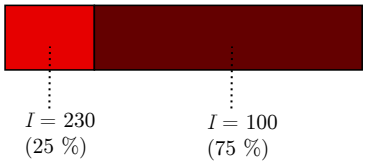


Abbildung 5.8: Cantilever mit inhomogener Intensitätsverteilung.

Für die mittlere Cantilever-Temperatur gilt

$$\bar{T}_{\mathrm{Canti}} = 0{,}25 \cdot T(I^{\mathrm{rot}}_{\mathrm{Canti}} = 230) + 0{,}75 \cdot T(I^{\mathrm{rot}}_{\mathrm{Canti}} = 100). \tag{5.10}$$

Einsetzen der in Abbildung 5.6 dargestellten Daten liefert $\bar{T}_{\mathrm{Canti}} = 0{,}25 \cdot 1684\,\mathrm{K} + 0{,}75 \cdot 1584\,\mathrm{K} = 1609\,\mathrm{K}$. Bei der Auswertung der experimentellen Daten erfolgt die Berechnung von $\bar{T}_{\mathrm{Canti}}$ allerdings über die mittlere Cantilever-Intensität mittels

$$\bar{T}_{\mathrm{Canti}} = T(\bar{I}^{\mathrm{rot}}_{\mathrm{Canti}}). \tag{5.11}$$

Im Rahmen des Beispiels ergeben sich $\bar{I}^{\mathrm{rot}}_{\mathrm{Canti}} = 0{,}25 \cdot 230 + 0{,}75 \cdot 100 = 132{,}5$ und $\bar{T}_{\mathrm{Canti}} = 1617\,\mathrm{K}$, also ein Fehler von $8\,\mathrm{K}$. Da es sich hier um ein Extrembeispiel handelt wird der Fehler durch die Verwendung der mittleren Cantilever-Intensität auf $\pm 5\,\mathrm{K}$ geschätzt. Absolute Temperaturangaben beziehen sich im Folgenden immer auf $\bar{T}_{\mathrm{Canti}}$ nach Gleichung 5.11.

Punkt 3: Die Anwendung von Formel 5.2 zur Bestimmung mittlerer Cantilever-Intensitäten führt üblicherweise zu einer Abweichung von bis zu $\pm 10\,\mathrm{K}$ für $\bar{T}_{\mathrm{Canti}}$ gegenüber der manuellen Auswertung einzelner Frames.

Punkt 4: Die Wahl des zur Temperatur-Kalibrierung verwendeten Frames wirkt sich mit einer Unsicherheit von bis zu etwa $\pm 10\,\mathrm{K}$ auf die berechneten mittleren Cantilever-Temperaturen aus.

Punkt 5: Die Wahl der Selektion für die Schmelzzone (siehe Abbildung 5.5) wirkt sich bezüglich der in Frage kommenden Bereiche mit einer Unsicherheit von bis zu etwa $\pm 10\,\mathrm{K}$ auf die berechneten mittleren Cantilever-Temperaturen aus.

Punkt 6: Es ist möglich, dass die Temperatur in der selektierten Schmelzzone nicht exakt $1687\,\mathrm{K}$ beträgt. Dieser Umstand wird vorsichtshalber mit einer Unsicherheit von $\pm 10\,\mathrm{K}$ bedacht.

Punkt 7: Die Variation des Cantilever-Bereichs (siehe gestrichelter Umriss in Abbildung 5.4) im Rahmen in Frage kommender Werte wirkt sich mit einer Unsicherheit von bis zu etwa $\pm 10\,\mathrm{K}$ auf die berechneten Temperaturen aus.

Die Abschätzung der in Punkt 2 bis 7 aufgeführten Fehlerquellen erfolgte relativ pessimistisch. Durch Addition ergeben sich $\pm 55\,\mathrm{K}$. Vorsichtshalber werden absolute Temperaturen im Folgenden mit einer Unsicherheit von $\pm 60\,\mathrm{K}$ angegeben. Temperaturdifferenzen können genauer bestimmt werden. Der Fehler durch die Umfeld-Korrektur (Punkt 2) kann eliminiert werden, indem für einen Temperatursprung jeweils ein Frame vor dem Sprung und ein Frame nach dem Sprung

durch manuelle Selektion des Cantilevers (ohne Anwendung von Formel 5.2) ausgewertet werden. Auf diese Weise bestimmte Temperaturdifferenzen sind sehr robust gegenüber der Variation der übrigen in diesem Abschnitt diskutierten Parameter und werden mit einer Unsicherheit von $\pm 5\,\mathrm{K}$ angegeben.

5.6 Zuordnung von Temperaturen

Die Zuordnung von pyrometrisch berechneten Temperaturen zu experimentell bestimmten Probenintensitäten ist in den Abbildungen 5.9 und 5.10 gezeigt. Abbildung 5.9 ist eine Übersichtsdarstellung für vier verschiedene Heizleistungen. Bei Abbildung 5.10 handelt es sich um eine Detaildarstellung eines Ausschnitts der bei 42,5 V gemessenen Intensitäten. Die Probentemperatur sinkt typischerweise über einen Zeitraum von mehreren Sekunden um etwa 40–80 K und kehrt dann im Zuge einer oder mehrerer schneller Sprünge auf die Ausgangstemperatur zurück.

Unterbrechungen in der Auftragung zeigen Temperatursprünge, die so schnell sind, dass sie nicht vollständig mit einer Sampelrate von 60 Bildern pro Sekunde aufgelöst werden können. Besonders auffällige Sprünge können beim Übergang von niedrigeren zu höheren Temperaturen bei hohen Heizleistungen beobachtet werden. Für vier dieser Sprünge wurde exemplarisch die Temperaturdifferenz durch manuelle Auswertung jeweils eines Einzelbildes vor und nach dem Ereignis bestimmt (Abbildung 5.10). Bei der manuellen Auswertung von einzelnen Frames entsteht kein Fehler durch die in Abschnitt 5.2 beschriebene Umfeld-Korrektur, da die Zone des Cantilevers von Hand selektiert wird.

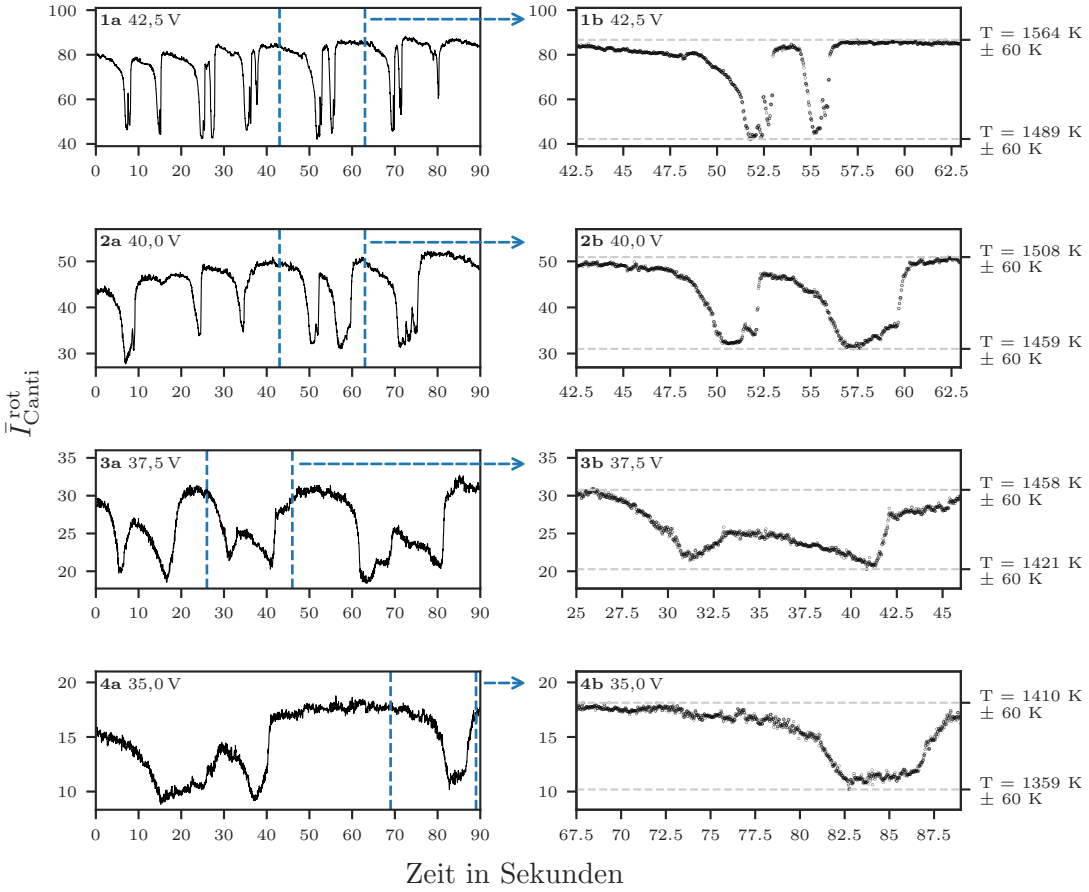


Abbildung 5.9: Durchschnittliche Rotkanal-Intensität bei vier verschiedenen Heizleistungen und pyrometrisch bestimmte Probentemperaturen. Die Abbildungen in der linken Spalte (**1a**-**4a**) zeigen die gemessene Intensität jeweils über einen Zeitraum von 90 s ab einem auf 0 s gesetzten Startzeitpunkt. Die blauen, gestrichelten Linien markieren Abschnitte, die mit höherer zeitlicher Auflösung in der rechten Spalte (**1b**-**4b**) dargestellt sind. Rechts sind pyrometrisch bestimmte Absoluttemperaturen vermerkt. Datenpunkte in der rechten Spalte sind nicht verbunden, sodass Unterbrechnungen in der Auftragung Intensitätssprünge kennzeichnen, die mit einer Bildrate von 60 Hz nicht vollständig aufgelöst werden können. Besonders auffällige Sprünge zeigen sich in Teilbild **1b**. Auf diese Sprünge wird im Detail in Abbildung 5.10 eingegangen. Die Daten wurden in ähnlicher Darstellung bereits veröffentlicht [28].

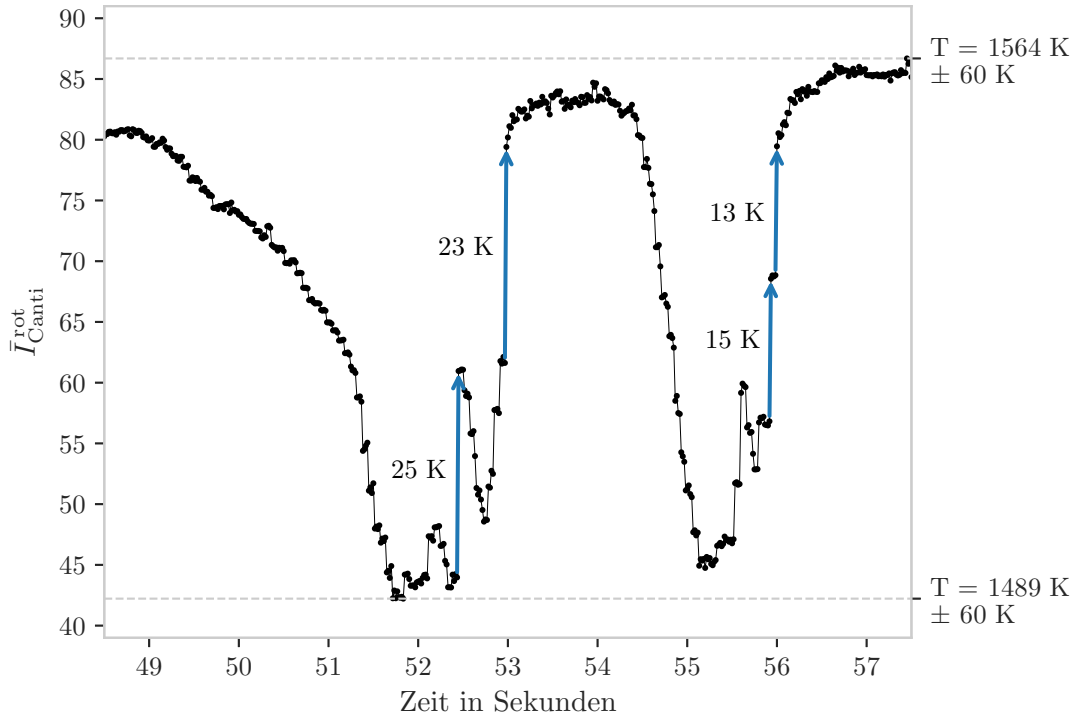


Abbildung 5.10: Daten aus Teilbild **1b** von Abbildung 5.9. Jeder schwarze Punkt repräsentiert eine aus den Messdaten berechnete Probenintensität (zeitliche Auflösung 60 Hz). Die blauen Pfeile kennzeichnen vier auffällige Intensitätssprünge. Die Unsicherheit der zugehörigen Temperaturdifferenzen ist ±5 K. Die Daten wurden in ähnlicher Darstellung bereits veröffentlicht [28].

6 Reaktionsmodell

Einige Aspekte der nachfolgenden Überlegungen wurden in knapper Form bereits in der Publikation „Transient Light Emission from the Silicothermic Reduction of Magnesium Oxide with Potential for Monitoring Intermediate Compound Formation and Decay“ diskutiert [28].

Die gemessene Zusammensetzung von etwa 2/1/5 (Mg/Si/O) der Schicht mit Reaktionsprodukten in der Ätzgrube liegt nahe bei der für eine Mg_2SiO_4-Schicht erwarteten Zusammensetzung von 2/1/4 (siehe Abbildung 3.10) in Kapitel 3). Es ist möglich, dass sich nach dem Experiment Luftsauerstoff in der fein strukturierten Oberfläche anlagert. Von Alphei u. a. wurde eine Oberflächenzusammensetzung von 4/1/8 angegeben. Hierbei handelte es sich allerdings um Werte aus in situ AES-Messungen von Bereichen des MgO-Substrats, die kurz zuvor von flüssigem Silicium benetzt wurden [29].

Es ist bekannt, dass die in der Einleitung im Rahmen des Pidgeon-Prozesses betrachtete Modellreaktion

$$2\,\mathrm{MgO\,(s)} + \mathrm{Si\,(s)} \longrightarrow \mathrm{SiO_2(s)} + 2\,\mathrm{Mg(g)} \tag{6.1}$$

die Umsetzung von Magnesiumoxid mit Silicium in der Praxis unzureichend beschreibt. Es kann festes Magnesiumsilikat (Mg_2SiO_4) entstehen, als Produkt der Teilreaktion

$$4\,\mathrm{MgO\,(s)} + \mathrm{Si\,(s)} \longrightarrow 2\,\mathrm{Mg\,(g)} + \mathrm{Mg_2SiO_4(s)}. \tag{6.2}$$

Tatsächlich spielt auch die Bildung von gasförmigem SiO eine Rolle. Im Rahmen von thermogravimetrischen Analysen beobachteten Toguri und Pidgeon einen zusätzlichen Gewichtsverlust bei den Edukten, der bei etwa 1573 K einsetzt und nicht vollständig mit Gleichung 6.1 und 6.2 erklärt werden kann [45]. Die Autoren begründen dies mit der Bildung von $\mathrm{SiO\,(g)}$ über die Teilreaktion

$$\mathrm{MgO\,(s)} + \mathrm{Si\,(s)} \longrightarrow \mathrm{Mg\,(g)} + \mathrm{SiO(g)} \tag{6.3}$$

und es ergibt sich eine mithilfe eines Parameters $n \geq 0$ als

$$\mathrm{MgO\,(s)} + \frac{n+1}{4n+1}\mathrm{Si\,(s)} \longrightarrow \frac{n}{4n+1}\mathrm{Mg_2SiO_4(s)} + \frac{2n+1}{4n+1}\mathrm{Mg\,(g)} + \frac{1}{4n+1}\mathrm{SiO\,(g)} \tag{6.4}$$

formulierbare Gesamtreaktion [45]. Für $n = 0$ geht Gleichung 6.4 in Reaktion 6.3 über, im Grenzfall $n \to \infty$ ergibt sich Reaktion 6.2. Der Parameter n steht im Zusammenhang mit dem Molenbruch $x_{\mathrm{SiO}} = \frac{1}{2n+2}$ von SiO in der Gasphase. Eine nicht mehr vernachlässigbare Bildung von SiO setzt in Pidgeons Experimenten ab einer Temperatur von 1573 K ein. Das theoretische Maximum von x_{SiO} beträgt $\frac{1}{2}$.

Die pyrometrisch bestimmte Temperatur von 1564 K für die höchste Heizstufe (Abbildung 5.9) liegt knapp unter der Temperatur, ab welcher die Freisetzung von SiO von Pidgeon thermogravimetrisch nachgewiesen werden konnte. Es ist möglich, dass die Entstehung geringer Mengen des Gases bereits bei niedrigeren Temperaturen einsetzt, und transienten Charakter besitzt. Eine vorübergehende, endotherme Bildung von gasförmigem SiO könnte die Probe kühlen und die wiederkehrenden Dunkelphasen erklären.

Des Weiteren ist es möglich, dass Reaktion 6.3 durch eine dünne Schicht von Magnesiumsilikat gehemmt wird, die sich zwischen Probe und Substrat bildet. Diese Schicht wäre erwartungsweise deutlich dünner als die durch Temperexperimente erzeugte 250 nm dicke Schicht in Abbildung 3.16. Das Silikat könnte in Folge der Reaktion

$$\mathrm{Mg_2SiO_4(s)} + 3\,\mathrm{Si(s)} \longrightarrow 2\,\mathrm{Mg(g)} + 4\,\mathrm{SiO(g)} \tag{6.5}$$

wieder endotherm zerfallen und Reaktion 6.3 erneut ermöglichen, da der Si-MgO-Kontakt wiederhergestellt wird. Die nächste Bildung der Silikatschicht würde die Gasentwicklung erneut hemmen, sodass die Probentemperur wieder steigt. Eine Skizze zu dieser Überlegung ist in Abbildung 6.1 gezeigt.

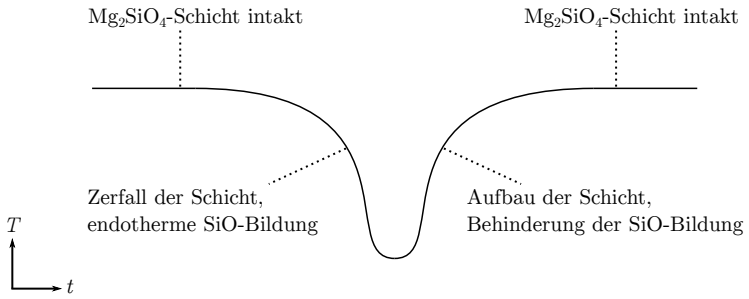


Abbildung 6.1: Modellvorstellung eines Temperatureinbruchs in Folge des Abbaus einer Zwischenschicht.

Es ist zu beachten, dass die in der Grenzfläche ablaufenden Prozesse wahrscheinlich komplexer als diese einfache Modellvorstellung sind. Details zum Reaktionsmechanismus sind unklar. Inbesondere erklärt Abbildung 6.1 nicht, warum Sprünge in Richtung höherer Temperaturen so schnell ablaufen können, wie in Abbildung 5.10 gezeigt.

Es ist natürlich interessant, die Dynamik der Zerfalls und Aufbaus der Reaktionsschicht genauer zu untersuchen. Dazu wurden die Reaktionsraten über die zugehörigen Frequenzen abgeschätzt und dann in Form eines Arrhenius-Plots aufgetragen (Abbildung 6.2).

Das Einsetzen eines Temperatureinbruchs wird mit dem Zerfall (Lochbildung) der Reaktionsschicht in Zusammenhang gebracht. Für die Frequenz f_{Effekt} der Temperatureinbrüche kann nach Arrhenius der Zusammenhang

$$f_{\text{Effekt}} = A\mathrm{e}^{-\frac{E_A^L}{RT}} \tag{6.6}$$

angesetzt werden. Mit E_A^L wird die effektive Aktivierungsenergie für die Bildung von Löchern in der Reaktionsschicht bezeichnet. Der Wert kann mit Hilfe der Steigung der Ausgleichsgeraden für die Auftragung von $\ln(f_{\text{Effekt}}/\text{Hz})$ gegen T^{-1} abgeschätzt werden (Abbildung 6.2 **a**). Die Frequenzen wurden durch Zählen der Ereignisse in Zeitbereichen mit typischem Verhalten bestimmt. Die lineare Regression liefert die Werte

$$-\frac{E_A^L}{R} \approx -33\,544\,\text{K}$$

als Steigung und

$$\ln(A/\text{Hz}) \approx 20{,}3$$

als Ordinatenachsenabschnitt. Durch Umstellen nach der Aktivierungsenergie wird der Wert $E_A^L \approx 279\,\text{kJ}\,\text{mol}^{-1}$ erhalten.

In analoger Weise wird die Schichtbildung mit der Dauer der Temperatureinbrüche über den Ansatz

$$\frac{1}{t_{\text{Effekt}}} = A\mathrm{e}^{-\frac{E_A^S}{RT}} \tag{6.7}$$

in Zusammenhang gebracht (Abbildung 6.2 **b**). Die Zeit t_{Effekt} beschreibt die temperaturabhängige Dauer eines Emissionseffekts. Der Wert wurde auf halber Tiefe der Temperatureinbrüche bestimmt. Es handelt sich also um die Halbwertsbreite typischer Einbrüche gemäß Abbildung 6.1. Die effektive Aktivierungsenergie der Schichtbildung ist mit E_A^S bezeichnet. Die lineare Regression liefert die Steigung

$$-\frac{E_A^S}{R} \approx -33\,154\,\text{K}$$

und den Ordinatenachsenabschnitt

$$\ln(A/\text{Hz}) \approx 21{,}7.$$

Damit ergibt sich durch Umstellen $E_A^S \approx 276\,\text{kJ}\,\text{mol}^{-1}$.

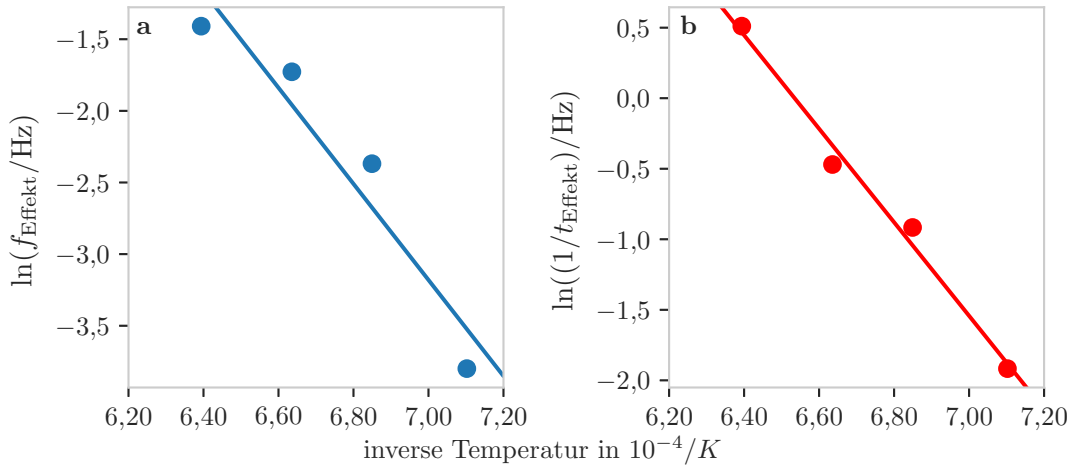


Abbildung 6.2: Arrhenius-Auftragungen. **a** Logarithmische Auftragung der Frequenz f_{Effekt} des Emissionseffekts gegen den Kehrwert der Probentemperatur und lineare Regression. **b** Analoge Auftragung für den Kehrwert der Dauer der Temperatureinbrüche t_{Effekt} (bestimmt auf halber Tiefe der Einbrüche), ebenfalls mit Ausgleichsgerade. Die Ergebnisse der linearen Regression werden im Text diskutiert.

Anhang und Details

Magnesium-Primärproduktion

Eine komplette Übersicht der Daten findet sich in Tabelle 6.1. Zu den Produktionszahlen gelten die folgenden Anmerkungen.

- Einige Werte basieren auf ungeprüften Angaben der Länder oder Schätzungen. Ein Produktionswert von Null muss nicht bedeuten, dass überhaupt kein Magnesium produziert wurde – allerdings wahrscheinlich keine für den Weltmarkt relevanten Mengen.
- Die Schätzung der Weltproduktion seitens des USGS berechnet sich bis auf kleinere Abweichungen aus der Summe der Produktionszahlen der geführten Länder. Daher wurde die Produktion für die Länder, die nur in einigen Jahren geführt werden, für alle anderen Jahre auf Null gesetzt.
- Die Primärproduktions-Daten für 1992 bis 2016 wurden aus den Jahrbüchern (Mineral Yearbook) des U. S. Geological Surveys zusammengestellt [19]. Aufgrund von Unstimmigkeiten und späteren Korrekturen, wurde jeweils das *aktuellste* Jahrbuch verwendet, welches noch Zahlen für ein betreffendes Land und Jahr führt.
- Die Daten von 2017 und 2018 stammen aus dem USGS Mineral Commodity Summary von 2019.

Außerdem gelten folgende länderspezifische Besonderheiten.

- USA: Seit 1999 werden die Produktionszahlen zurückgehalten. Ab diesem Zeitpunkt fließt in den USA produziertes Magnesium nicht mehr in den Schätzwert für die Weltproduktion mit ein.
- Russland: Für 1992 – 2010 ist die Sekundärproduktion in den Zahlen enthalten. Für 2011 – 2016 ist für die Titanproduktion verbrauchtes Magnesium enthalten. Es ist unklar, ob dies auch für die Zahlen von 2017 und 2018 gilt.
- Kanada: Für 1994 – 2008 ist die Sekundärproduktion in den Zahlen enthalten.

- Serbien: Bis Juni 2005 gelten die Zahlen für Serbien und Montenegro. Für 2010 ist die Sekundärproduktion in den Zahlen enthalten.
- Kasachstan: Für 2011 – 2016 ist für die Titanproduktion verbrauchtes Magnesium enthalten. Es ist unklar, ob dies auch für die Zahlen von 2017 und 2018 gilt.

Tabelle 6.1: Weltweite Magnesium-Primärproduktion in Kilotonnen [19].

Jahr	Brasilien	China	Frankreich	Iran	Israel	Italien	Japan	Kanada	Kasachstan	Norwegen	Russland	Serbien	Südkorea	Türkei	USA	Ukraine	Welt[e]
1992	7	11	14	0	0	1	7	26	3	30	40[s]	4	0	0	137	15	295
1993	10	12	11	0	0	0	7	23	2	27	30[s]	0	0	0	132	15	269
1994	10	24	12	0	0	0	3	29[s]	0	28	35[s]	0	0	0	128	12	282
1995	10	94	14	0	0	0	0	48[s]	9	28	38[s]	3	0	0	142	10	395
1996	9	73	14	0	0	0	0	54[s]	9	38	35[s]	3	0	0	133	10	378
1997	9	76	14	0	7	0	0	58[s]	9	34	40[s]	3	0	0	125	10	384
1998	9	71	14	0	25	0	0	77[s]	9	35	42[s]	4	0	0	106	5	396
1999	8	120	16	0	25	0	0	74[s]	11	41	45[s]	1	0	0	–[z]	3	341
2000	6	190	17	0	32	0	0	80[s]	10	41	45[s]	1	0	0	–[z]	3	422
2001	6	200	4	0	34	0	0	83[s]	16	36	40[s]	2	0	0	–[z]	3	420
2002	6	250	0	0	26	0	0	80[s]	18	10	40[s]	2	0	0	–[z]	3	432
2003	6	340	0	0	26	0	0	78[s]	14	0	43[s]	2	0	0	–[z]	3	509
2004	6	442	0	0	28	0	0	54[s]	18	0	45[s]	2	0	0	–[z]	3	595
2005	6	470	0	0	28	0	0	50[s]	20	0	45[s]	2	0	0	–[z]	2	622
2006	6	520	0	0	25	0	0	65[s]	21	0	35[s]	2	0	0	–[z]	2	675
2007	18	625	0	0	30	0	0	16[s]	21	0	37[s]	2	0	0	–[z]	3	751
2008	15	559	0	0	32	0	0	2[s]	21	0	37[s]	2	0	0	–[z]	2	670
2009	16	501	0	0	19	0	0	0	21	0	29[s]	–[n]	0	0	–[z]	2	588
2010	16	654	0	0	23	0	0	0	15	0	21[s]	1[s]	0	0	–[z]	7	737
2011	16	675	0	0	26	0	0	0	21[t]	0	58[t]	0	0	0	–[z]	9	806
2012	16	698	0	0	27	0	0	0	22[t]	0	65[t]	0	3	0	–[z]	9	840
2013	16	770	0	0	27	0	0	0	13[t]	0	66[t]	0	8	0	–[z]	10	910
2014	16	874	0	0	26	0	0	0	10[t]	0	62[t]	0	10	0	–[z]	7	1000
2015	15	859	0	0	19	0	0	0	8[t]	0	60[t]	0	10	0	–[z]	8	979
2016	16	871	0	0	23	0	0	0	10[t]	0	58[t]	0	10	5	–[z]	5	998
2017	15	930	0	3	23	0	0	0	9	0	40	0	10	14	–[z]	8	1050
2018	15	800	0	5	25	0	0	0	23	0	65	0	10	10	–[z]	19	970

Auf Kiltonnen gerundet. [e] Schätzung, kann von Zeilensumme abweichen. [n] Nicht verfügbar.
[s] Enthält sekundäres Mg. [t] Enthält Verbrauch für Ti-Produktion. [z] Zurückgehalten.

Programmcode

Die verwendete Programmiersprache ist Python in der Version 2.7. Es werden außerdem die Pakete `matplotlib` (ab Version 2.2.4), `opencv_python` (ab Version 4.1.1.26), `numpy` (ab Version 1.13.3) und `Pillow` (ab Version 6.2.1) benötigt. Die Lizenzen der Drittpartei-Bibliotheken sind bei Verwendung zu beachten.

Alle Programme sind voll funktionsfähig. Segmente, welche sinnvoll weiterentwickelt oder optimiert werden können, sind mit Kommentaren (`# TODO ...`) gekennzeichnet. Wichtige Hinweise zur Implementierung beginnen mit `# NOTE`.

Paket roitools

Das Paket `roitools` umfasst die Module `vidtools.py`, `roitools.py`, `roi_planner_gui.py` und `examples.py` in folgender Ordnerstruktur.

```
roitools/
|-- examples
| |-- examples.py
| |-- __init__.py
|-- __init__.py
|-- roi_planner_gui.py
|-- roitools.py
|-- vidtools.py
```

Zusätzlich sind Hilfs- und Metadateien wie Tests, Installationsanweisungen und Dokumentation enthalten. Das komplette Projekt auf dem aktuellen Stand ist unter `https://github.com/timgeb/roitools` verfügbar.

Das Modul `vidtools.py` implementiert die Klasse `PyCap`, eine Datenstruktur zum Auslesen der Einzelbilder einer Videodatei.

vidtools.py

```
1  'pythonic facade for some cv2 functionality'
2
3  # TODO port to Python 3 ...
4
5  from collections import namedtuple, OrderedDict
6  import os
7
8  import cv2
9  import numpy as np
10
```

```
11 # gather capture properties defined by cv2
12 # and create mapping property name -> property id
13 # (uses OrderedDict for consistent output from Frame.info)
14 _cv2version = int(cv2.__version__[0])
15
16 if _cv2version <= 2:
17     _prop_prefix = 'CV_CAP_PROP_'
18     _consts_loc = cv2.cv
19 else:
20     _prop_prefix = 'CAP_PROP_'
21     _consts_loc = cv2
22
23 _prop_names = sorted(
24     x for x in _consts_loc.__dict__ if x.startswith(_prop_prefix))
25
26 _prop2id = OrderedDict(
27     (name[len(_prop_prefix):].lower(), getattr(_consts_loc, name))
28     for name in _prop_names)
29
30 # gather events defined by cv2
31 # and create mapping name -> event id
32 _ev_prefix = 'EVENT_'
33 _ev_names = [x for x in cv2.__dict__ if x.startswith(_ev_prefix)]
34 _ev2id = {ev[len(_ev_prefix):].lower() : getattr(cv2, ev) for ev in _ev_names}
35 _id2ev = {id_:event for event, id_ in _ev2id.iteritems()}
36
37 # module level functions
38 imwrite = cv2.imwrite
39
40 def wait(delay):
41     'wait delay milliseconds, to be used after showing a frame'
42
43     # cv2.waitKey with a delay <= 0 pauses, we suppress this behavior
44     # and have a pause function instead
45     if delay > 0:
46         cv2.waitKey(delay)
47
48 def pause():
49     'wait indefinitely, to be used after showing a frame'
50     cv2.waitKey(0)
51
52 class PyCap(object):
53     '''Python adapter/facade for cv2.VideoCapture
```

```
54
55      - instance attributes can be accessed/set directly via the dot notation,
56      e.g. cap.pos_msec = 1000 instead of cap.set(cv2.CAP_PROP_POS_MSEC, 1000)
57
58      - instances are iterable
59
60      - iteration yields Frame objects (numpy arrays with extra methods)'''
61
62      def __init__(self, video, title=None):
63          'PyCap(path) -> PyCap object'
64          if not os.path.isfile(video):
65              raise IOError("file not found: '{}'".format(video))
66
67          self._cv2cap = cv2.VideoCapture(video)
68          self.video = video
69          self.title = str(video) if title is None else title
70          self.open()
71
72      def open(self):
73          'open the capture if not already opened'
74          if not self._cv2cap.isOpened():
75              self._cv2cap.open(self.video)
76
77      def __getattr__(self, attr):
78          # allow accessing capture properties with dot notation, e.g.
79          # cap.buffersize instead of cap.get(cv2.cv.CV_CAP_PROP_BUFFERSIZE)
80          if hasattr(self, '_cv2cap') and attr in _prop2id:
81              return self._cv2cap.get(_prop2id[attr])
82
83          msg = "'{}' has no attribute '{}'".format(type(self).__name__, attr)
84          raise AttributeError(msg)
85
86      def __setattr__(self, attr, value):
87          # allow setting capture properties with dot notation, e.g.
88          # cap.pos_msec = 2000 vs cap.set(cv2.cv.CV_CAP_PROP_POS_MSEC, 2000)
89          if hasattr(self, '_cv2cap') and attr in _prop2id:
90              success = self._cv2cap.set(_prop2id[attr], value)
91
92              # VideoCapture.set returns False if the property cannot be set
93              if not success:
94                  raise AttributeError('read only or unknown property')
95          else:
96              super(PyCap, self).__setattr__(attr, value)
```

```
97
98      def info(self):
99          'return video information as string'
100         template = '{{: <{}}} {{}}\n'.format(len(max(_prop2id, key=len)))
101         info = ''.join(
102             [template.format(prop, getattr(self, prop)) for prop in _prop2id])
103
104         return info + template.format(
105             'len/s', self.frame_count/(self.fps or np.nan))
106
107     def __iter__(self):
108         return self
109
110     def next(self):
111         'get next frame'
112         success, array = self._cv2cap.read()
113         if not success:
114             raise StopIteration
115         return Frame(array, self.title)
116
117     def release(self):
118         self._cv2cap.release()
119
120     def __enter__(self):
121         self.open()
122         return self
123
124     def __exit__(self, exc_type, exc_val, exc_tb):
125         self.release() # 'rewind and close'
126         self.open()
127
128     def play(self, delay=None, start=None, stop=None, rewind=False):
129         '''play the video from its current position
130
131         delay: wait delay milliseconds after showing a frame;
132         if delay is None, waits int(ceil(1000.0/fps)) ms if fps
133         is available, else waits 1 ms
134
135         start: play from start pos_msec or from current position
136         if start is None
137
138         stop: play to stop pos_msec or play to last frame and rewind
139         if stop is None
```

```
140
141         rewind: if True, release and reopen capture if played to end'''
142         if delay is None:
143             has_fps = not (np.isnan(self.fps) or self.fps == 0)
144             delay = int(np.ceil(1000.0/self.fps)) if has_fps else 1
145
146         if start is not None:
147             self.pos_msec = start
148
149         if stop is None:
150             stop = np.nan
151
152         self.open()
153
154         for frame in self:
155             # if the video lies about its true fps, i.e. fps has been upscaled
156             # by repeat-last-frame instructions, the self.pos_msec > stop
157             # check will overshoot the destination time
158             if self.pos_msec > stop:
159                 break
160
161             if delay > 0:
162                 frame.show()
163                 wait(delay)
164         else:
165             # played to end -> "rewind and reopen"
166             if rewind:
167                 self._cv2cap.release()
168                 self.open()
169
170 # classes for self-documenting drawing
171 Point = namedtuple('Point', 'x y')
172 Color = namedtuple('Color', 'blue green red') # BGR!
173
174 # for the intricacies of sublassing ndarray, see
175 # http://docs.scipy.org/doc/numpy/user/basics.subclassing.html
176 class Frame(np.ndarray):
177     # TODO: does currently not play nice with some ufuncs,
178     # e.g. np.copy(Frame object) returns ndarray instance
179     # and converting to masked array masks properties and methods
180     # -> needs __array_wrap__ implemented
181     '''Frame(array, title) -> Frame object
182
```

```
183     encapsulates common frame operations, e.g. frame.show() instead of
184     cv2.imshow(title, frame) or frame.blue instead of frame[:, :, 0]'''
185
186     def __new__(cls, input_array, title='unnamed'):
187         '''Frame(numpy array, title) -> Frame
188
189         array must be two-dimensional (grayscale image) or three-dimensional
190         with array.shape[2] == 3 (bgr image) and of type np.uint8'''
191         # view calls ndarray.__new__ which calls Frame.__array_finalize__
192         # with input_array as the second parameter (from_array)
193         self = input_array.view(cls)
194
195         # set title in case Frame was instantiated explicitly through
196         # the constructor, override the value just set in __array_finalize__
197         self.title = title
198         return self
199
200     def __array_finalize__(self, from_array):
201         # from_array: template array (e.g. from slicing) or viewed array
202         #
203         # if from_array is None, for some reason ndarray.__new__(Frame,
204         # ((10,))) or similar got called, i.e. __new__ on ndarray was invoked
205         # directly in order to instantiate a Frame; I don't know why
206         # this would happen
207         #
208         # when instantiating a Frame directly (i.e. Frame.__new__ is
209         # called) or constructing through a view or template, from_array cannot
210         # be None
211         if from_array is not None:
212             # when constructing a Frame directly, the fallback getattr value
213             # will be overriden once __array_finalize__ returns to __new__,
214             # without a fallback value getattr would raise an AttributeError
215             # in this scenario
216             #
217             # when constructing a Frame from view casting of a non-Frame
218             # e.g. arange(3).view(Frame), Frame.__new__ is never called
219             # so the fallback value from this method will be used
220             #
221             # if creating a new instance/view from an existing Frame (i.e.
222             # title is set) we just forward the value to the new instance
223             self.title = getattr(from_array, 'title', 'unnamed')
224
225     def adjust_brightness(self, offset):
```

```
226         '''brighten or darken the frame in-place by integer offset
227         offset > 0: brightness up
228         offset < 0: brightness down'''
229 
230         if offset != 0:
231             h, s, v = cv2.split(cv2.cvtColor(self, cv2.COLOR_BGR2HSV, self))
232             cv2.add(v, offset, v)
233             cv2.cvtColor(cv2.merge((h, s, v)), cv2.COLOR_HSV2BGR, self)
234 
235     def show(self):
236         '''shows the frame
237 
238         use module level function wait(delay) after call to show to
239         slow playback when iterating over a sequence of frames'''
240         cv2.imshow(self.title, self)
241 
242     def draw_circle(self, center_xy, radius, color=(0, 255, 0), thickness=1,
243         linetype=8, shift=0):
244         '''draw_circle((x, y), radius[, (blue, green, red)[, thickness[,
245         linetype[, shift]]]]) -> None
246 
247         draws simple circle on the frame
248         set thickness=-1 to fill circle
249         see cv2.line documentation for optional linetype and shift arguments'''
250         cv2.circle(self, center_xy, radius, color, thickness, linetype, shift)
251 
252     def draw_rectangle(self, vertex1, vertex2, color=(0, 255, 0),
253         thickness=1, linetype=8, shift=0):
254         '''draw_rectangle((x1, y1), (x2, y2)[, (blue, green, red)[, thickness[,
255         linetype[, shift]]]]) -> None
256 
257         draws simple rectangle on the frame
258         vertex2 is the vertex of the rectangle opposite to vertex1
259         set thickness=-1 to fill rectangle
260         see cv2.line documentation for optional linetype and shift arguments'''
261         cv2.rectangle(self, vertex1, vertex2, color, thickness,
262             linetype, shift)
263 
264     def save(self, filename):
265         '''saves image, returns True on success
266         examples: frame.save('screenshot.png') -> True
267                   frame.save('/home/tim/screenshot.bmp') -> True'''
268         return cv2.imwrite(filename, self)
```

```
269
270     # TODO: documentation for other drawing methods, make *args explicit
271     def put_text(self, *args):
272         cv2.putText(self, *args)
273
274     def draw_line(self, *args):
275         cv2.line(self, *args)
276
277     def draw_ellipse(self, *args):
278         cv2.ellipse(self, *args)
279
280     def draw_polygon(self, *args):
281         cv2.polygon(self, *args)
282
283     @property
284     def blue(self):
285         'blue channel data'
286         return self[:, :, 0]
287
288     @blue.setter
289     def blue(self, blue):
290         self[:, :, 0] = blue
291
292     @property
293     def green(self):
294         'green channel data'
295         return self[:, :, 1]
296
297     @green.setter
298     def green(self, green):
299         self[:, :, 1] = green
300
301     @property
302     def red(self):
303         'red channel data'
304         return self[:, :, 2]
305
306     @red.setter
307     def red(self, red):
308         self[:, :, 2] = red
```

Das Modul `roitools.py` implementiert die Klasse `RoiCap`, eine Video-Datenstruktur, welcher Beobachtungszonen hinzugefügt werden können. Außerdem enthält das Modul Klassen für verschiedene Arten von Beobachtunszonen.

roitools.py

```
'tools for creating and analyzing regions of interest (ROIs) in a video'

# TODO instead of having a circular ROI, implement a general elliptical ROI
# of which a circular ROI is a special case
# once this is done, support for drawing ellipses can be added in the GUI

import csv
from collections import namedtuple
from datetime import datetime
from itertools import izip, product

import numpy as np

from vidtools import Color, PyCap

# observer pattern:
# ROIs: observers, RoiCap: observable

class RoiCap(PyCap):
    'video capture that supports regions of interest (ROIs)'

    _max_rois = 100

    def __init__(self, video, title=None):
        'RoiCap(path or device id) -> RoiCap object'
        self.rois = {}
        self.latest_frame = None # latest frame read from capture
        super(RoiCap, self).__init__(video, title)

    def add_roi(self, roi):
        'register a new ROI'
        if len(self.rois) < RoiCap._max_rois:
            self.rois[roi._id] = roi
            roi.cap = self
        else:
            template = 'cannot have more than {} regions'
            msg = template.format(RoiCap._max_rois)
            raise ValueError(msg)

    def delete_roi(self, roi_id):
        'delete ROI by ID'
        del self.rois[roi_id]
```

```
43
44      def _notify_rois(self):
45          'notify all ROIs on change'
46          for roi in self.rois.viewvalues():
47              roi.notified()
48
49          # drawing must take place after all ROIs collected data
50          self.draw_rois()
51
52      def draw_rois(self):
53          'draw all rois'
54          # can't be done by the ROIs themselves because all ROIs have to be
55          # notified and collect data before the frame may be drawn on
56          for roi in self.rois.viewvalues():
57              roi.draw()
58
59          # remember to show the frame AFTER drawing, remember not to
60          # draw on the frame before notifying all observers
61
62      def next(self):
63          'get next frame, notify ROIs, draw ROIs'
64          self.latest_frame = super(RoiCap, self).next()
65          self._notify_rois()
66          return self.latest_frame
67
68  # the observer
69  class BaseRoi(object):
70      'base class to derive ROIs from'
71
72      # next unique ID a ROI will get
73      _next_id = 1
74
75      # default colors - can be overridden in childclasses or individual instances
76      active_color = Color(green=255, red=0, blue=0)
77      passive_color = Color(green=200, red=200, blue=200)
78
79      def __init__(self, vertex1, vertex2, description='N/A'):
80          '''BaseRoi((x1, y1), (x2, y2)[, description])
81          -> BaseRoi object
82
83          vertex1 and vertex2 are opposite vertices of rectangular ROI
84
85          description: optional description for ROI'''
```

```
86          self._id = BaseRoi._next_id
87          BaseRoi._next_id += 1
88
89          self.deaf = False # flag used to ignore notifications
90          self.cap = None # set when registered as an observer via RoiCap.add_roi
91          self.vertex1 = vertex1
92          self.vertex2 = vertex2
93          self.description = description
94          self.collected = []
95
96          # compute center of rectangle for drawing and
97          # start and end indices for slicing
98          self.st_x, self.end_x = sorted((vertex1[0], vertex2[0]))
99          self.st_y, self.end_y = sorted((vertex1[1], vertex2[1]))
100         center_x = self.st_x + (self.end_x - self.st_x)/2
101         center_y = self.st_y + (self.end_y - self.st_y)/2
102         self.center = (center_x, center_y)
103
104         # increment ends by one to get correct values for slicing
105         self.end_x += 1
106         self.end_y += 1
107
108         # compute the mask
109         self.mask = self.compute_mask()
110
111     def compute_mask(self):
112         '''returns boolean mask (or None) used to customize shape of region
113         (see CircRoi.compute_mask for an example)'''
114         return None
115
116     @property
117     def roicolor(self):
118         return self.passive_color if self.deaf else self.active_color
119
120     def __repr__(self):
121         return 'BaseRoi({}, {}, {}, {})'.format(self.vertex1, self.vertex2,
122             self.description, str(self.mask).replace('\n', ','))
123
124     def __str__(self):
125         return 'BaseRoi(vertex1={}, vertex2={}, ...)'.format(vertex1, vertex2)
126
127     def notified(self):
128         'called on notfication from observable'
```

```
129         if self.deaf:
130             return
131 
132         collected = self.collect()
133         if collected is not None:
134             self.collected.append(collected)
135 
136     def info(self):
137         'get ROI info as string'
138         now = datetime.now()
139         date = now.date().isoformat()
140         time = now.time().strftime('%H:%M:%S')
141 
142         info = 'ROI_ID {}\nROI_TYPE {}\nROI_DESCRIPTION {}\nDATE {}\nTIME {}'
143         info = info.format(self._id, self, self.description, date, time)
144 
145         if self.cap is not None:
146             capinfo = 'CAP_FILE {}\nCAP_TITLE {}\nCAP_FPS {}'
147             title, video, fps = self.cap.video, self.cap.title, self.cap.fps
148             capinfo = capinfo.format(title, video, fps)
149             info += '\n' + capinfo
150 
151         return info + '\n'
152 
153     def get_rect(self):
154         'slice rectangular ROI from observed frame (does not apply mask)'
155         # NOTE: applying boolean mask to 2D array flattens array!
156         frame = self.cap.latest_frame
157         return frame[self.st_y:self.end_y, self.st_x:self.end_x]
158 
159     def collect(self):
160         '''collect data from observed capture or return None if nothing of
161         interest has been observed'''
162         return None
163 
164     def draw(self):
165         'draw ROI with id on observed frame'
166         self.draw_id()
167         self.draw_outline()
168 
169     def draw_id(self):
170         'draws ROI id on observed frame at center of rectangle'
171         # not exactly flexible, but these parameters look ok-ish for now...
```

```
172         font = 1 # HERSHEY_PLAIN
173         font_scale = 0.9
174         pos = self.center[0] - 4, self.center[1] + 4
175         self.cap.latest_frame.put_text(
176             str(self._id), pos, font, font_scale, self.roicolor)
177 
178     def draw_outline(self):
179         'outline the actual ROI area (respecting masks) on observed frame'
180         raise NotImplementedError
181 
182     def contains(self, point):
183         'contains((x, y)) -> bool'
184         raise NotImplementedError
185 
186 class RectRoi(BaseRoi):
187     'simple rectangular region of interest'
188 
189     def __init__(self, vertex1, vertex2, description='N/A'):
190         # the reason for overriding this method was in order to adjust
191         # the docstring - is there a better way?
192         '''RectRoi((x1, y1), (x2, y2)[, description]) -> RectRoi object
193 
194         vertex1 and vertex2 are opposite vertices of the rectangle
195         description: optional description for ROI'''
196         super(RectRoi, self).__init__(vertex1, vertex2, description)
197 
198     def __repr__(self):
199         return "RectRoi({}, {}, '{}')".format(
200             self.vertex1, self.vertex2, self.description)
201 
202     def __str__(self):
203         return 'RectRoi(vertex1={}, vertex2={})'.format(
204             self.vertex1, self.vertex2)
205 
206     def draw_outline(self):
207         'draw rectangular ROI area on observed frame'
208         frame = self.cap.latest_frame
209         frame.draw_rectangle(self.vertex1, self.vertex2, self.roicolor)
210 
211     def contains(self, point):
212         'contains((x, y)) -> bool'
213         x, y = point
214         return self.st_x <= x < self.end_x and self.st_y <= y < self.end_y
```

```
215
216 class CircRoi(BaseRoi):
217     'circular region of interest'
218     # TODO support partial circles
219
220     def __init__(self, center, radius, description='N/A'):
221         '''CircRoi((x, y), radius[, description]) -> CircRoi object
222
223         description: optional description for ROI'''
224         # calling superclass-init does a few lines of redudant computation
225         # such as recomputing the center (big whoop)
226         v1 = (center[0] - radius, center[1] - radius)
227         v2 = (center[0] + radius, center[1] + radius)
228         self.radius = radius
229         self.radius_sq = radius**2
230
231         super(CircRoi, self).__init__(v1, v2, description)
232
233     def compute_mask(self):
234         dx = self.end_x - self.st_x
235         dy = self.end_y - self.st_y
236
237         mask = np.zeros((dx, dy), dtype=np.bool)
238         macenter = (dx/2, dy/2)
239
240         # NOTE: this takes long if radius is large
241         for x, y in product(xrange(dx), xrange(dy)):
242             if (x - macenter[0])**2 + (y - macenter[1])**2 <= self.radius_sq:
243                 mask[y][x] = True
244
245         return mask
246
247     def __str__(self):
248         return 'CircRoi(center={}, radius={})'.format(self.center, self.radius)
249
250     def __repr__(self):
251         return "CircRoi({}, {}, '{}')".format(self.center, self.radius,
252             self.description)
253
254     def draw_outline(self):
255         'draw circular ROI area on observed frame'
256         frame = self.cap.latest_frame
257         frame.draw_circle(self.center, self.radius, self.roicolor)
```

```
258
259     def contains(self, point):
260         'contains((x, y)) -> bool'
261         x, y = point
262         dx = x - self.center[0]
263         dy = y - self.center[1]
264         return dx**2 + dy**2 <= self.radius_sq
265
266 # set up specialized example ROIs that MeanCircRoi and
267 # MeanRectRoi that collect mean channel data
268 MeanBgrRecord = namedtuple('MeanBgrRecord', 'pos_frames pos_msec color')
269
270 def _collect_mean_bgr(instance, use_mask):
271     '''helper for MeanCircRoi.collect and MeanRectRoi.collect -
272     returns MeanBgrRecord or None'''
273     rect = instance.get_rect()
274
275     blue, green, red = [
276         getattr(rect, color) for color in ('blue', 'green', 'red')]
277
278     if use_mask:
279         blue, green, red = [
280             array[instance.mask] for array in (blue, green, red)]
281
282     blue_avg, green_avg, red_avg = [
283         float(np.mean(array)) for array in (blue, green, red)]
284
285     col = Color(blue_avg, green_avg, red_avg)
286
287     # test if col should be ignored
288     if instance.collected:
289         lastcol = instance.collected[-1].color
290         if all(abs(x - y) < instance.delta_ig for x, y in izip(col, lastcol)):
291             return None
292
293     return MeanBgrRecord(instance.cap.pos_frames, instance.cap.pos_msec, col)
294
295 def _to_file_mean_bgr(instance, datafile, maskfile):
296     'helper for MeanCircRoi.to_file and MeanRectRoi.to_file'
297     if maskfile:
298         np.save(maskfile, instance.mask)
299
300     datafile.write(instance.info() + '\n')
```

```
301     writer = csv.writer(datafile)
302     writer.writerow(('pos_frames', 'pos_msec', 'blue_avg',
303         'green_avg', 'red_avg'))
304     datafile.write('HEADER_END\n')
305 
306     rows = ((fp, ms, b, g, r) for fp, ms, (b, g, r) in instance.collected)
307     writer.writerows(rows)
308 
309 class MeanCircRoi(CircRoi):
310     '''circular region of interest that collects the mean blue, green and red
311     intensities in the observed region for each frame'''
312 
313     def __init__(self, center, radius, description='N/A', delta_ig=0):
314         '''MeanCircRoi((x, y), radius[, description[, delta_ig]])
315         -> MeanCircRoi object
316 
317         description: optional description for ROI
318 
319         delta_ig: in order to conserve memory, ignore any observed mean
320         color that differs less than delta_ig from the last collected color
321         for each channel'''
322 
323         super(MeanCircRoi, self).__init__(center, radius, description)
324         self.delta_ig = delta_ig
325 
326     def __repr__(self):
327         return "MeanCircRoi({}, {}, '{}', {})".format(
328             self.center, self.radius, self.description, self.delta_ig)
329 
330     def __str__(self):
331         return 'MeanCircRoi(center={}, radius={}, delta_ig={})'.format(
332             self.center, self.radius, self.delta_ig)
333 
334     def collect(self):
335         'collect() -> MeanBgrRecord object or None'
336         return _collect_mean_bgr(instance=self, use_mask=True)
337 
338     def to_file(self, datafile, maskfile=None):
339         'write info and collected data to datafile, save mask to maskfile'
340         _to_file_mean_bgr(instance=self, datafile=datafile, maskfile=maskfile)
341 
342 class MeanRectRoi(RectRoi):
343     '''rectangular region of interest that collects the mean blue, green and red
```

```
344     intensities in the observed region for each frame'''
345
346     def __init__(self, vertex1, vertex2, description='N/A', delta_ig=0):
347         super(MeanRectRoi, self).__init__(vertex1, vertex2, description)
348         self.delta_ig = delta_ig
349
350     def __repr__(self):
351         return "MeanRectRoi({}, {}, '{}', {})".format(
352             self.vertex1, self.vertex2, self.description, self.delta_ig)
353
354     def __str__(self):
355         return 'MeanRectRoi(vertex1={}, vertex2={}, delta_ig={})'.format(
356             self.vertex1, self.vertex2, self.delta_ig)
357
358     def collect(self):
359         'collect() -> MeanBgrRecord object or None'
360         return _collect_mean_bgr(instance=self, use_mask=False)
361
362     def to_file(self, datafile):
363         'write info and collected data to datafile'
364         _to_file_mean_bgr(instance=self, datafile=datafile, maskfile=None)
```

Das Modul `roi_planner_gui.py` ist eine grafische Benutzeroberfläche zur Planung von Beobachtungszonen.

roi_planner_gui.py

```
1  'a GUI for planning regions of interest in a video'
2
3  # NOTE: naming conventions for global variables
4  # - Tkinter widgets/frames: preceded by tk_
5  # example: tk_export_button = ttk.Button(...)
6  # - Tkinter mutable variables: preceded by tkvar_
7  # example: tkvar_allow_all = tk.BooleanVar(...)
8  # - functions used as callbacks: preceded by cb_
9  # example: cb_select_video
10 # - variables holding styling information/parameters: preceded by st_
11 # example: st_button_border
12 # - important non-widget globals/constants used in lots of places: uppercase
13 # example: PLAYER = VideoPlayer(...)
14 # - single character unicode strings used as symbols: preceded by sym_
15 # example: sym_white_circle = u'\u25CB'
16
17 # TODO: refactoring -> RoiBox class for listboxes? Move ROI related helpers?
18
```

```
19 # TODO: (option to) show preview of ROI w.r.t current parameters
20 # on video-mouseover?
21
22 # TODO: coupling between the player, its canvas and its video is too tight
23 # - both player and canvas unidirectionally interact with the video
24 # - player and canvas interact bidirectionally
25 # (this is not optimal, but hard to disentangle)
26
27 # standard library imports
28 from __future__ import division
29 from collections import OrderedDict
30 import csv
31 from datetime import timedelta
32 from functools import partial
33 import math
34 import os
35 import re
36 from ScrolledText import ScrolledText
37 from timeit import default_timer
38 import tkColorChooser
39 import tkFileDialog
40 import Tkinter as tk
41 import tkMessageBox
42 import ttk
43
44 # third party imports
45 import cv2
46 import numpy as np
47 from PIL import ImageTk, Image
48
49 # local application imports
50 import roitools
51 import vidtools
52
53 # --- GUI ROOT ---
54 # (must exist before creating StringVars and setting styles)
55 tk_root = tk.Tk()
56 # ----------------
57
58 # --- STYLE SET UP ---
59 st_font_name = 'Helvetica'
60 st_med_pt = 12
61 st_big_pt = 14
```

```
62  st_med_font = (st_font_name, st_med_pt)
63  st_big_font = (st_font_name, st_big_pt)
64  st_big_bold_font = (st_font_name, st_big_pt, 'bold')
65
66  # the global ttk style
67  st_global = ttk.Style(tk_root)
68  #st_global.theme_use('default')
69
70  # TODO: find nice theme?
71  # https://wiki.tcl-lang.org/page/List+of+ttk+Themes
72
73  # bordered label with big font
74  st_global.configure(
75      'Bordered.TLabel', borderwidth=1, foreground='black',
76      font=st_big_font, relief='solid')
77
78  # bordered button
79  st_button_border = dict(borderwidth=4, relief='raised')
80  st_global.configure(
81      'Generic.TButton', font=st_med_font, **st_button_border)
82
83  # bordered button for single symbols (play, stop, ...)
84  st_global.configure(
85      'SingleSymbol.TButton', font=st_big_bold_font,
86      width=3, **st_button_border)
87  # --------------------
88
89  # --- CLASSES/FUNCTIONS FOR CREATING WIDGETS ---
90
91  # NOTE input validation has been implemented via tracing - another option
92  # is to use the validatecommand keyword argument of entry widgets
93
94  class TracedVar(tk.StringVar, object):
95      'base class for a string variable with "live" value restriction'
96
97      def __init__(self, allow_empty=True, **kwargs):
98          super(TracedVar, self).__init__(**kwargs)
99          self._allow_empty = allow_empty
100         self.trace('w', self._validate_kill_args)
101         self._last_value = kwargs.get('value', '')
102
103     def check(self, value):
104         '''returns True if value is OK, False otherwise
```

```
105         (override this method in child classes)'''
106         return True
107 
108     # for user
109     def get(self):
110         return self._last_value
111 
112     # for internal use
113     def _get(self):
114         return super(TracedVar, self).get()
115 
116     def _accept(self, value):
117         self._last_value = value
118 
119     def _deny(self):
120         self.set(self._last_value)
121 
122     def _validate_kill_args(self, *_):
123         self._validate()
124 
125     def _validate(self):
126         value = self._get()
127         value_empty = not value
128 
129         if value_empty:
130             if self._allow_empty:
131                 self._accept(value)
132             else:
133                 self._deny()
134         elif self.check(value):
135             self._accept(value)
136         else:
137             self._deny()
138 
139 class TracedMaxLen(TracedVar):
140     'holds string with a maximum length'
141 
142     def __init__(self, max_len=np.inf, **kwargs):
143         super(TracedMaxLen, self).__init__(**kwargs)
144         self._max_len = max_len
145 
146     def check(self, value):
147         return (
```

```
148             super(TracedMaxLen, self).check(value) and
149             len(value) <= self._max_len)
150
151 class TracedNumber(TracedMaxLen):
152     'holds string representing a number'
153     # NOTE: remember that the user must be able to enter strings that start
154     # with . or - or -. (leading + is disallowed for simplicity)
155
156     # strings that are treated as '0' when encountered on their own
157     # (when the user calls get and inside the check method)
158     _allowed_prefixes = {'.', '-', '-.'}
159
160     def prefix_conv(self, value):
161         # convert values such as '.' to '0' when encountered on their own
162         return '0' if value in self._allowed_prefixes else value
163
164     def get(self):
165         return self.prefix_conv(super(TracedNumber, self).get())
166
167     def check(self, value):
168         success = False
169
170         if super(TracedNumber, self).check(value):
171             try:
172                 float(self.prefix_conv(value))
173             except ValueError:
174                 pass
175             else:
176                 # prevent useless leading zeros
177                 if not re.match('-?0\d', value):
178                     success = True
179
180         return success
181
182 class TracedMinMaxValue(TracedNumber):
183     # TODO: snapping to min/max value would be nice (if allowed by max length)
184     'holds string representing a number with a maximum and minimum value'
185
186     def __init__(self, min_value=-np.inf, max_value=np.inf, **kwargs):
187         super(TracedMinMaxValue, self).__init__(**kwargs)
188
189         if min_value >= 0:
190             self._allowed_prefixes = ('.',)
```

```
191
192         self._min_value = min_value
193         self._max_value = max_value
194
195     def check(self, value):
196         conv = self.prefix_conv(value)
197         return (
198             super(TracedMinMaxValue, self).check(value) and
199             self._min_value <= float(conv) <= self._max_value)
200
201 class TracedInt(TracedMinMaxValue):
202     'holds string representing an integer'
203
204     def __init__(self, **kwargs):
205         super(TracedInt, self).__init__(**kwargs)
206
207         if self._min_value < 0:
208             self._allowed_prefixes = ('-',)
209         else:
210             self._allowed_prefixes = tuple()
211
212     def check(self, value):
213         # NOTE: checking float(value).is_integer() would allow trailing
214         # zeros after the decimal
215         return (
216             super(TracedInt, self).check(value) and
217             self.prefix_conv(value).lstrip('+').lstrip('-').isdigit())
218
219 class DeleGetEntry(ttk.Entry, object):
220     '''intended for use with traced variables:
221     the entry delegates get calls to its textvariable and resets to
222     latest acceptable value on losing focus'''
223     # (in effect, this resets strings such as . or -. to '0' in this program)
224
225     def __init__(self, master, textvariable, *args, **kwargs):
226         self._textvariable = textvariable
227         super(DeleGetEntry, self).__init__(
228             master, *args, textvariable=textvariable, **kwargs)
229
230         self.bind('<FocusOut>', self._reset)
231
232     def get(self, *args, **kwargs):
233         return(self._textvariable.get())
```

```
234
235     def _reset(self, *_):
236         value = self.get()
237         self.delete(0, tk.END)
238         self.insert(0, value)
239
240     def configure(self, *args, **kwargs):
241         self._textvariable = kwargs.get('textvariable', self._textvariable)
242         return super(DeleGetEntry, self).configure(*args, **kwargs)
243
244     def __setitem__(self, item, value):
245         if item == 'textvariable':
246             self._textvariable = value
247         super(DeleGetEntry, self).__setitem__(item, value)
248
249 class Log(ScrolledText, object):
250     'scrolled textbox for usage as log'
251
252     def __init__(
253         self, master=None,
254         warning_tag_kws=dict(background='#FFFF00'), # yellow
255         error_tag_kws=dict(background='#FF9EAB'), # rosy red
256         success_tag_kws=dict(background='#9AFC58'), # light green
257         **kwargs):
258
259         if not ('background' in kwargs or 'bg' in kwargs):
260             kwargs['bg'] = '#EFF0F1' # light gray
261
262         super(Log, self).__init__(master, **kwargs)
263
264         self['state'] = 'disabled'
265         self.tag_config('warning', **warning_tag_kws)
266         self.tag_config('error', **error_tag_kws)
267         self.tag_config('success', **success_tag_kws)
268
269     def _empty(self):
270         return self.get('1.0', tk.END) == '\n'
271
272     def append(self, text, tag=None, prefix=None):
273         'appends text to the log'
274         self['state'] = tk.NORMAL
275
276         # start a new line if the log is not empty
```

```
277         if not self._empty():
278             self.insert(tk.END, '\n')
279 
280         self.insert(tk.END, '{}{}'.format(prefix or '', text), tag)
281         self['state'] = tk.DISABLED
282         self.see(tk.END)
283 
284     def warning(self, text, prefix='WARNING: '):
285         self.append(text, 'warning', prefix)
286 
287     def error(self, text, prefix='ERROR: '):
288         self.append(text, 'error', prefix)
289 
290     def success(self, text, prefix=None):
291         self.append(text, 'success', prefix)
292 
293 def make_scrolled_listbox(
294     frame_master, header=None, vertical=True, horizontal=True,
295     make_labelframe=False, **kws):
296     '''set up scolled ListBox inside a fresh frame, return (frame, box)
297     kws are passed through to Listbox constructor'''
298     # TODO: (option to) hide scrollbars when not needed
299 
300     # set up box in frame
301     boxrow = 0
302     if make_labelframe:
303         frame = ttk.Labelframe(frame_master, text=header or '')
304     else:
305         frame = ttk.Frame(frame_master)
306         if header:
307             ttk.Label(frame, text=header).grid(row=0, column=0)
308             boxrow = 1
309 
310     box = tk.Listbox(frame, **(kws or {}))
311     box.grid(row=boxrow, column=0)
312 
313     # set up scrollbars
314     if vertical:
315         vbar = ttk.Scrollbar(frame, command=box.yview, orient=tk.VERTICAL)
316         box['yscrollcommand'] = vbar.set
317         vbar.grid(row=boxrow, column=1, sticky=tk.NS)
318 
319     if horizontal:
```

```
320         hbar = ttk.Scrollbar(frame, command=box.xview, orient=tk.HORIZONTAL)
321         box['xscrollcommand'] = hbar.set
322         hbar.grid(row=boxrow + 1, column=0, sticky=tk.EW)
323 
324     # let <Control-a> select everything
325     box.bind('<Control-a>', lambda _: partial(box.selection_set, 0, tk.END)())
326 
327     return frame, box
328 # ---------------------------------------------------------------------
329 
330 # --- MISCELLANEOUS HELPERS ---
331 def bgr_to_hex(b, g, r):
332     '''bgr color intensities to hex rgb notation
333     example: bgr_to_hex(16, 32, 64) -> #402010'''
334     return '#{:02x}{:02x}{:02x}'.format(r, g, b)
335 
336 def walk_widgets(root):
337     'yield all descendants of a frame'
338     for child in root.winfo_children():
339         yield child
340         if isinstance(child, (ttk.Frame, ttk.Labelframe)):
341             for grandchild in walk_widgets(child):
342                 yield grandchild
343 
344 def time_str(timespan, unit, rounding=None):
345     'turns timespan in unit to human readable string'
346     if rounding is not None:
347         timespan = round(timespan, rounding)
348     td = timedelta(**{unit:timespan})
349     return re.sub('(\.\d+?)0+$', r'\1', str(td)) # strip unnecessary zeros
350 
351 def is_avi(path):
352     'checks first twelve bytes of file for valid avi header'
353     with open(path, 'rb') as f:
354         head = f.read(12)
355 
356     RIFF_signature = '\x52\x49\x46\x46'
357     type_avi = '\x41\x56\x49\x20'
358 
359     return (
360         len(head) == 12 and
361         head[:4] == RIFF_signature and
362         head[-4:] == type_avi)
```

```
363
364 def check_empty(value, description=None):
365     'issues warning and raises ValueError if value is the empty string'
366     if value == '':
367         msg = 'missing {} value'.format(description or '')
368         raise ValueError(msg)
369 # ----------------------------
370
371 # -- ROI RELATED HELPERS ---
372 class DummyCircRoi(roitools.CircRoi):
373     '''dummy circular ROI that does not compute a mask on construction -
374     mask computation can be expensive for large radius values and is
375     not needed for planning ROIs'''
376
377     def compute_mask(self):
378         'does nothing'
379         pass
380
381 def construct_roi_from_params(x, y):
382     'construct a ROI with center (x, y) and w.r.t. current region parameters'
383     rtype = tkvar_roitype.get()
384
385     if rtype == RECTANGULAR:
386         # get width and height
387         width_str, height_str = tkvar_width.get(), tkvar_height.get()
388         check_empty(width_str, 'region width')
389         check_empty(height_str, 'region height')
390         width, height = int(width_str), int(height_str)
391
392         # construct vertices such that (x, y) is region center
393         vertex1 = (
394             x - width//2,
395             y - height//2)
396         vertex2 = (
397             int(x + math.ceil(width/2.0)),
398             int(y + math.ceil(height/2.0)))
399
400         roi = roitools.RectRoi(vertex1, vertex2)
401
402     elif rtype == CIRCULAR:
403         radius_str = tkvar_radius.get()
404         check_empty(radius_str, 'region radius')
405         radius = int(radius_str)
```

```
406
407         # check if this ROI can fit
408         # TODO I really need to start supporting partial circular ROIs
409         # in order to get rid of this mess
410         in_bounds = not (
411             x - radius < 0 or
412             y - radius < 0 or
413             x + radius >= PLAYER.video.frame_width or
414             y + radius >= PLAYER.video.frame_height)
415
416         if in_bounds:
417             roi = DummyCircRoi((x, y), radius)
418         else:
419             msg = 'region out of frame - partial circles are not supported yet'
420             raise ValueError(msg)
421
422     return roi
423
424 def roi_info_line(roi):
425     '''returns simplified region info for display in listbox
426     the line always starts with the ROI id'''
427     circ = isinstance(roi, roitools.CircRoi)
428     symbol = sym_white_circle if circ else sym_ballot_box
429     unit = 'seconds'
430     rounding = 3
431
432     start = time_str(roi.start_pos_msec/1000.0, unit, rounding)
433     line = u'{} {} {}'.format(roi._id, symbol, start)
434
435     if hasattr(roi, 'end_pos_msec'):
436         end = time_str(roi.end_pos_msec/1000.0, unit, rounding)
437         line += u' {} {}'.format(sym_arrow_right, end)
438
439     return line
440
441 def roi_info_dict(roi):
442     'creates ordered dict with information on roi'
443     # keys of info dict
444     keys = (
445         'type', 'radius', 'center_x', 'center_y', 'vertex1_x', 'vertex1_y',
446         'vertex2_x', 'vertex2_y', 'start_pos_msec', 'end_pos_msec',
447         'start_pos_frames', 'end_pos_frames', 'constructor')
448
```

```
449     # keys with identical ROI attribute names
450     attrs = [
451         'start_pos_msec', 'end_pos_msec',
452         'start_pos_frames', 'end_pos_frames']
453 
454     info = OrderedDict.fromkeys(keys, '')
455     type_ = CIRCULAR if isinstance(roi, roitools.CircRoi) else RECTANGULAR
456     info['type'] = type_
457     info['constructor'] = repr(roi)
458 
459     if type_ == CIRCULAR:
460         attrs.append('radius')
461         info['center_x'] = roi.center[0]
462         info['center_y'] = roi.center[1]
463     else:
464         info['vertex1_x'] = roi.vertex1[0]
465         info['vertex1_y'] = roi.vertex1[1]
466         info['vertex2_x'] = roi.vertex2[0]
467         info['vertex2_y'] = roi.vertex2[1]
468 
469     for attr in attrs:
470         info[attr] = getattr(roi, attr, '')
471 
472     return info
473 # ----------------------------
474 
475 # --- CLASSES FOR PLAYBACK ---
476 class RoiVideo(roitools.RoiCap):
477     '''modified/extended RoiCap for use in this GUI
478     (keeps track of a brightness offset and yields RGB frames)'''
479 
480     # NOTE: I'd like to keep this from initiating interactions with widgets,
481     # if possible
482 
483     def __init__(self, *args, **kwargs):
484         self._brightness_offset = 0
485         super(RoiVideo, self).__init__(*args, **kwargs)
486 
487     def add_roi(self, roi):
488         'register new ROI and add starting timestamps to ROI on success'
489         super(RoiVideo, self).add_roi(roi)
490         roi.start_pos_msec = self.pos_msec
491         roi.start_pos_frames = self.pos_frames
```

```
492
493     def _bgr_to_rgb(self, frame):
494         cv2.cvtColor(frame, cv2.COLOR_BGR2RGB, frame)
495
496     def draw_active_rois(self):
497         for roi in self.rois.viewvalues():
498             if not hasattr(roi, 'end_pos_frames'):
499                 roi.draw()
500
501     def redraw(self):
502         'apply brightness offset to current frame and redraw rois'
503         # NOTE: the order is important here!
504
505         # get copy of unmodified frame (BGR) as read by VideoCapture
506         frame = self.latest_frame_original.copy()
507
508         # apply brightness offset before drawing ROIs
509         frame.adjust_brightness(self._brightness_offset)
510
511         # redraw rois before converting from BGR to RGB
512         self.latest_frame = frame
513         self.draw_active_rois()
514
515         # convert to RGB
516         self._bgr_to_rgb(frame)
517
518     @property
519     def brightness_offset(self):
520         return self._brightness_offset
521
522     @brightness_offset.setter
523     def brightness_offset(self, value):
524         # NOTE: currently, ROIs are not notified again if the brightness of a
525         # frame that has already been read changes - this has no effect for
526         # CircRoi and RectRoi which don't collect data
527
528         # when the offset is set, it needs to be applied to a copy of the
529         # unmodified current frame, else information can be lost
530         self._brightness_offset = np.clip(value, -255, 255)
531         self.redraw()
532
533     def next(self):
534         # NOTE: ROIs are not notified (like in the parent class), only
```

```
535         # redrawn, and only if active - this is fine for simple ROI planning
536         # with dummy ROIs that don't collect any data
537         # -> adjust code as needed if program is extended to collect ROI data
538 
539         self.latest_frame_original = vidtools.PyCap.next(self)
540         self.redraw()
541 
542         return self.latest_frame
543 
544 class VideoCaptureError(Exception):
545     'custom exception for video loading problems'
546     pass
547 
548 class VideoPlayer(object):
549     'class that mimics a video player'
550     # NOTE: I would like the player to be decoupled from widgets as strictly
551     # as possible, i.e. no initiation of interactions with widgets from here
552     # (I could not reasonably prevent the canvas from doing it, though)
553 
554     # NOTE: methods intended to be called from outside (not preceded by _)
555     # should perform actions that could reasonably be imagined to be found
556     # on a video player
557     # (it's OK to get attributes of the loaded video such as fps or number of
558     # frames through the video instance variable, because these are not
559     # properties of the player itself)
560 
561     def _re_init(self):
562         'reinitialize'
563         self.video = None # RoiVideo instance
564         self.videoname = None # base name of current video
565         self._paused = None # flag for pausing
566         self._backwards = False # flag for backwards play
567         self._target_delay = None # optimal delay between frames when playing
568         self._sch_play_delay = None # current scheduling delay when playing
569         self._last_play_show = None # last time of frame shown during playback
570         self._after_handle = None # return value of self.tk_root.after
571         self._canvas = None # canvas that shows image
572         self._window = None # window that holds canvas
573 
574         self.pause()
575         self._update_pos_vars()
576 
577     def __init__(self, tk_root=tk_root):
```

```
578         self.tk_root = tk_root
579         self._idle_delay = 50 # approximate delay between redraws when paused
580         self.tkvar_pos_frames = tk.DoubleVar() # current frame
581         self.tkvar_pos_time = tk.StringVar() # current time as string
582         self._re_init()
583 
584     def _update_pos_vars(self):
585         if self.video:
586             frame = self.video.pos_frames
587             msec = self.video.pos_msec
588         else:
589             frame = 1
590             msec = 0
591 
592         self.tkvar_pos_frames.set(frame)
593         self.tkvar_pos_time.set(time_str(msec/1000.0, 'seconds', rounding=0))
594 
595     def next(self):
596         frame = next(self.video)
597         self._update_pos_vars()
598         return frame
599 
600     def set_position(self, frame):
601         'set video to specific frame position and update display'
602         # NOTE: the check whether the video has more than one frame should not
603         # be necessary, but testing revealed that cv2.VideoCapture.read seems
604         # to have a bug for single frame videos:
605         # after reading the frame once and resetting the capture's position to
606         # zero, subsequent calls to read fail to return the frame
607 
608         if self.video.frame_count > 1:
609             self.video.pos_frames = frame
610             self.video.pos_frames -= 1
611             next(self)
612 
613     def refresh(self):
614         self.video.redraw()
615 
616     def pause(self):
617         self._paused = True
618 
619     def play(self):
620         'resume playback in the current direction'
```

```
621         if self._paused:
622             now = default_timer()
623             self._last_play_show = now
624         self._paused = False
625 
626     def toggle_play(self):
627         'pauses or resumes playback'
628         if self._paused:
629             self.play()
630         else:
631             self.pause()
632 
633     def play_backwards(self):
634         self._backwards = True
635         self.play()
636 
637     def play_forwards(self):
638         self._backwards = False
639         self.play()
640 
641     def stop(self):
642         self.pause()
643         self.set_position(0)
644 
645     def brightness_adjust(self, increment):
646         self.video.brightness_offset += increment
647 
648     def brightness_reset(self):
649         self.video.brightness_offset = 0
650 
651     def load_video(self, path):
652         'load video from filename - expects to be called from unloaded state'
653         # try to load the video
654         try:
655             video = RoiVideo(path)
656             next(video)
657         except (StopIteration, TypeError):
658             # TODO: what else can be raised by cv2.VideoCapture(path)
659             raise VideoCaptureError
660         else:
661             # missing critical attributes? (zero or None)
662             criticals = (
663                 video.fps, video.frame_count,
```

```
664                 video.frame_width, video.frame_height)
665
666             if not all(criticals):
667                 raise VideoCaptureError
668             else:
669                 # video seems to be fine
670                 # TODO consider maximum delay for videos with very low fps
671                 self._target_delay = max(int(1000.0/video.fps), 1) # msec
672                 self._sch_play_delay = self._target_delay # initial value
673                 self.video = video
674                 self.videoname = os.path.basename(path)
675                 self._set_up_window()
676                 self._schedule_next_frame()
677
678     def unload_video(self):
679         'unload video - expects to be called from loaded state'
680         self.tk_root.after_cancel(self._after_handle)
681         self.video.release()
682         self._window.destroy()
683         self._re_init()
684
685     def _set_up_window(self):
686         'sets up the display window and canvas'
687         no_border = dict(borderwidth=0, highlightthickness=0)
688
689         # set up window
690         self._window = tk.Toplevel(tk_root, **no_border)
691         self._window.title(self.videoname)
692         self._window.resizable(False, False)
693
694         # set up canvas
695         self._canvas = TouchScreen(
696             player=self, master=self._window, height=self.video.frame_height,
697             width=self.video.frame_width, **no_border)
698         self._canvas.grid(row=0, column=0)
699
700         # let close eject video
701         self._window.protocol("WM_DELETE_WINDOW", cb_eject_video)
702
703         # let space bar pause/resume
704         self._window.bind('<space>', lambda _: self.toggle_play())
705
706     def _manage_playback_scheduling(self):
```

```
707         '''to be called directly after showing a frame during playback!
708         performs actions to recalculate the current _sch_play_delay'''
709         # NOTE: the scheduling mechanism is not very sophisticated and
710         # constrained by the fact that Tkinter's after method only accepts
711         # integer millisecond delays - that said, for lots of videos a
712         # (forward) playback speed roughly equal to native speed is achieved
713         # (which is fine for planning regions of interest - maybe use another
714         # software to watch the latest blockbuster)
715 
716         now = default_timer()
717         actual_delay = (now - self._last_play_show)*1000.0 # msec
718         self._last_play_show = now
719 
720         adjusted = (self._target_delay/actual_delay)*self._sch_play_delay
721         self._sch_play_delay = max(int(round(adjusted)), 1)
722 
723     def _schedule_next_frame(self):
724         'pause or play the video by scheduling the next frame to show'
725         delay = self._idle_delay # when unloaded or paused
726 
727         if self.video:
728             # if playing, get the next frame and set
729             # a roughly appropriate waiting time between frames
730             if not self._paused:
731                 if self._backwards:
732                     self.video.pos_frames -= 2 # clips to zero
733 
734                 try:
735                     next(self)
736                 except StopIteration:
737                     self.pause()
738 
739                 # reached beginning of video while playing backwards? -> stop
740                 if self._backwards and self.video.pos_frames == 1:
741                     self.pause()
742 
743             # update canvas
744             # (even when paused because ROIs/brightness may have changed)
745             self._show_frame(self.video.latest_frame)
746 
747             # adjust delay (needs to be done after showing the frame!)
748             # NOTE: time _manage_playback_scheduling takes itself is ignored
749             if not self._paused:
```

```
750                 self._manage_playback_scheduling()
751                 delay = self._sch_play_delay
752 
753         # reschedule
754         self._after_handle = self.tk_root.after(
755             delay, self._schedule_next_frame)
756 
757     def _show_frame(self, frame):
758         'convert frame to PhotoImage and put on canvas'
759         image = ImageTk.PhotoImage(image=Image.fromarray(frame))
760         self._image = image # prevent garbage collection!
761         self._canvas.create_image(0, 0, anchor=tk.NW, image=image)
762 
763 class TouchScreen(tk.Canvas, object):
764     # TODO no rectangle selection outside of canvas!
765 
766     '''a canvas for use with a VideoPlayer instance
767     which supports mouse actions ("touching") to add ROIs
768 
769     right click -> add ROI with current region parameters
770     left click + drag -> add rectangular ROI at selected area'''
771 
772     def _re_init(self):
773         # keep track of the starting position of a mouse rectangle selection
774         self._start_x = None
775         self._start_y = None
776 
777         # delete last selection rectangle
778         if getattr(self, '_rect', None) is not None:
779             self.delete(self._rect)
780         self._rect = None
781 
782     def __init__(self, player, master=None, **kwargs):
783         kwargs.setdefault('cursor', 'cross')
784         super(TouchScreen, self).__init__(master, **kwargs)
785         self._player = player
786         self._re_init()
787 
788         # right click -> add ROI with current region parameters
789         self.bind('<Button-3>', self._rclick_add_roi)
790 
791         # left click and drag -> add ROI at area of selected rectangle
792         self.bind('<Button-1>', self._lclick)
```

```
793         self.bind('<B1-Motion>', self._ldrag)
794         self.bind('<ButtonRelease-1>', self._lrelease_add_roi)
795 
796     def _canvas_xy(self, event):
797         'convert event x, y coordinates to canvas x, y coordinates as integers'
798         x = self.canvasx(event.x)
799         y = self.canvasx(event.y)
800         return int(x), int(y)
801 
802     def _rclick_add_roi(self, event):
803         'right click adds ROI w.r.t. current region parameters'
804         x, y = self._canvas_xy(event)
805 
806         try:
807             roi = construct_roi_from_params(x, y)
808         except ValueError as e:
809             tk_log.error(str(e))
810         else:
811             self._add_roi_helper(roi)
812 
813     def _lclick(self, event):
814         'freeze the player and create a modifiable rectangle on the canvas'
815         # make player leave the canvas alone
816         tk_root.after_cancel(self._player._after_handle)
817 
818         # create rectangle
819         x, y = self._start_x, self._start_y = self._canvas_xy(event)
820         color = bgr_to_hex(*roitools.BaseRoi.active_color)
821         self._rect = self.create_rectangle(x, y, x, y, outline=color)
822 
823     def _ldrag(self, event):
824         'expand the current rectangle'
825         x, y = self._canvas_xy(event)
826         self.coords(self._rect, self._start_x, self._start_y, x, y)
827 
828     def _lrelease_add_roi(self, event):
829         'add ROI at selected rectangle, unfreeze player'
830         x, y = self._canvas_xy(event)
831 
832         # add ROI
833         vertex1 = (self._start_x, self._start_y)
834         vertex2 = (x, y)
835         roi = roitools.RectRoi(vertex1, vertex2)
```

```
836         self._add_roi_helper(roi)
837 
838         # cleanup
839         self._re_init()
840 
841         # unfreeze player
842         self._player._schedule_next_frame()
843 
844     def _add_roi_helper(self, roi):
845         'helper for adding roi to associated player after mouse action'
846         # add ROI to video
847         try:
848             self._player.video.add_roi(roi)
849         except ValueError as e:
850             # happens when maximum number of regions is exceeded
851             tk_log.error(str(e))
852             roitools.BaseRoi._next_id -= 1
853         else:
854             self._player.refresh()
855             tk_active_roi_box.insert(tk.END, roi_info_line(roi))
856             tk_log.append('started region {}'.format(roi._id))
857 # ----------------------------
858 
859 # --- GLOBALS/CONSTANTS ----
860 # (no widgets)
861 PLAYER = VideoPlayer()
862 FRAMES = 'frames'
863 SECONDS = 'seconds'
864 CIRCULAR = 'circular'
865 RECTANGULAR = 'rectangular'
866 ttk_DISABLED = 'disabled'
867 ttk_NORMAL = '!{}'.format(ttk_DISABLED)
868 # --------------------------
869 
870 # --- SYMBOLS ---
871 sym_eject = u'\u23CF'
872 sym_triangle_right = u'\u23F5'
873 sym_triangle_left = u'\u23F4'
874 sym_double_vbar = u'\u23F8'
875 sym_black_square = u'\u23F9'
876 sym_white_circle = u'\u25CB'
877 sym_ballot_box = u'\u2610'
878 sym_double_triangle_left = u'\u23EA'
```

```
879 sym_double_triangle_right = u'\u23E9'
880 sym_arrow_up = u'\u2B06'
881 sym_arrow_down = u'\u2B07'
882 sym_arrow_right = u'\u2192'
883 sym_stroked_o = u'\u00D8'
884 # ---------------
885
886 # --- WIDGET CALLBACKS ---
887 # (and helpers)
888
889 # INTERACTION WITH MAIN WINDOW
890 lose_unsaved_warning = 'All unsaved regions will be lost.'
891
892 def cb_confirm_quit():
893     'if a video is loaded, ask before closing the program'
894     msg = 'Exit program?\n{}'.format(lose_unsaved_warning)
895     if not PLAYER.video or tkMessageBox.askyesno('Exit?', msg):
896         tk_root.destroy()
897
898 # VIDEO EJECTION
899 def cb_eject_video(confirm=True):
900     'unloads video through PLAYER and updates GUI'
901     if confirm:
902         msg = 'Eject video?\n{}'.format(lose_unsaved_warning)
903         if not tkMessageBox.askyesno('Eject?', msg):
904             return
905
906     # clear listboxes
907     for box in (tk_active_roi_box, tk_finished_roi_box):
908         box.delete(0, tk.END)
909
910     set_widget_states_unloaded()
911
912     videoname = PLAYER.videoname
913     PLAYER.unload_video()
914     tkvar_videoname.set('')
915     tk_log.append('ejected "{}"'.format(os.path.basename(videoname)))
916
917 # VIDEO SELECTION
918 def cb_select_video():
919     'ejects previous video and loads video through PLAYER, then updates GUI'
920
921     # get file from user
```

```
922     path = tkFileDialog.askopenfilename()
923     basename = os.path.basename(path) if path else ''
924     if not basename:
925         return
926 
927     if PLAYER.video:
928         cb_eject_video(confirm=False)
929 
930     # currently, we want avi files for best OpenCV support, see
931     # https://stackoverflow.com/a/20865122
932     if not (tkvar_allow_all.get() or is_avi(path)):
933         msg = '"{}" not an avi - currently only avi files are supported'
934         msg = msg.format(basename)
935         tk_log.error(msg)
936         return
937 
938     # try to load video, set up widgets, log
939     try:
940         PLAYER.load_video(path)
941     except VideoCaptureError:
942         msg = 'can\'t process "{}" as video (corrupt file?)'.format(basename)
943         tk_log.error(msg)
944     else:
945         roitools.BaseRoi._next_id = 1
946         tk_bar['to'] = PLAYER.video.frame_count
947         set_widget_states_loaded()
948 
949         # set video name label
950         width = tk_videoname_label['width']
951         label_text = (basename if len(basename) <= width else
952                       basename[:max(width - 3, 0)] + '...')
953         tkvar_videoname.set(label_text)
954 
955         # log
956         num_frames = int(PLAYER.video.frame_count)
957         msg = 'loaded "{}" ({} fps, {} frame{})'.format(
958             basename, PLAYER.video.fps, num_frames, 's'*(num_frames > 1))
959         tk_log.success(msg)
960 
961         # TODO MANUAL
962         manual = (
963             'left click: add ROI by drawing a rectangle\n'
964             'right click: add ROI with fixed parameters')
```

```
965         tk_log.append(manual)
966 
967 # VIDEO PROGRESS
968 def cb_bar_set_frame(event):
969     'sets video position according to clicked position on progress bar'
970     # NOTE: the problem here is that BETWEEN the mousebutton-release-event
971     # and the execution of this function, the bar can be re-set by reading
972     # from PLAYER.tkvar_pos_frames - which means calling tk_bar.get will
973     # not return the position the bar was just clicked to
974     #
975     # solution:
976     # 1. the bar is disabled by default
977     # (events are still registered, this function is still executed)
978     # 2. in this callback: enable bar
979     # 3. generate click event with same x, y coordinates -> bar moves
980     # 4. disable bar again
981     # 5. NOW get the bar's position and set the video frame
982 
983     if PLAYER.video:
984         tk_bar.state([ttk_NORMAL])
985         tk_bar.event_generate('<Button-1>', x=event.x, y=event.y)
986         tk_bar.state([ttk_DISABLED])
987 
988         frame = round(tk_bar.get())
989         PLAYER.set_position(frame)
990 
991 # CLASSICAL CONTROLS (play, pause, stop)
992 cb_play_forwards = PLAYER.play_forwards
993 cb_play_backwards = PLAYER.play_backwards
994 cb_pause = PLAYER.pause
995 cb_stop = PLAYER.stop
996 
997 # JUMPING
998 def secs_to_frames(seconds):
999     '''helper for jumping
1000     converts seconds to nearest possible value in frames and logs adjustment'''
1001     frames = seconds*PLAYER.video.fps
1002     frames_adjusted = int(round(frames))
1003 
1004     # issue warning if adjusting changed input value
1005     if frames != frames_adjusted:
1006         secs_adjusted = frames_adjusted/PLAYER.video.fps
1007         template = 'adjusted jump value to {} {} ({} {})'
```

```
1008            insert = (secs_adjusted, SECONDS, frames_adjusted, FRAMES)
1009            tk_log.warning(template.format(*insert))
1010
1011        return frames_adjusted
1012
1013    def cb_jump(back=False):
1014        'changes video position according to jump step field'
1015        step = tk_step_entry.get()
1016        unit = tkvar_unit.get()
1017
1018        # empty?
1019        try:
1020            check_empty(step, description=unit)
1021        except ValueError as e:
1022            tk_log.error(str(e))
1023
1024        # get step in frames
1025        if unit == SECONDS:
1026            jump_frames = secs_to_frames(float(step))
1027        else:
1028            jump_frames = int(step)
1029
1030        if back:
1031            jump_frames *= -1
1032
1033        # work to do?
1034        if jump_frames == 0:
1035            return
1036
1037        # perform the jump (always in frames for exactness)
1038        before_pos_frames = PLAYER.video.pos_frames
1039        before_pos_msec = PLAYER.video.pos_msec
1040
1041        PLAYER.set_position(before_pos_frames + jump_frames)
1042
1043        # log jump
1044        delta_frames = int(PLAYER.video.pos_frames - before_pos_frames)
1045        delta_msec = PLAYER.video.pos_msec - before_pos_msec
1046        delta_sec = delta_msec/1000.0
1047
1048        template = 'jumped {} {} ({} {})'
1049        if unit == SECONDS:
1050            insert = (delta_sec, SECONDS, delta_frames, FRAMES)
```

```
1051        else:
1052            insert = (delta_frames, FRAMES, delta_sec, SECONDS)
1053
1054        msg = template.format(*insert)
1055        tk_log.append(msg)
1056
1057        # issue warning if target not met
1058        # NOTE: if the target has not been met AND we're not at the beginning
1059        # or end of the video, that's a bug! :)
1060        if abs(delta_frames) < abs(jump_frames):
1061            if back:
1062                where = 'beginning'
1063            else:
1064                where = 'end'
1065            tk_log.warning('reached {} of video'.format(where))
1066
1067    cb_jump_forward = partial(cb_jump, back=False)
1068    cb_jump_back = partial(cb_jump, back=True)
1069
1070    # BRIGHTNESS
1071
1072    # the brightness increment value should give the user reasonably fine grained
1073    # control over the video brightness without larger adjustments being tiresome
1074    # - ten seems to be a good value
1075    brightness_increment = 10
1076    cb_brightness_up = partial(PLAYER.brightness_adjust, brightness_increment)
1077    cb_brightness_down = partial(PLAYER.brightness_adjust, -brightness_increment)
1078    cb_brightness_reset = PLAYER.brightness_reset
1079
1080    # ROI DELETION
1081    def selected_ids(listbox):
1082        # NOTE: parsing ROI ids from the beginning of listbox strings is straight
1083        # forward but mixes presentation and model -> on next refactoring consider
1084        # Listbox subclass internally keeping track of ROIs being held
1085        'returns list of ROI ids corresponding to selected listbox items'
1086        lines = (listbox.get(i) for i in listbox.curselection())
1087        return [int(line.split(' ', 1)[0]) for line in lines]
1088
1089    def del_rois(listbox):
1090        'delete selected ROIs from listbox'
1091        ids = selected_ids(listbox)
1092
1093        # no selection but box not empty?
```

```
1094        if not ids and listbox.get(0, tk.END):
1095            msg = 'No regions selected. Delete all?'
1096            if tkMessageBox.askyesno('Delete all?', msg):
1097                listbox.selection_set(0, tk.END)
1098                ids = selected_ids(listbox)
1099
1100        # no work -> abort
1101        if not ids:
1102            return
1103
1104        # delete selected lines in listbox
1105        for line_index in reversed(listbox.curselection()):
1106            listbox.delete(line_index)
1107
1108        # delete from capture
1109        for id_ in ids:
1110            del PLAYER.video.rois[id_]
1111
1112        # log
1113        msg = 'deleted region{} {}'.format(
1114            's'*(len(ids) > 1), ', '.join(map(str, ids)))
1115        tk_log.append(msg)
1116
1117    def cb_del_active_rois():
1118        del_rois(tk_active_roi_box)
1119        PLAYER.refresh()
1120
1121    def cb_del_finished_rois():
1122        del_rois(tk_finished_roi_box)
1123
1124    # ROI FINALIZATION
1125    def cb_finish_rois():
1126        'move selected active ROIs to finished ROIs'
1127        # TODO: structure in parts similar to cb_delete_rois function -> refactor?
1128        # (also, this function got pretty long)
1129        ids = selected_ids(tk_active_roi_box)
1130
1131        # no selection but box not empty? -> ask to finish all
1132        if not ids and tk_active_roi_box.get(0, tk.END):
1133            msg = 'No regions selected. Finish all?'
1134            if tkMessageBox.askyesno('Finish all?', msg):
1135                tk_active_roi_box.selection_set(0, tk.END)
1136                ids = selected_ids(tk_active_roi_box)
```

```
1137
1138    # no work -> abort
1139    if not ids:
1140        return
1141
1142    # delete selected lines in active roi listbox
1143    for i in reversed(tk_active_roi_box.curselection()):
1144        tk_active_roi_box.delete(i)
1145
1146    # mark ROIs as finished by giving them ending timestamps
1147    end_msec = PLAYER.video.pos_msec
1148    end_frame = PLAYER.video.pos_frames
1149    swap = tkvar_swap_times.get()
1150
1151    for id_ in ids:
1152        roi = PLAYER.video.rois[id_]
1153        roi.end_pos_msec = end_msec
1154        roi.end_pos_frames = end_frame
1155
1156        # switch start/end if option set
1157        if swap and roi.end_pos_frames < roi.start_pos_frames:
1158            roi.end_pos_msec, roi.start_pos_msec = (
1159                roi.start_pos_msec, roi.end_pos_msec)
1160            roi.end_pos_frames, roi.start_pos_frames = (
1161                roi.start_pos_frames, roi.end_pos_frames)
1162
1163    # insert lines into finished roi box
1164    for id_ in ids:
1165        roi = PLAYER.video.rois[id_]
1166        tk_finished_roi_box.insert(tk.END, roi_info_line(roi))
1167
1168    # log
1169    msg = 'finished region{} {}'.format(
1170        's'*(len(ids) > 1), ', '.join(map(str, ids)))
1171    tk_log.append(msg)
1172
1173    # redraw frame
1174    PLAYER.refresh()
1175
1176 # ROI EXPORTING
1177 def cb_export():
1178    'export finished ROIs'
1179    ids = selected_ids(tk_finished_roi_box)
```

```
    # no selection but box not empty? -> ask to finish all
    if not ids and tk_finished_roi_box.get(0, tk.END):
        msg = 'No regions selected. Export all?'
        if tkMessageBox.askyesno('Export all?', msg):
            tk_finished_roi_box.selection_set(0, tk.END)
            ids = selected_ids(tk_finished_roi_box)

    # no work -> abort
    if not ids:
        return

    # ask the user for a filename
    ext = '.csv'
    filename = tkFileDialog.asksaveasfilename(
        title='Export region information', defaultextension=ext,
        initialfile='regions_export{}'.format(ext),
        filetypes=(('spreadsheet', '*{}'.format(ext)),))

    if not filename:
        return

    # build lines for csv file
    row_dicts = [roi_info_dict(PLAYER.video.rois[id_]) for id_ in ids]

    # write csv file
    try:
        with open(filename, 'w') as out:
            writer = csv.DictWriter(out, fieldnames=row_dicts[0].keys())
            writer.writeheader()
            writer.writerows(row_dicts)
    except EnvironmentError as e:
        tk_log.error(str(e))
    else:
        # log
        msg = 'exported region{} {}'.format(
            's'*(len(ids) > 1), ', '.join(map(str, ids)))
        tk_log.success(msg)

        # delete if option set
        if tkvar_del_exported.get():
            del_rois(tk_finished_roi_box)
```

```
1223 # COLOR CHOOSING
1224 def cb_select_color():
1225     'set display color for all regions'
1226     current_bgr = roitools.BaseRoi.active_color
1227     current_rgb = current_bgr[::-1]
1228 
1229     new_rgb, _ = tkColorChooser.askcolor(current_rgb)
1230 
1231     if new_rgb:
1232         new_bgr = vidtools.Color(*new_rgb[::-1])
1233         roitools.BaseRoi.active_color = new_bgr
1234 
1235         if PLAYER.video:
1236             PLAYER.refresh()
1237 # ------------------------
1238 
1239 # --- WIDGET CREATION, PLACEMENT, CALLBACK BINDING ---
1240 # (from top to bottom, left to right)
1241 
1242 tk_root.title('ROI Planner')
1243 tk_root.protocol('WM_DELETE_WINDOW', cb_confirm_quit)
1244 
1245 # parent frame for all widgets
1246 tk_root_frame = ttk.Frame(tk_root)
1247 tk_root_frame.grid(row=0, column=0)
1248 
1249 # SET UP MENU
1250 tk_menubar = tk.Menu(tk_root)
1251 tk_options_menu = tk.Menu(tk_menubar)
1252 tk_menubar.add_cascade(label='Options', menu=tk_options_menu)
1253 tk_root['menu'] = tk_menubar
1254 
1255 # option to globally switch ROI color
1256 tk_options_menu.add_command(
1257     label='set region color...', command=cb_select_color)
1258 
1259 # separator before checkbuttons
1260 tk_options_menu.add_separator()
1261 
1262 # option to automatically delete exported videos
1263 tkvar_del_exported = tk.BooleanVar(value=True)
1264 tk_options_menu.add_checkbutton(
1265     label='delete exported regions',
```

```
1266     offvalue=False, onvalue=True, variable=tkvar_del_exported)
1267
1268 # option to switch start and end times for rois with start > end
1269 tkvar_swap_times = tk.BooleanVar(value=False)
1270 label = (
1271     'swap timestamps when regions finish '
1272     'with end time earlier than start time')
1273 tk_options_menu.add_checkbutton(
1274     label=label, offvalue=False, onvalue=True, variable=tkvar_swap_times)
1275
1276 # option to allow non-avi videos
1277 tkvar_allow_all = tk.BooleanVar(value=False)
1278 tk_options_menu.add_checkbutton(
1279     label='allow non-avi files (unstable)',
1280     offvalue=False, onvalue=True, variable=tkvar_allow_all)
1281
1282 # SET UP FILE PICKER
1283 tk_select_frame = ttk.Frame(tk_root_frame)
1284 tk_select_frame.grid(row=0, column=0)
1285 tkvar_videoname = tk.StringVar()
1286
1287 # select video button
1288 tk_select_button = ttk.Button(
1289     tk_select_frame, text='select video',
1290     style='Generic.TButton', command=cb_select_video)
1291 tk_select_button.grid(row=0, column=0)
1292
1293 # video name label
1294 # TODO: make label a scrolled readonly entry, hide the
1295 # scrollbar whenever the entry fits
1296 tk_videoname_label = ttk.Label(
1297     tk_select_frame, width=57,
1298     textvariable=tkvar_videoname, style='Bordered.TLabel')
1299 tk_videoname_label.bind('<Button-1>', lambda *_: cb_select_video())
1300 tk_videoname_label.grid(row=0, column=2)
1301
1302 # eject video button
1303 tk_eject_button = ttk.Button(
1304     tk_select_frame, text=sym_eject, style='SingleSymbol.TButton',
1305     command=cb_eject_video)
1306 tk_eject_button.grid(row=0, column=1)
1307
1308 # separator under file picker
```

```
1309 ttk.Separator(tk_root_frame, orient=tk.HORIZONTAL).grid(row=1, column=0)
1310
1311 # SET UP PROGRESS INFO
1312 tk_progress_frame = ttk.Frame(tk_root_frame)
1313 tk_progress_frame.grid(row=2, column=0)
1314
1315 # progress bar
1316 tk_bar = ttk.Scale(
1317     tk_progress_frame, orient=tk.HORIZONTAL, from_=1,
1318     takefocus=False, variable=PLAYER.tkvar_pos_frames)
1319
1320 # jump bar with left click in addition to right click
1321 # -> make bar think a '<Button-1>' event is a '<Button-3>' event
1322 tk_bar.bind(
1323     '<Button-1>',
1324     lambda event: tk_bar.event_generate('<Button-3>', x=event.x, y=event.y))
1325
1326 # jump to clicked position at mousebutton-release
1327 for button in range(1, 6):
1328     event = '<ButtonRelease-{}>'.format(button)
1329     tk_bar.bind(event, cb_bar_set_frame)
1330
1331 tk_bar.grid(row=0, column=0)
1332
1333 # time label next to progress bar
1334 tk_timelabel = ttk.Label(tk_progress_frame, textvariable=PLAYER.tkvar_pos_time)
1335 tk_timelabel.grid(row=0, column=1)
1336
1337 # SET UP PLAYER CONTROLS
1338 tk_control_frame = ttk.Frame(tk_root_frame)
1339 tk_control_frame.grid(row=3, column=0)
1340
1341 tk_pauseplay_frame = ttk.Labelframe(tk_control_frame, text='player controls')
1342 tk_pauseplay_frame.grid(row=1, column=0)
1343
1344 # play button
1345 tk_play_button = ttk.Button(
1346     tk_pauseplay_frame, text=sym_triangle_right,
1347     style='SingleSymbol.TButton', command=cb_play_forwards)
1348 tk_play_button.grid(row=0, column=0)
1349
1350 # play backwards button
1351 tk_play_backwards_button = ttk.Button(
```

```
1352     tk_pauseplay_frame, text=sym_triangle_left,
1353     style='SingleSymbol.TButton', command=cb_play_backwards)
1354 tk_play_backwards_button.grid(row=0, column=1)
1355 
1356 # pause button
1357 tk_pause_button = ttk.Button(
1358     tk_pauseplay_frame, text=sym_double_vbar,
1359     style='SingleSymbol.TButton', command=cb_pause)
1360 tk_pause_button.grid(row=0, column=2)
1361 
1362 # stop button
1363 tk_stop_button = ttk.Button(
1364     tk_pauseplay_frame, text=sym_black_square,
1365     style='SingleSymbol.TButton', command=cb_stop)
1366 tk_stop_button.grid(row=0, column=3)
1367 
1368 # SET UP JUMP CONTROLS
1369 tk_jump_frame = ttk.Labelframe(tk_control_frame, text='jump controls')
1370 tk_jump_frame.grid(row=1, column=1)
1371 
1372 # jump back button
1373 tk_jump_back_button = ttk.Button(
1374     tk_jump_frame, text=sym_double_triangle_left,
1375     style='SingleSymbol.TButton', command=cb_jump_back)
1376 tk_jump_back_button.grid(row=0, column=0)
1377 
1378 # jump step entry
1379 val_restrictions = dict(min_value=-1000, max_value=99999, max_len=10)
1380 tk_timeentry_frame = ttk.Frame(tk_jump_frame)
1381 tk_timeentry_frame.grid(row=0, column=1)
1382 tkvar_seconds = TracedMinMaxValue(**val_restrictions)
1383 
1384 tkvar_frames = TracedInt(**val_restrictions)
1385 tk_step_entry = DeleGetEntry(
1386     tk_timeentry_frame, width=7, justify=tk.CENTER, textvariable=None)
1387 tk_step_entry.grid(row=0, column=0)
1388 
1389 # jump step unit label
1390 tkvar_unit = tk.StringVar()
1391 tk_unit_label = ttk.Label(
1392     tk_timeentry_frame, anchor=tk.CENTER, textvariable=tkvar_unit)
1393 tk_unit_label.grid(row=1, column=0)
1394 
```

```
1395 # jump forward button
1396 tk_jump_forward_button = ttk.Button(
1397     tk_jump_frame, text=sym_double_triangle_right,
1398     style='SingleSymbol.TButton', command=cb_jump_forward)
1399 tk_jump_forward_button.grid(row=0, column=2)
1400
1401 # group of radiobuttons to select unit for jump step (seconds or frames)
1402 # select seconds: tie jump entry to TracedMinMaxValue (tkvar_seconds)
1403 # select frames: tie jump entry to TracedInt (tkvar_frames)
1404 tk_selectunit_frame = ttk.Frame(tk_jump_frame)
1405 tk_selectunit_frame.grid(row=0, column=3)
1406 tk_seconds_radiobutton = ttk.Radiobutton(
1407     tk_selectunit_frame, text=SECONDS, value=SECONDS, variable=tkvar_unit,
1408     command=partial(tk_step_entry.configure, textvariable=tkvar_seconds))
1409 tk_seconds_radiobutton.invoke() # set initial selection, trigger command
1410 tk_seconds_radiobutton.grid(row=0, column=0)
1411
1412 tk_frames_radiobutton = ttk.Radiobutton(
1413     tk_selectunit_frame, text=FRAMES, value=FRAMES, variable=tkvar_unit,
1414     command=partial(tk_step_entry.configure, textvariable=tkvar_frames))
1415 tk_frames_radiobutton.grid(row=1, column=0)
1416
1417 # SET UP BRIGHTNESS CONTROLS
1418 tk_bright_frame = ttk.Labelframe(
1419     tk_control_frame, text='brightness controls')
1420 tk_bright_frame.grid(row=1, column=2)
1421
1422 # brightness up button
1423 tk_brightup_button = ttk.Button(
1424     tk_bright_frame, text=sym_arrow_up,
1425     style='SingleSymbol.TButton', command=cb_brightness_up)
1426 tk_brightup_button.grid(row=0, column=0)
1427
1428 # brightness down button
1429 tk_brightdown_button = ttk.Button(
1430     tk_bright_frame, text=sym_arrow_down,
1431     style='SingleSymbol.TButton', command=cb_brightness_down)
1432 tk_brightdown_button.grid(row=0, column=1)
1433
1434 # brightness reset button
1435 tk_brightreset_button = ttk.Button(
1436     tk_bright_frame, text=sym_stroked_o,
1437     style='SingleSymbol.TButton', command=cb_brightness_reset)
```

```
1438 tk_brightreset_button.grid(row=0, column=2)
1439
1440 # separator under controls
1441 ttk.Separator(tk_root_frame, orient=tk.HORIZONTAL).grid(row=4, column=0)
1442
1443 # SET UP FRAMES FOR CONTROLING ACTIVE AND FINISHED ROIS
1444 tk_roi_frame = ttk.Frame(tk_root_frame)
1445 tk_roi_frame.grid(row=6, column=0)
1446
1447 # frame and box for active ROIs
1448 box_kws = dict(
1449     frame_master=tk_roi_frame, make_labelframe=True, horizontal=False,
1450     selectmode=tk.MULTIPLE, exportselection=False, height=8)
1451 tk_active_roi_frame, tk_active_roi_box = make_scrolled_listbox(
1452     header='active regions - start', **box_kws)
1453 tk_active_roi_box['width'] = 25
1454 tk_active_roi_frame.grid(row=0, column=0)
1455
1456 # frame and box for finished ROIs
1457 tk_finished_roi_frame, tk_finished_roi_box = make_scrolled_listbox(
1458     header='finished regions - lifetime', **box_kws)
1459 tk_finished_roi_box['width'] = 35
1460 tk_finished_roi_frame.grid(row=0, column=1)
1461
1462 # buttons for roi deletion
1463 del_text = 'delete'
1464 del_button_kws = dict(
1465     text=del_text, width=len(del_text), style='Generic.TButton')
1466 del_grid_kws = dict(row=3, column=0)
1467
1468 # delete active ROIs button
1469 tk_del_active_button = ttk.Button(
1470     tk_active_roi_frame, command=cb_del_active_rois, **del_button_kws)
1471 tk_del_active_button.grid(**del_grid_kws)
1472
1473 # delete finished ROIs button
1474 tk_del_finished_button = ttk.Button(
1475     tk_finished_roi_frame, command=cb_del_finished_rois, **del_button_kws)
1476 tk_del_finished_button.grid(**del_grid_kws)
1477
1478 # finish active ROIs button
1479 fin_text = 'finish'
1480 tk_finish_button = ttk.Button(
```

```
1481        tk_active_roi_frame, text=fin_text, width=len(fin_text),
1482        style='Generic.TButton', command=cb_finish_rois)
1483    tk_finish_button.grid(row=3, column=0)
1484
1485    # export finished ROIs button
1486    exp_text = 'export'
1487    tk_export_button = ttk.Button(
1488        tk_finished_roi_frame, text=exp_text, width=len(exp_text),
1489        command=cb_export, style='Generic.TButton')
1490    tk_export_button.grid(row=3, column=0)
1491
1492    # SET UP ROI PARAMETER SELECTION FRAME
1493    # TODO: section might need refactoring
1494    tk_roi_select_frame = ttk.Labelframe(tk_roi_frame, text='region parameters')
1495    tk_roi_select_frame.grid(row=0, column=2)
1496
1497    tkvar_roitype = tk.StringVar(value=RECTANGULAR)
1498
1499    roi_entry_params = dict(master=tk_roi_select_frame, width=5, justify=tk.CENTER)
1500    tkvar_width = TracedInt(**val_restrictions)
1501    tkvar_height = TracedInt(**val_restrictions)
1502
1503    tkvar_radius = TracedInt(**val_restrictions)
1504    tk_width_entry, tk_height_entry, tk_radius_entry = [
1505        DeleGetEntry(textvariable=var, **roi_entry_params)
1506        for var in (tkvar_width, tkvar_height, tkvar_radius)]
1507
1508    tk_rectangular_radiobutton = ttk.Radiobutton(
1509        tk_roi_select_frame, value=RECTANGULAR, variable=tkvar_roitype)
1510    tk_rectangular_radiobutton.grid(row=0, column=0)
1511    ttk.Label(tk_roi_select_frame, text=RECTANGULAR).grid(
1512        row=0, column=1, columnspan=2)
1513
1514    tk_width_entry.grid(row=1, column=1)
1515    ttk.Label(tk_roi_select_frame, text='px width').grid(row=1, column=2)
1516
1517    tk_height_entry.grid(row=3, column=1)
1518    ttk.Label(tk_roi_select_frame, text='px height').grid(row=3, column=2)
1519
1520    tk_circular_radiobutton = ttk.Radiobutton(
1521        tk_roi_select_frame, value=CIRCULAR, variable=tkvar_roitype)
1522    tk_circular_radiobutton.grid(row=4, column=0)
1523    tk_circular_label = ttk.Label(tk_roi_select_frame, text=CIRCULAR)
```

```
1524 tk_circular_label.grid(row=4, column=1, columnspan=2)
1525 tk_radius_entry.grid(row=5, column=1)
1526 ttk.Label(tk_roi_select_frame, text='px radius').grid(row=5, column=2)
1527
1528 # SET UP LOG
1529 tk_log = Log(tk_root_frame, height=10)
1530 tk_log.grid(row=7, column=0)
1531 # -------------------------------------------------
1532
1533 # --- GRID CONFIGURATON ---
1534
1535 # NOTE getting all widgets here assumes they are static - if widgets are added
1536 # or removed at runtime, walk_widgets needs to be called at runtime as needed
1537 ALL_WIDGETS = set(walk_widgets(tk_root_frame))
1538 ALL_WIDGETS.add(tk_root_frame)
1539
1540 # this is a (crude?) implementation of cascading grid configuration:
1541 # the three dictionaries map widget classes, styles and individual widgets to
1542 # parameter dictionaries - each widget in the GUI is configured with the most
1543 # specific parameters that are available
1544
1545 # parameters will likely need adjustment if the GUI changes!
1546
1547 # map widget class to grid parameters
1548 class_grid_params = {
1549     ttk.Frame: dict(sticky=tk.EW),
1550     ttk.Label: dict(sticky=tk.W),
1551     ttk.Separator: dict(pady=20, sticky=tk.EW),
1552     ttk.Scale: dict(sticky=tk.EW),
1553     ttk.Radiobutton: dict(sticky=tk.W),
1554 }
1555
1556 # map widget style to grid parameters
1557 style_grid_params = {
1558     'Generic.TButton': dict(pady=(5, 0)),
1559     'SingleSymbol.TButton': dict(padx=1),
1560 }
1561
1562 # map individual widgets to grid parameters
1563 widget_grid_params = {
1564     tk_root_frame: dict(padx=10, pady=10),
1565     tk_select_button: dict(sticky=tk.NS, padx=(0, 1), pady=0),
1566     tk_videoname_label: dict(sticky=(tk.W, tk.NS), padx=(10, 0)),
```

```
1567     tk_progress_frame: dict(pady=(0, 15)),
1568     tk_timeentry_frame: dict(padx=5),
1569     tk_unit_label: dict(sticky=tk.EW),
1570     tk_selectunit_frame: dict(padx=(5, 0)),
1571     tk_roi_frame: dict(pady=(0, 15)),
1572     tk_bright_frame: dict(sticky=tk.E),
1573     tk_del_active_button: dict(sticky=tk.W),
1574     tk_finish_button: dict(sticky=tk.E),
1575     tk_log.frame: dict(sticky=tk.EW),
1576     tk_roi_select_frame: dict(sticky=tk.N),
1577     tk_circular_radiobutton: dict(pady=(5,0)),
1578 }
1579
1580 widget_grid_params[tk_export_button] = (
1581     widget_grid_params[tk_finish_button])
1582
1583 widget_grid_params[tk_del_finished_button] = (
1584     widget_grid_params[tk_del_active_button])
1585
1586 widget_grid_params[tk_circular_label] = (
1587     widget_grid_params[tk_circular_radiobutton])
1588
1589 # apply grid parameters
1590 for widget in ALL_WIDGETS:
1591     try:
1592         style = widget['style']
1593     except tk.TclError:
1594         style = None
1595
1596     kwargs = class_grid_params.get(widget.__class__, {}).copy()
1597     kwargs.update(style_grid_params.get(style, {}))
1598     kwargs.update(widget_grid_params.get(widget, {}))
1599
1600     widget.grid_configure(**kwargs)
1601
1602 # additional row/column configuration
1603 tk_progress_frame.columnconfigure(0, weight=1) # expand progress bar
1604 tk_control_frame.columnconfigure(1, pad=40)
1605 tk_roi_frame.columnconfigure(1, pad=60)
1606
1607 # pad all Labelframes
1608 # NOTE: can't use ipadx/ipady through Labelframe.configure because internal
1609 # padding is applied to frame before widgets are placed
```

```
1610 for widget in ALL_WIDGETS:
1611     if isinstance(widget, ttk.Labelframe):
1612         widget['padding'] = 5
1613 
1614 # pad left/right of video label
1615 tk_videoname_label['padding'] = (10, 0)
1616 
1617 # miscellaneous
1618 tk_root.resizable(False, False)
1619 # -------------------------
1620 
1621 # --- ENABLING/DISABLING WIDGETS ---
1622 def set_widget_states(widgets, state=tk.NORMAL):
1623     '''given an iterable of widgets, tries to set state to tk.NORMAL or
1624     tk.DISABLED and suppresses errors if that's not possible'''
1625     for widget in widgets:
1626         try:
1627             widget['state'] = state
1628         except tk.TclError:
1629             # deal with ttk widgets
1630             ttk_state = ttk_NORMAL if state == tk.NORMAL else ttk_DISABLED
1631 
1632             try:
1633                 widget.state([ttk_state])
1634             except (AttributeError, tk.TclError):
1635                 pass
1636 
1637 # function for state: no video
1638 set_widget_states_unloaded = partial(
1639     set_widget_states, ALL_WIDGETS - {tk_select_button}, tk.DISABLED)
1640 
1641 # function for state: video loaded
1642 # NOTE disabling the bar is a kludge to make setting the video position work
1643 # properly while playing - see cb_bar_set_frame for details
1644 set_widget_states_loaded = partial(
1645     set_widget_states, ALL_WIDGETS - {tk_bar}, tk.NORMAL)
1646 # ---------------------------------
1647 
1648 # --- TEST SECTION ---
1649 if __name__ == '__main__':
1650     set_widget_states_unloaded()
1651     tk_root.mainloop()
```

Das Modul `examples.py` bietet Hilfsroutinen und Anwendungsbeispiele, welche

auf den in `vidtools.py` und `roitools.py` bereitgestellten Werkzeugen aufbauen. Die Auswertung der Laborvideos in Kapitel 5 erfolgte mit Hilfe der Funktionen `get_roidata` und `collect_plot_export`.

examples.py

```
'''examples and helpers for vidtools and roitools users

Functions accepting ROIs work with MeanRectRoi or MeanCircRoi instances.

The main data collection functions are get_roidata and collect_plot_export.'''

from collections import deque
from datetime import datetime
from itertools import count
from math import log10, floor
from pprint import pprint as pretty_print
import os
import sys

import numpy as np
import matplotlib.pyplot as plt

# run as module with -m from root directory?
try:
    from roitools.roitools import RoiCap
except ImportError:
    pass

# (try to) import useful objects for the user if script is executed directly
if __name__ == '__main__' and __package__ is None:
    # add roitools directory to path
    dirname = os.path.dirname

    abs_path = os.path.abspath(sys.argv[0])
    one_up = dirname(dirname(abs_path))
    sys.path.insert(0, one_up)

    # imports
    try:
        import vidtools
        from vidtools import PyCap

        import roitools
```

```
39          from roitools import RoiCap, MeanBgrRecord, MeanRectRoi, MeanCircRoi
40      except ImportError:
41          pass
42
43
44  def reset_roi_ids():
45      'make new ROIs start at id 1 again'
46      BaseRoi._next_id = 1
47
48  def save_roi(roi, path):
49      '''save ROI as csv file
50      if path is a directory, uses filename roi_<id>_out.csv'''
51      if os.path.isdir(path):
52          filename = 'roi_{}_out.csv'.format(roi._id)
53          path = os.path.join(path, filename)
54
55      with open(path, 'w') as outfile:
56          roi.to_file(outfile)
57
58  def array_to_csv(filepath, array, header=''):
59      np.savetxt(filepath, array, delimiter=',', header=header, comments='')
60
61  def load_roi(roi_csv_path, header_rows=11):
62      'load ROI data as array from csv'
63      return np.loadtxt(roi_csv_path, skiprows=header_rows, delimiter=',')
64
65  def concat_arrays(arrays):
66      'concats arrays vertically'
67      return np.vstack(arrays)
68
69  def ms_to_frame(pos_msec, fps, ceil=True):
70      '''convert pos_msec to corresponding pos_frames of a RoiCap with given fps
71
72      ceil: if True, round up frame values lower than one to one'''
73      frame = int(pos_msec*fps/1000.0)
74      if ceil:
75          frame = max(frame, 1)
76      return frame
77
78  def frame_to_ms(pos_frames, fps, ceil=True):
79      '''convert pos_frames to corresponding pos_msec of a RoiCap with given fps
80
81      ceil: if True, returns at least 1000.0/fps (pos_msec of first frame)'''
```

```
82      msec = 1000.0*pos_frames/fps
83      if ceil:
84          msec = max(msec, 1000.0/fps)
85      return msec
86
87  def _set_up_result_directory(videopath):
88      'helper function to set up a directory for result files'
89      dirname = os.path.dirname(videopath)
90      videoname = os.path.basename(videopath)
91
92      now = datetime.now()
93      date = now.date().isoformat()
94      time = now.time().strftime('%Hh%Mm%Ss')
95
96      saveto = os.path.join(
97          dirname, '{}_analyzed_{}_{}'.format(videoname, date, time))
98      os.mkdir(saveto)
99
100     return saveto
101
102 def get_roidata(videopath, roi_specs):
103     '''videopath: complete path to video file
104     roi_specs: list of (roi_instance, birth_msec, death_msec) tuples, e.g.
105
106     roi_specs = [
107         (MeanRectRoi((0, 0), (100, 100)), 2000, 10000), # observe sec 2 -> 10
108         (MeanRectRoi((10, 10), (110, 110)), 10000, 20000) # observe sec 10 -> 20
109     ]
110
111     lets regions in roi_specs collect data during their lifetime
112
113     will dump all results and region screenshots into a new directory located
114     in the same directory as the video file, returns path'''
115
116     # work?
117     if not roi_specs:
118         return
119
120     # set up a directory for results
121     saveto = _set_up_result_directory(videopath)
122
123     # save ROI specs
124     with open(os.path.join(saveto, 'roi_specs.py'), 'w') as f:
```

```
125         pretty_print(roi_specs, f)
126
127     # msec -> frame
128     cap = RoiCap(videopath)
129     roi_specs = [(roi, ms_to_frame(start, cap.fps), ms_to_frame(end, cap.fps))
130                  for roi, start, end in roi_specs]
131
132     # set up queues for insertion/deletion
133     insertion_order = deque(
134         sorted(roi_specs, key=lambda (roi, birth, death): birth))
135     deletion_order = deque(
136         sorted(insertion_order, key=lambda (roi, birth, death): death))
137
138     # get last interesting frame number for progress-print
139     last_deletion_frame = min(deletion_order[-1][2], cap.frame_count)
140
141     while True:
142         # NOTE: fast forwarding to the next birth/death (via cap.play) instead
143         # of executing loop body for every frame has been tried and does not
144         # seem to yield a noticeable performance boost compared to this simpler
145         # version
146         # also, skipping via setting attributes seems to be non-exact for some
147         # types of videos -> see comments in image_series function
148
149         # add ROIs
150         take_screenshot = False
151         while insertion_order:
152             roi, birth, _ = insertion_order[0]
153
154             if cap.pos_frames + 1 == birth:
155                 insertion_order.popleft()
156                 cap.add_roi(roi)
157                 take_screenshot = True
158             else:
159                 break
160
161         # generate next frame, draw ROIs
162         try:
163             frame = next(cap)
164         except StopIteration:
165             break
166
167         # if ROIs were added, take screenshot
```

```
168         if take_screenshot:
169             im_filename = 'frame_{}.png'.format(int(cap.pos_frames))
170             frame.save(os.path.join(saveto, im_filename))
171 
172         # delete rois
173         while deletion_order:
174             roi, _, death = deletion_order[0]
175             if cap.pos_frames == death:
176                 deletion_order.popleft()
177                 save_roi(roi, saveto)
178                 cap.delete_roi(roi._id)
179             else:
180                 break
181 
182         # abort if all data is already collected
183         if not cap.rois and not insertion_order:
184             break
185 
186         # show progress
187         print '{:.1f} %\r'.format(100.0*cap.pos_frames/last_deletion_frame),
188         sys.stdout.flush()
189 
190     # save leftover rois (with death > video length)
191     for roi in cap.rois.itervalues():
192         save_roi(roi, saveto)
193 
194     # save a screenshot of all rois
195     cap.rois = {}
196     cap.pos_frames -= 1
197     for roi, _, _ in roi_specs:
198         cap.add_roi(roi)
199     last_frame = next(cap)
200     last_frame.save(os.path.join(saveto, 'all_rois.png'))
201 
202     print '100.0 %'
203     return saveto
204 
205 def plot_roi(array, linetype='-', channels='bgr', title='ROI Plot', ax=None):
206     '''creates plot from ROI data array, returns the axes object (the plot)
207     (use plt.show() to show the plot)'''
208     if len(array.shape) <= 1:
209         print 'nothing to plot'
210         return
```

```
211
212     msec = array[:, 1]
213     channels = set(channels.lower()).intersection('bgr')
214
215     if ax is None:
216         _, ax = plt.subplots()
217
218     ax.set_xlabel('milliseconds')
219     ax.set_ylabel('mean channel intensity')
220     ax.set_title(title)
221
222     for c, idx in zip('bgr', xrange(2, 5)):
223         if c in channels:
224             ax.plot(msec, array[:, idx], linetype + c)
225
226     ax.set_xlim(array[0][1], array[-1][1])
227     return ax
228
229 def collect_plot_export(videopath, roi_specs):
230     '''compound comfort function
231
232     visualizes regions with non-overlapping lifetimes and saves helpful files
233
234     - runs get_roidata with roi_specs (see get_roidata documentation)
235     - concatenates all rows of collected data and saves to all_rois.csv
236     - plots the result with horizontal lines indicating change of ROI
237     - saves the figure as an image
238
239     returns the axes object (the plot)
240     (use plt.show() to show the plot)'''
241
242     # get numerical data
243     saveto = get_roidata(videopath, roi_specs)
244
245     # concat data from all rois
246     region_ids = [roi._id for roi, _, _ in roi_specs]
247     roi_arrays = [load_roi(os.path.join(saveto, 'roi_{}_out.csv'.format(id_)))
248                   for id_ in region_ids]
249
250     merged = concat_arrays(roi_arrays)
251     array_to_csv(os.path.join(saveto, 'all_rois.csv'), merged)
252
253     # plot and add ROI-deaths to plot
```

```
254     ax = plot_roi(merged)
255     for roi, birth, death in roi_specs:
256         ax.axvline(death, color='black')
257 
258     ax.get_figure().savefig(os.path.join(saveto, 'fig_autosave.png'),
259                             dpi=1200, bbox_inches='tight')
260 
261     print 'saved data to {}'.format(saveto)
262     return ax
263 
264 def image_series(videopath, start_msec, end_msec, n):
265     'save every nth frame between start_msec and end_msec'
266     # NOTE: I compared iterating the video frame-by-frame versus skipping ahead
267     # n frames by setting the pos_frames attribute. OpenCV sometimes misses
268     # the correct frame for non-avi files with the second approach.
269     # This can only be seen when taking images from a video that
270     # numbers its frames on-screen because the pos_frames and pos_msec
271     # attributes report the values you would expect after setting them.
272 
273     # NOTE: tested with well behaved 60 fps videos
274 
275     # set up a directory for results
276     saveto = _set_up_result_directory(videopath)
277 
278     # create template to pad filenames with zeros to make them ordered
279     cap = PyCap(videopath)
280     pad = int(floor(log10(cap.frame_count/float(n)) + 1))
281     template = '{{:0>{}d}}_{{}}.png'.format(pad)
282 
283     # iterate and save images
284     imgcount = count(1)
285     t_one_frame = 1000.0/cap.fps
286     cap.pos_msec = start_msec
287 
288     if cap.pos_msec != start_msec:
289         print('adjusted start_msec to {}'.format(cap.pos_msec))
290         start_msec = cap.pos_msec
291 
292     for i, frame in enumerate(cap):
293         # current cap.pos_msec and cap.pos_frames are the values for the NEXT
294         # frame that can be retrieved with next(cap)
295         t_abs = cap.pos_msec - t_one_frame
296         if t_abs > end_msec:
```

```
297             break
298
299         if i%n == 0:
300             t_rel = t_abs - start_msec
301             img_fname = template.format(next(imgcount), t_rel)
302             frame.save(os.path.join(saveto, img_fname))
303
304     return saveto
305
306 def get_frames(videopath, frames):
307     '''frames: iterable of frame numbers
308     saves screenshots for given frame numbers'''
309
310     # set up a directory for results
311     saveto = _set_up_result_directory(videopath)
312
313     frames = set(frames)
314     last = max(frames)
315     cap = PyCap(videopath)
316     template = 'frame_{}.png'
317
318     # we don't jump around with cap.pos_frames directly because it can be
319     # inexact for some types of videos (i.e. non-avi files)
320     for frame in cap:
321         if cap.pos_frames in frames:
322             frame.save(os.path.join(saveto, template.format(cap.pos_frames)))
323         if cap.pos_frames > last:
324             break
325
326     return saveto
```

Temperaturberechnung

Das Modul `signal_to_temp.py` implementiert die Umrechnung von gemessener Kanalitensität zu Temperatur nach dem im Haupttext beschriebenen Verfahren. Die Bedienung erfolgt über die Kommandozeile unter der Verwendung der folgenden Optionen.

```
usage: signal_to_temp.py [-h] [-red RED] [-green GREEN] [-blue BLUE]
                         [-tstart TSTART] [-tstep TSTEP]

optional arguments:
  -h, --help show this help message and exit
```

```
  -red RED signal of red channel at melting point
  -green GREEN signal of green channel at melting point
  -blue BLUE signal of blue channel at melting point
  -tstart TSTART start temperature [in K]
  -tstep TSTEP temperature step [in K]
```

signal_to_temp.py

```
1  # -*- coding: utf-8 -*-
2
3  from __future__ import division
4  from argparse import ArgumentParser
5  from datetime import datetime
6  import numpy as np
7  import os
8  from sys import argv
9  import pylab
10
11 # CONSTANTS
12 h = 6.62607004*(10**(-34.0)) # Planck's constant [in Js]
13 c = 299792458.0 # speed of light [in m/s]
14 k = 1.38064852*(10**(-23.0)) # Boltzmann constant [in J/K]
15 T_melt = 1687.0 # melting point of Si [in K]
16
17 # parse arguments
18 parser = ArgumentParser()
19 helptxt = 'signal of {} channel at melting point'
20 parser.add_argument(
21     '-red', type=float, default=np.nan, help=helptxt.format('red'))
22 parser.add_argument(
23     '-green', type=float, default=np.nan, help=helptxt.format('green'))
24 parser.add_argument(
25     '-blue', type=float, default=np.nan, help=helptxt.format('blue'))
26 parser.add_argument(
27     '-tstart', type=float, default=1200.0, help='start temperature [in K]')
28 parser.add_argument(
29     '-tstep', type=int, default=1, help='temperature step [in K]')
30
31 if len(argv) == 1:
32     parser.print_help()
33     parser.exit()
34
35 args = parser.parse_args()
36 S_red_melt, S_green_melt, S_blue_melt = args.red, args.green, args.blue
```

```
37 T_start, T_step = args.tstart, args.tstep
38
39 # Quantum Efficiency of Guppy G 146
40 # columns: lambda [in nm], red [in %], green [in %], blue [in %]
41 Q = np.array([
42 [400, 1, 1, 12],
43 [410, 1, 2, 16],
44 [420, 0, 2, 20],
45 [430, 0, 2, 24],
46 [440, 0, 3, 28],
47 [450, 0, 4, 30],
48 [460, 0, 6, 30],
49 [470, 0, 9, 28],
50 [480, 0, 11, 26],
51 [490, 0, 14, 22],
52 [500, 0, 18, 18],
53 [510, 1, 24, 12],
54 [520, 1, 28, 8],
55 [530, 2, 30, 4],
56 [540, 1, 30, 2],
57 [550, 1, 28, 1],
58 [560, 2, 25, 0],
59 [570, 6, 21, 0],
60 [580, 13, 17, 0],
61 [590, 20, 11, 0],
62 [600, 24, 7, 0],
63 [610, 24, 3, 0],
64 [620, 24, 2, 0],
65 [630, 23, 1, 0],
66 [640, 22, 0, 0],
67 [650, 21, 0, 0],
68 [660, 19, 0, 0],
69 [670, 18, 1, 0],
70 [680, 16, 1, 0],
71 [690, 15, 2, 0],
72 [700, 14, 2, 0]]).astype(np.float64)
73
74 Q[:, 0] = Q[:, 0]*(10**(-9.0)) # convert nm to m
75 Q[:, 1:] /= 100.0 # convert from percent
76
77 # set up temperature range to examine
78 T_range = np.arange(T_start, T_melt + T_step, T_step, dtype=np.float64)
79 T_range[-1] = T_melt
```

```
80
81  # compute C_red, C_green, C_blue
82  # from equation S_color = C_color * SUM(...)
83  # with known values for S_color at melting point from experiment
84  lambda_i = Q[:-1, 0]
85  delta_lambda_i = np.diff(Q[:, 0])
86  lambda_i_pow_minus5 = Q[:-1, 0]**(-5.0)
87  lambda_term_i = delta_lambda_i*lambda_i_pow_minus5
88
89  exp_func = lambda lambda_i, T: np.exp(h*c/(lambda_i*k*T))
90  exp_term_i_melt = 1.0/(exp_func(lambda_i, T_melt) - 1)
91
92  Q_red_i = Q[:-1, 1]
93  Q_green_i = Q[:-1, 2]
94  Q_blue_i = Q[:-1, 3]
95
96  exp_lambda_term_i_melt = exp_term_i_melt*lambda_term_i
97  S_red_melt_div_C_red = np.sum(Q_red_i*exp_lambda_term_i_melt)
98  S_green_melt_div_C_green = np.sum(Q_green_i*exp_lambda_term_i_melt)
99  S_blue_melt_div_C_blue = np.sum(Q_blue_i*exp_lambda_term_i_melt)
100
101 C_red = S_red_melt/S_red_melt_div_C_red
102 C_green = S_green_melt/S_green_melt_div_C_green
103 C_blue = S_blue_melt/S_blue_melt_div_C_blue
104
105 # debug prints
106 # print 'S_red_melt = {:5.1f} => C_red = {:e}'.format(S_red_melt, C_red)
107 # print 'S_green_melt = {:5.1f} => C_green = {:e}'.format(S_green_melt, C_green)
108 # print 'S_blue_melt = {:5.1f} => C_blue = {:e}'.format(S_blue_melt, C_blue)
109
110 # compute S_color(T) for all colors and all temperatures
111 # TODO numpyfy the loop(s)
112 C_Q = ((C_red, Q_red_i), (C_green, Q_green_i), (C_blue, Q_blue_i))
113
114 signals = [[C_col*np.sum(Q_i*lambda_term_i*(1.0/(exp_func(lambda_i, T) -1)))
115             for C_col, Q_i in C_Q]
116           for T in T_range]
117
118 signals = np.array(signals, dtype=np.float64)
119 T_range_transposed = T_range[:, np.newaxis]
120
121 # mapping T -> S_color(T)
122 # columns: T, S_red(T), S_green(T), S_blue(T)
```

```
123 ST_function = np.hstack((T_range_transposed, signals))
124
125 # write csv file
126 filename = './ST_r={}_g={}_b={}_T_melt={}.csv'.format(
127     S_red_melt, S_green_melt, S_blue_melt, T_melt)
128 header = 'T, S_red, S_green, S_blue'
129 np.savetxt(filename, ST_function, delimiter=',', header=header, comments='')
130
131 # plot
132 pylab.ion()
133 pylab.figure()
134 pylab.plot(ST_function[:,0], ST_function[:,1], 'r')
135 pylab.plot(ST_function[:,0], ST_function[:,2], 'g')
136 pylab.plot(ST_function[:,0], ST_function[:,3], 'b')
137 pylab.xlabel('Temperature T/K')
138 pylab.ylabel('Signal S')
```

Literaturverzeichnis

[1] M. Binnewies, M. Finze, M. Jäckel, P. Schmidt, H. Willner, G. Rayner-Canham, *Allgemeine und Anorganische Chemie*, 3., vollständig überarbeitete Auflage, Springer Spektrum, Berlin, **2016**, ISBN: 978-3-662-45066-6 (siehe S. 1–4).

[2] A. F. Holleman, E. Wiberg, N. Wiberg, *Lehrbuch der Anorganischen Chemie*, 102., stark umgearbeitete und verbesserte Auflage, OCLC: 180963521, Walter de Gruyter, Berlin New York, **2007**, ISBN: 978-3-11-017770-1 (siehe S. 1–4).

[3] H. Binder, *Lexikon der chemischen Elemente: Das Periodensystem in Fakten, Daten und Zahlen*, 1., Hirzel, S., Verlag, Stuttgart, **1999**, ISBN: 978-3-7776-0736-8 (siehe S. 1).

[4] *CRC Handbook of Chemistry and Physics: A Ready-Reference Book of Chemical and Physical Data*, 84th ed, (Hrsg.: C. R. Company, D. R. Lide), CRC Press, Boca Raton, **2003**, ISBN: 978-0-8493-0484-2 (siehe S. 1, 23, 83).

[5] *Magnesium and Magnesium Alloys*, (Hrsg.: M. M. Avedesian, H. Baker, A. International), ASM International, Materials Park, OH, **1999**, ISBN: 978-0-87170-657-7 (siehe S. 2, 4–7).

[6] D. Lindström, P. Nortier, D. Sichen, „Functions of Mg and Mg–CaO Mixtures in Hot Metal Desulfurization“, *steel research international* **2014**, *85*, 76–88, ISSN: 1869-344X, DOI 10.1002/srin.201300071 (siehe S. 2).

[7] D. Kopeliovich, Desulfurization of Steel, **2012**, https://www.substech.com/dokuwiki/doku.php?id=desulfurization_of_steel (besucht am 21. 08. 2019) (siehe S. 2).

[8] S. Sandlöbes, M. Friák, S. Korte-Kerzel, Z. Pei, J. Neugebauer, D. Raabe, „A Rare-Earth Free Magnesium Alloy with Improved Intrinsic Ductility“, *Scientific Reports* **2017**, *7*, DOI 10.1038/s41598-017-10384-0, ISSN: 2045-2322 (siehe S. 2).

[9] M. H. Yoo, „Slip, Twinning, and Fracture in Hexagonal Close-Packed Metals“, *Metallurgical Transactions A* **1981**, *12*, 409–418, ISSN: 1543-1940, DOI 10.1007/BF02648537 (siehe S. 2).

[10] P. G. Partridge, „The Crystallography and Deformation Modes of Hexagonal Close-Packed Metals“, *Metallurgical Reviews* **1967**, *12*, 169–194, ISSN: 0076-6690, DOI 10.1179/mtlr.1967.12.1.169 (siehe S. 2).

[11] Z. Zeng, J.-F. Nie, S.-W. Xu, C. H. J. Davies, N. Birbilis, „Super-Formable Pure Magnesium at Room Temperature“, *Nature Communications* **2017**, *8*, DOI 10.1038/s41467-017-01330-9, ISSN: 2041-1723 (siehe S. 2).

[12] A. Kielbus, „Corrosion Resistance of Elektron 21 Magnesium Alloy“, *Journal of Achievements in Materials and Manufacturing Engineering* **2007**, *22* (siehe S. 3).

[13] E. L. Dreizin, C. H. Berman, E. P. Vicenzi, „Condensed-Phase Modifications in Magnesium Particle Combustion in Air“, *Combustion and Flame* **2000**, *122*, 30–42, ISSN: 0010-2180, DOI 10.1016/S0010-2180(00)00101-2 (siehe S. 3).

[14] ICSC 0289 - MAGNESIUM (POWDER), http://www.inchem.org/documents/icsc/icsc/eics0289.htm (besucht am 21. 08. 2019) (siehe S. 3).

[15] Magnesium 13112, https://www.sigmaaldrich.com/catalog/product/aldrich/13112 (besucht am 21. 08. 2019) (siehe S. 3).

[16] Versuch: Magnesium verbrennt unter Wasser, https://lp.uni-goettingen.de/get/text/2111 (besucht am 20. 08. 2019) (siehe S. 3).

[17] C. Yuill, Extinguishing Magnesium Fires, **1955**, https://www.fireengineering.com/articles/print/volume-108/issue-11/features/extinguishing-magnesium-fires.html (besucht am 19. 08. 2019) (siehe S. 3).

[18] S. Frank, S. Gneiger in Development of Non-Flammable Magnesium Alloys, ISDM Conference, Leoben, **2017** (siehe S. 3).

[19] U.S. Geological Survey, Magnesium Statistics and Information, **2019**, https://www.usgs.gov/centers/nmic/magnesium-statistics-and-information (besucht am 03. 08. 2019) (siehe S. 3, 6, 8, 97, 99).

[20] S. Ramakrishnan, P. Koltun, „Global Warming Impact of the Magnesium Produced in China Using the Pidgeon Process“, *Resources Conservation and Recycling* **2004**, *42*, 49–64, ISSN: 09213449, DOI 10.1016/j.resconrec.2004.02.003 (siehe S. 3, 4, 6–9).

[21] H. Davy, „XXIII. Electro-Chemical Researches, on the Decomposition of the Earths; with Observations on the Metals Obtained from the Alkaline Earths, and on the Amalgam Procured from Ammonia“, *Philosophical Transactions of the Royal Society of London* **1808**, *98*, 333–370, DOI 10.1098/rstl.1808.0023 (siehe S. 3, 4).

[22] F. Cherubini, M. Raugei, S. Ulgiati, „LCA of Magnesium Production“, *Resources Conservation and Recycling* **2008**, *52*, 1093–1100, ISSN: 09213449, DOI 10.1016/j.resconrec.2008.05.001 (siehe S. 4, 7).

[23] US Magnesium LLC, http://usmagnesium.com/ (besucht am 18. 08. 2019) (siehe S. 4, 6).

[24] V. Hasenberg in Lebenszyklusanalyse und ökologische Bewertung der Magnesiumherstellung, Werkstoff Forum intelligenter Leichtbau, Hannover Messe, **2012** (siehe S. 4, 7).

[25] L. M. Pidgeon (Dominion Magnesium Ltd), *US-Pat.*, 2330143A, **1943** (siehe S. 5, 6).

[26] J. M. Toguri, L. M. Pidgeon, „HIGH-TEMPERATURE STUDIES OF METALLURGICAL PROCESSES: PART II. THE THERMAL REDUCTION OF CALCINED DOLOMITE WITH SILICON“, *Canadian Journal of Chemistry* **1962**, *40*, 1769–1776, ISSN: 0008-4042, 1480-3291, DOI 10.1139/v62-271 (siehe S. 6).

[27] J. R. Wynnyckyj, L. M. Pidgeon, „Equilibria in the Silicothermic Reduction of Calcined Dolomite“, *Metallurgical Transactions* **1971**, *2*, 979–986, ISSN: 0360-2133, 1543-1916, DOI 10.1007/BF02664228 (siehe S. 6).

[28] T. Gebensleben, V. Becker, J. A. Becker, „Transient Light Emission from the Silicothermic Reduction of Magnesium Oxide with Potential for Monitoring Intermediate Compound Formation and Decay“, *SN Applied Sciences* **2020**, *2*, 401, ISSN: 2523-3971, DOI 10.1007/s42452-020-2126-4 (siehe S. 10–12, 14, 16, 20, 21, 26, 37, 40, 46, 90, 91, 93, 177).

[29] L. D. Alphei, C. Dobbe, V. Becker, J. A. Becker, „A High-Temperature Auger Electron Spectrometer Setup and Its Application to Reactive Wetting Experiments at 1700 K“, *Journal of Materials Science* **2015**, *50*, 3175–3182, ISSN: 0022-2461, 1573-4803, DOI 10.1007/s10853-015-8879-2 (siehe S. 10, 13, 15, 93).

[30] L. Alphei, R. Grotjahn, C. Dobbe, M. Douvidzon, R. Janhsen, T. Gebensleben u. a., „Relating Wetting and Reduction Processes in the Si-Liquid/SiO2-Solid Interface“, *Journal of Crystal Growth* **2015**, *419*, 165–171, ISSN: 00220248, DOI 10.1016/j.jcrysgro.2015.03.003 (siehe S. 10, 177).

[31] C. Elliott, V. Vijayakumar, W. Zink, R. Hansen, „National Instruments LabVIEW: A Programming Environment for Laboratory Automation and Measurement“, *JALA: Journal of the Association for Laboratory Automation* **2007**, *12*, Publisher: SAGE Publications Inc, 17–24, ISSN: 1535-5535, DOI 10.1016/j.jala.2006.07.012 (siehe S. 14).

[32] D. Weiß, T. Gebensleben, L. Diestel, L. Alphei, V. Becker, J. A. Becker, „The Influence of Crystallographic Orientation on the Wetting of Silicon on Quartz Single Crystals“, *Journal of Materials Science* **2011**, *46*, 3436–3444, ISSN: 0022-2461, 1573-4803, DOI 10.1007/s10853-010-5246-1 (siehe S. 24, 177).

[33] C. Dobbe, R. Grotjahn, T. Gebensleben, L. D. Alphei, V. Becker, J. A. Becker, „Reactive Wetting Controlled by Very Small Vertical Temperature Gradients in a Chemical Transport Mini Reactor“, *Zeitschrift für Physikalische Chemie* **2017**, *232*, DOI 10.1515/zpch-2017-0963, ISSN: 2196-7156, 0942-9352 (siehe S. 24, 177).

[34] D. Wachsmuth, T. Gebensleben, D. Weiß, V. Becker, L. D. Alphei, J. A. Becker, „SiO Gas Emission and Triple Line Dynamics of Small Silicon Droplets on Quartz“, *Journal of Crystal Growth* **2012**, *355*, 122–128, ISSN: 00220248, DOI 10.1016/j.jcrysgro.2012.06.044 (siehe S. 27, 177).

[35] G. Raynald, P. Hovongton, D. Drouin, P. Horny, H. Demers, A. Réal Couture, *CASINO*, Version 3.3, **2016** (siehe S. 46, 58).

[36] E. Gamma, R. Helm, R. E. Johnson, J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*, 1st ed., Reprint, Prentice Hall, Reading, Mass, **1994**, ISBN: 978-0-201-63361-0 (siehe S. 65, 67).

[37] G. Bradski, „The OpenCV Library“, *Dr. Dobb's Journal of Software Tools* **2000**, 122–125 (siehe S. 67).

[38] The GIMP Development Team, *GIMP*, Version 2.8 (siehe S. 82).

[39] W. McKinney in Proceedings of the 9th Python in Science Conference, **2010**, S. 51–56 (siehe S. 82).

[40] E. Hecht, *Optik*, 6., verbesserte, De Gruyter Oldenbourg, Berlin ; Boston, **2014**, ISBN: 978-3-11-034796-8 (siehe S. 82).

[41] S. C. Jain, S. K. Agarwal, W. N. Borle, S. Tata, „Total Emissivity of Silicon at High Temperatures“, *Journal of Physics D: Applied Physics* **1971**, *4*, 1207–1209, ISSN: 00223727, DOI 10.1088/0022-3727/4/8/323 (siehe S. 83).

[42] P. J. Timans, „Emissivity of Silicon at Elevated Temperatures“, *Journal of Applied Physics* **1993**, *74*, 6353–6364, ISSN: 0021-8979, 1089-7550, DOI 10.1063/1.355159 (siehe S. 83).

[43] E. Takasuka, E. Tokizaki, K. Terashima, S. Kimura, „Emissivity of Liquid Silicon in Visible and Infrared Regions“, *Journal of Applied Physics* **1997**, *81*, 6384–6389, ISSN: 0021-8979, 1089-7550, DOI 10.1063/1.364418 (siehe S. 83).

[44] N. M. Ravindra, B. Sopori, O. H. Gokce, S. X. Cheng, A. Shenoy, L. Jin u. a., „Emissivity Measurements and Modeling of Silicon-Related Materials: An Overview“, **2001**, 19 (siehe S. 83).

[45] J. M. Toguri, L. M. Pidgeon, „HIGH-TEMPERATURE STUDIES OF METALLURGICAL PROCESSES: PART I. THE THERMAL REDUCTION OF MAGNESIUM OXIDE WITH SILICON“, *Canadian Journal of Chemistry* **1961**, *39*, 540–547, ISSN: 0008-4042, 1480-3291, DOI 10.1139/v61-065 (siehe S. 93, 94).

[46] H. Oschinski, I. Kesuma, T. Gebensleben, J. A. Becker, „Structures and Thermodynamics of MgO/SiO Interfaces“, *The Journal of Physical Chemistry C* **2020**, *124*, 1923–1931, DOI 10.1021/acs.jpcc.9b08608 (siehe S. 177).

Danksagung

Ich möchte Herrn Professor Becker herzlich für die hoch engagierte Betreuung und das Interesse an meiner Arbeit danken. Die Gelegenheit, zu jeder möglichen und unmöglichen Uhrzeit ein klärendes Gespräch zu führen oder Experimente zu diskutieren, hat mir sehr geholfen.

Ebenso möchte ich Herrn Professor Grabow und Herrn Professor Renz für die Bereitschaft danken, als Prüfer zur Verfügung zu stehen.

Mein Dank gilt ebenfalls allen Mitarbeitern der Arbeitsgruppen von Herrn Professor Becker und Herrn Professor Grabow und ganz besonders Verena Becker, Tobias Alznauer und Lukas Alphei für die Unterstützung und den anregenden fachlichen Austausch.

Für die freundliche Unterstützung bei der Elektronenmikroskopie durch die Arbeitsgruppe von Herrn Professor Feldhoff und hierbei insbesondere durch Herrn Steinbach, sowie durch das Institut für Werkstoffkunde und hierbei insbesondere durch Herrn Heidenblut, möchte ich mich ebenfalls bedanken.

Danke Mareike, für deine Liebe und deine Geduld.

Mein größter, liebevoller Dank gilt meiner Mutter Heike, die schon immer felsenfest davon überzeugt war, dass ich eines Tages eine Dissertation einreichen werde, die Fertigstellung dieser Arbeit aber leider nicht mehr miterleben kann.

Lebenslauf

Name, Geburtsdatum und -ort	Tim Gebensleben, geboren am 14.05.1986 in Hannover.
August 1992 bis Juli 1996	Besuch der Grundschule Grasdorf in Laatzen.
August 1996 bis Juli 1998	Besuch der Orientierungsstufe der Albert Einstein Schule in Laatzen.
August 1998 bis Juli 2005	Besuch des Gymnasiums der Albert Einstein Schule in Laatzen mit Abschluss Abitur.
November 2005 bis Dezember 2005	Zivildienst, Agnes Karll Krankenhaus Laatzen.
Januar 2006 bis April 2006	Tätigkeit in der Datenerfassung, Agnes Karll Krankenhaus Laatzen.
Oktober 2006 bis März 2007	Studium im Fach Biophysik, Humboldt-Universität Berlin.
Oktober 2007 bis Oktober 2011	Bachelorstudium im Fach Chemie. Abschlussarbeit zum Thema „Dynamische reaktive Benetzung“.
Oktober 2011 bis Oktober 2014	Masterstudium im Fach Materialchemie und Nanochemie. Abschlussarbeit zum Thema „Simulation des Sauerstofftransports an Si(l)/SiO_2(s)-Grenzflächen mit einem Markov-Ketten-Algorithmus“.
Oktober 2014 bis Oktober 2019	Wissenschaftlicher Mitarbeiter am Institut für Physikalische Chemie und Elektrochemie der Leibniz Universität Hannover. Forschungsarbeiten im Bereich der Hochtemperaturchemie zur Erlangung des Grades Dr. rer. nat., gleichzeitig Bachelorstudium der Informatik.
Seit März 2020	Anstellung als Informatiker im Bereich maschinelles Lernen und künstliche Intelligenz im Unternehmen MARX KRONTAL PARTNER.

Publikationsliste

D. Weiß, T. Gebensleben, L. Diestel, L. Alphei, V. Becker, J. A. Becker, „The Influence of Crystallographic Orientation on the Wetting of Silicon on Quartz Single Crystals“, *Journal of Materials Science* **2011**, *46*, 3436–3444, ISSN: 0022-2461, 1573-4803, DOI 10.1007/s10853-010-5246-1

D. Wachsmuth, T. Gebensleben, D. Weiß, V. Becker, L. D. Alphei, J. A. Becker, „SiO Gas Emission and Triple Line Dynamics of Small Silicon Droplets on Quartz“, *Journal of Crystal Growth* **2012**, *355*, 122–128, ISSN: 00220248, DOI 10.1016/j.jcrysgro.2012.06.044

L. Alphei, R. Grotjahn, C. Dobbe, M. Douvidzon, R. Janhsen, T. Gebensleben, T. Alznauer, V. Becker, J. Becker, „Relating Wetting and Reduction Processes in the Si-Liquid/SiO2-Solid Interface“, *Journal of Crystal Growth* **2015**, *419*, 165–171, ISSN: 00220248, DOI 10.1016/j.jcrysgro.2015.03.003

C. Dobbe, R. Grotjahn, T. Gebensleben, L. D. Alphei, V. Becker, J. A. Becker, „Reactive Wetting Controlled by Very Small Vertical Temperature Gradients in a Chemical Transport Mini Reactor“, *Zeitschrift für Physikalische Chemie* **2017**, *232*, DOI 10.1515/zpch-2017-0963, ISSN: 2196-7156, 0942-9352

H. Oschinski, I. Kesuma, T. Gebensleben, J. A. Becker, „Structures and Thermodynamics of MgO/SiO Interfaces“, *The Journal of Physical Chemistry C* **2020**, *124*, 1923–1931, DOI 10.1021/acs.jpcc.9b08608

T. Gebensleben, V. Becker, J. A. Becker, „Transient Light Emission from the Silicothermic Reduction of Magnesium Oxide with Potential for Monitoring Intermediate Compound Formation and Decay“, *SN Applied Sciences* **2020**, *2*, 401, ISSN: 2523-3971, DOI 10.1007/s42452-020-2126-4