Fachbereich Mechatronik und
Maschinenbau

Hochschule Bochum
Bochum University
of Applied Sciences

Enhancing Computer Vision with Cloud Computing

Entwicklungsprojekt

Autor: Nagiev Emil

Matrikelnummer: 018105207
Prüfer: Prof. Dr.-Ing. Daniel Schilberg

04.10.2023

1. Inhaltsverzeichnis

2. Einleitung

Die Objekterkennung spielt eine wesentliche Rolle in vielen modernen Anwendungen der heutigen Industrie, wie beispielsweise der Erkennung fehlerhafter Produktion auf dem Laufband oder der Positionsbestimmung von Mitarbeitern in der Smart Factory. Sie ermöglicht es, bestimmte Objekte in Bildern oder Videos automatisch zu identifizieren und zu klassifizieren. Die Entwicklung von leistungsstarken und zuverlässigen Objekterkennungssystemen stellt jedoch nach wie vor eine Herausforderung dar, insbesondere wenn es um die Unterscheidung verschiedener Formen und Strukturen geht.

Das Ziel dieses Projekts ist es, eine Objekterkennungslösung zu entwickeln, die in der Lage ist, zwischen Dreiecken, Rechtecken und Kreisen zu unterscheiden. Dabei wird OpenCV, eine beliebte und leistungsstarke Open-Source-Bibliothek für Objekterkennung, als Grundlage für die Implementierung verwendet. Im Rahmen dieses Projekts werden wir uns auf die grundlegenden Konzepte der Objekterkennung konzentrieren und die Herausforderungen untersuchen, die einen Entwickler dabei immer verfolgen. Durch die Entwicklung einer robusten und effektiven Objekterkennungslösung werden wir in der Lage sein, Dreiecke, Rechtecke und Kreise präzise zu erkennen und zu unterscheiden. Dies kann als grundlegendes Konzept für fortgeschrittene Modelle betrachtet werden, die in verschiedenen Anwendungen von großem Nutzen sein können, sei es in der industriellen Automatisierung, der Bildverarbeitung oder der Robotik. Die vorliegende Dokumentation dient dazu, den Prozess der Entwicklung und Implementierung der Objekterkennungslösung detailliert zu beschreiben und wertvolle Einblicke in die Möglichkeiten und Herausforderungen der Objekterkennung mit ComputerVision zu geben.

Ein weiteres Thema dieses Projekts ist die Frage der Ressourcennutzung.

Der Wachstum der Cloud Computing ist kolossal. Die Unternehmen entscheiden sich anstatt die „traditionelle“ on-premisses Lösungen zu nutzen, für Cloud-Solutions.

Wie es im Abbildung 1 zu sehen ist, hat den klassischen op-premisses Typ der IT workload fast 23% der Markt verloren.

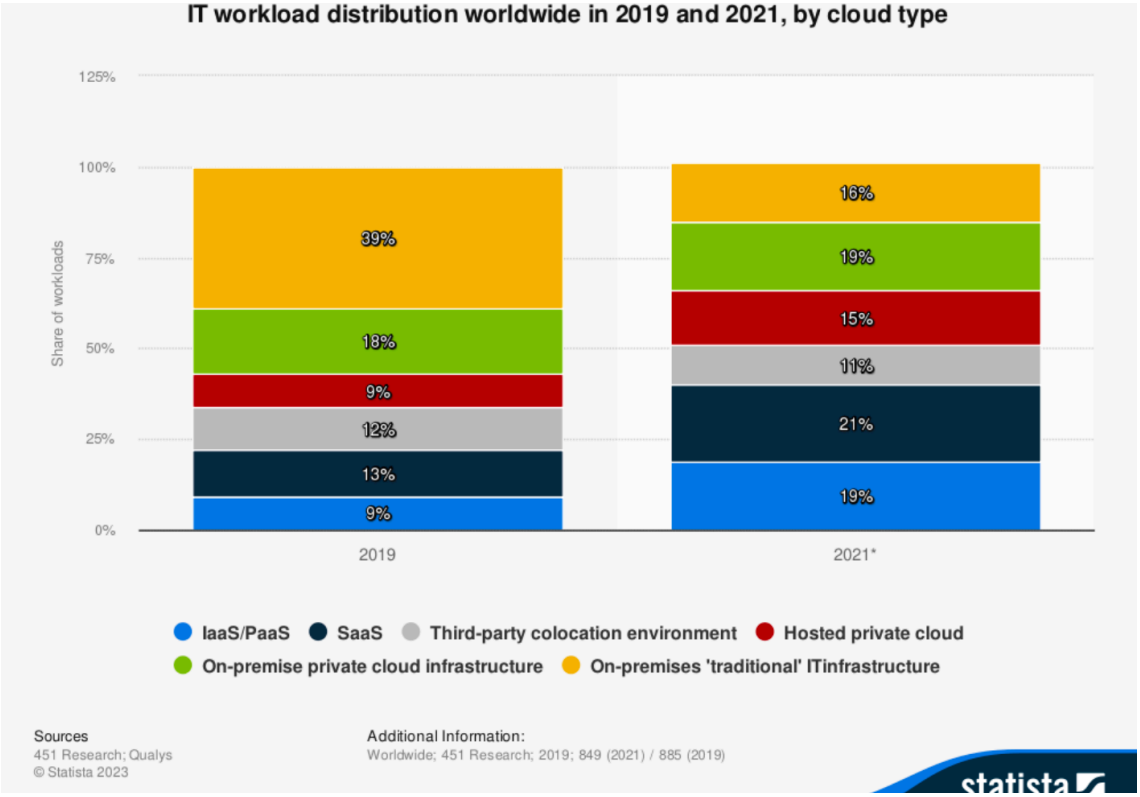

Abbildung 1 - IT workload weltweit in den Jahren 2019 und 2021 nach Cloud-Typ

Jetzt stellt sich die Frage, welche Unternehmen an der Spitze des Marktes für Cloud-Computing-Dienstleistungen stehen?

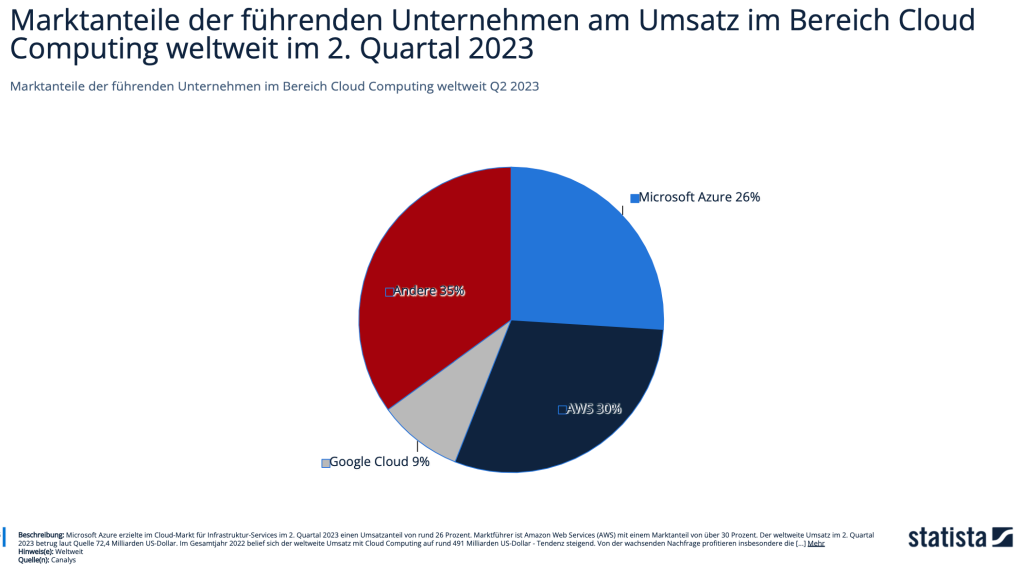

Abbildung 2 - Marktverteilung im Bereich Cloud Computing

Wie die Abbildung 2 zeigt, sind auf dem Markt drei große Unternehmen:

- AWS von Amazon
- GCP von Google
- Azure von Microsoft

Diese Unternehmen dominieren den globalen Markt für Cloud Computing-Dienste und bieten eine breite Palette von Lösungen für Unternehmen und Organisationen weltweit an. Wie Abbildung 3 zeigt, wächst der Umsatz in diesem Bereich exponentiell. Im Jahr 2010 betrug der Umsatz weniger als 3 Milliarden US-Dollar, während für das Jahr 2023 ein erwarteter Umsatz von bereits 150 Milliarden US-Dollar prognostiziert wird

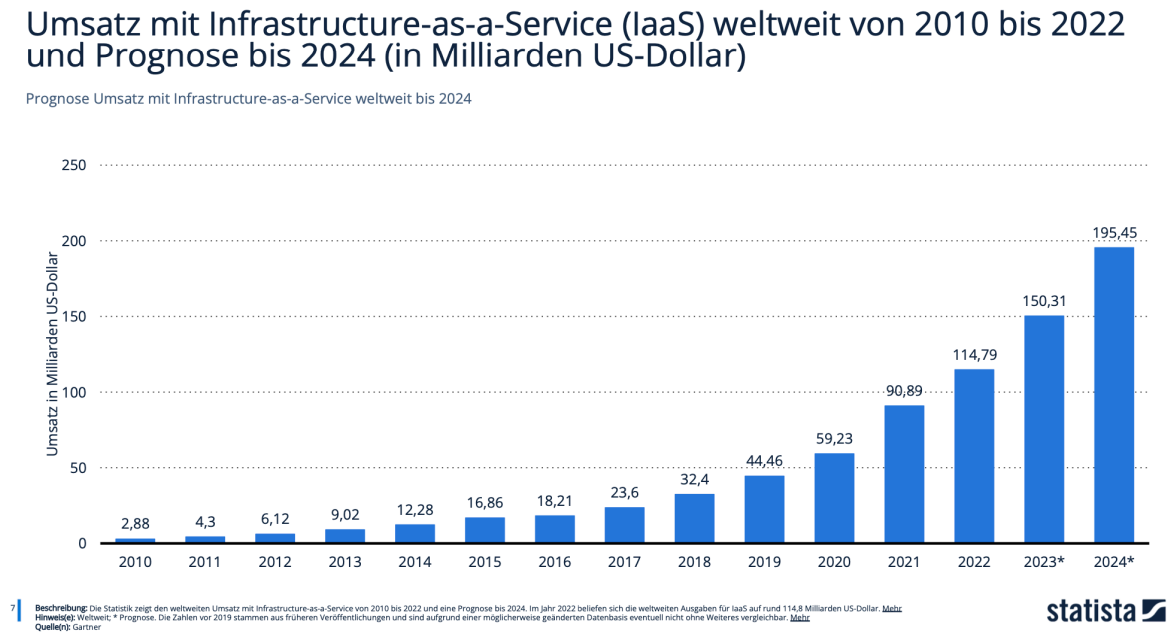

Abbildung 3 - Umsatz von Cloud-Diensten

Da in der Industrie immer häufiger das Konzept des sogenannten Cloud Computing verwendet wird, wird dies ebenfalls Gegenstand dieser Betrachtung sein.

3. Grundlagen

3.1 Computer Vision

OpenCV ist eine Open-Source Bibliothek, die als standard Tools für Computer Vision verwendet wird. Hauptsächlich handelt es sich um eine kostenlose Softwarebibliothek für die Bildverarbeitung. Sie bietet eine Menge von Funktionen und Algorithmen zur Verarbeitung und Analyse von Bildern und Videos.

Näher dazu in den Kapitell "Modellentwicklung" OpenCV unterstützt verschiedene Programmiersprachen wie C++, Java und Python.

3.2 Azure - Cloud Computing

In der letzen Decade ist die Datenmenge, die von Unternehmen konsumiert wird, rasant angestiegen. Dadurch stellt sich die Frage, wo und wie die Daten gehostet werden sollen?

Azure hat weltweit regional verteilte Rechenzentren aufgebaut und bieten ihre Dienstleitung unter dem Konzept 'Pay-as-you-go' an. Das bedeutet, dass Unternehmen nur für die tatsächlich genutzten Ressourcen bezahlen. Das heißt, es ist nicht mehr notwendig, einen rechenstarken PC zu haben, um die Bilder verarbeiten zu können. Die Entwickler können die Ressourcen der Cloud nutzen. Für die Cloud-Engineers wurde von HashiCorp ein weiteres flexibles und leistungsfähiges Toll namens 'Terraform' entwickelt. Terraform gehört zu dem Konzept 'IaS', so genanten 'Infrastructure as Code', bei dem die Infrastruktur mittels einer deskriptive Programmiersprache beschrieben wird. Im Rahmen dieses Projekts werden wir uns mit Azure mit Hilfe des Terraform auseinandersetzen.

4. Cloud-Architektur Entwurf

4.1 Lokaler Rechner als Steuergerät für Cloud Computing

Um mit Azure arbeiten zu können, müssen die Developer einem einfachen Schema folgen.

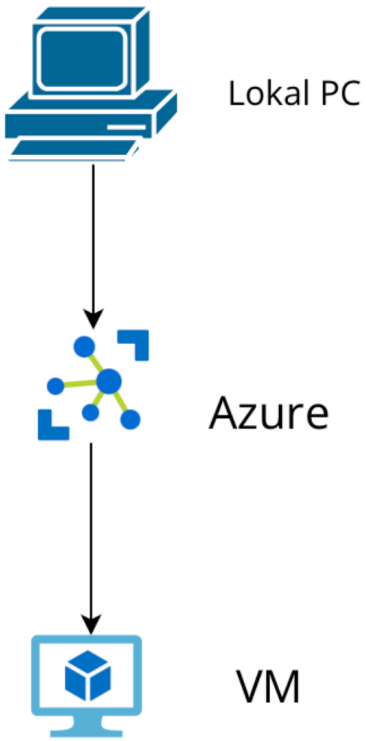

Abbildung 4: Nutzungsstruktur cloudbasierter virtueller Maschinen

Alle Ressourcen, die erstellen werden sollen, müssen einer bestimmten Resource group zugeordnet werden. Unter Ressourcen versteht man in Azure die Services, die von Azure als Dienstanbieter angeboten werden.

Für dieses Projekt werden wir folgenden Ressourcen benötigen:

- Subscription
- Resource group
- Network Interface
- Publik IP Adress
- Blob Storage
- Virtual Machine

4.2 infrastructure as Code

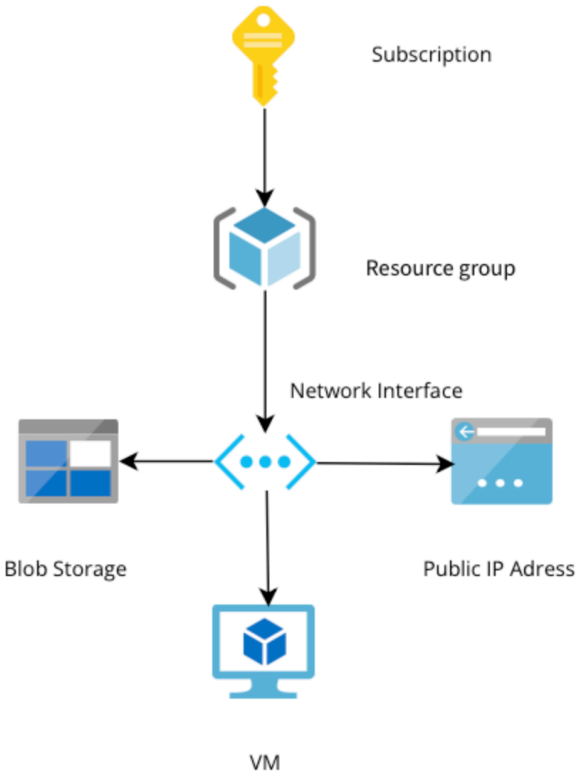

Abbildung 5: Cloud Architektur in Azure

Deployment von Architektur wird mit Hilfe von Terraform erschafft.

In Abbildung 6 wird folgenden Ressourcen konfiguriert:

• **required_providers**: Definiert die benötigten Terraform-Provider. In diesem Fall wird der Provider **azurerm** verwendet, um mit Microsoft Azure zu interagieren. Die Version des Providers wird auf mindestens 3.0.2 festgelegt.

• **required_version**: Gibt an, dass mindestens Terraform-Version 1.0.4 erforderlich ist, um dieses Konfigurationsfile auszuführen.

• **provider „azurerm":** Dieser Block konfiguriert den Azure-Provider. Es wird die subscription_id angegeben, die die eindeutige Kennung für Ihr Azure-Abonnement ist. In diesem Fall ist es "c4c1dd3f-4f33-487a-ac21-e8c5f83b1c58".

• **features**: In diesem Fall wird als einfacher Platzhalter verwendet.

• **skip_provider_registration**: Dies ist auf **true** gesetzt, was bedeutet, dass Terraform die Überprüfung und Registrierung des Providers in Azure-Abonnement überspringt, da die Registrierung bereits manuell vorgenommen ist.

```terraform
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0.2"
    }
  }

  required_version = ">= 1.0.4"
}

provider "azurerm" {

  subscription_id = "c4c1dd3f-4f33-487a-ac21-e8c5f83b1c58"
  features {

  }
  skip_provider_registration = true
}
```

Abbildung 6: Codeblock für Terraform Provider

In der Abbildung 7 wird Ressource Group konfiguriert.

• **azurerm_resource_group**: Dieser Block erstellt eine Azure Resource Group mit dem Namen "lime" in der Region "West Europe".

```terraform
resource "azurerm_resource_group" "lime" {

   name     = "lime"
   location = "West Europe"
 }
```

Abbildung 7: Codeblock für Resource Group

In Abbildung 8 wird Storage Account konfiguriert.

• **name:** Der Name des Storage Accounts wird auf "reslimesa01" gesetzt.

• **resource_group_name:** Die Resource Group, zu der dieser Storage Account gehört, wird aus der Eigenschaft **azurerm_resource_group.lime.name** abgeleitet. Dies bedeutet, dass dieser Storage Account unter der Resource Group „lime“erstellt wird

• **location:** Die Region, in der dieser Storage Account erstellt wird, wird ebenfalls aus der Eigenschaft **azurerm_resource_group.lime.location** übernommen. Das heißt, beide Ressourcen werden in einem Rechnerzentrum gehostet.

• **account_tier:** Der Kontotyp des Storage Accounts. Hier ist es "Standard", was bedeutet, dass der Storage Account die Standardleistung bietet.

• **account_replication_type:** Der Replikationstyp des Storage Accounts. Hier ist es "LRS" (Locally Redundant Storage), was bedeutet, dass die Daten lokal in einer Region repliziert werden.

```terraform
resource "azurerm_storage_account" "lime-sa01" {
  name                     = "reslimesa01"
  resource_group_name      = azurerm_resource_group.lime.name
  location                 = azurerm_resource_group.lime.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```

Abbildung 8: Codeblock für Storage Account

In Abbildung 9 wird Storage Container konfiguriert.

• **name:** Der Name des Storage Containers wird auf "rescn-sink" gesetzt.

• **storage_account_name:** Der Name des Storage Accounts, zu dem dieser Container gehört, wird aus der Eigenschaft **azurerm_storage_account.lime-sa01.name** abgeleitet. Dies bedeutet, dass dieser Container im zuvor erstellten "lime-sa01" Storage Account liegt.

• **container_access_type:** Der Zugriffstyp für den Container. Hier ist es "private", was bedeutet, dass der Container standardmäßig privat ist.

```terraform
resource "azurerm_storage_container" "lime-sc-source" {
  name                  = "rescn-sink"
  storage_account_name  = azurerm_storage_account.lime-sa01.name
  container_access_type = "private"
}
```

Abbildung 9: Codeblock für Storage Container

In der Abbildung 10 wird Blob Storage Account konfiguriert.

• **name:** Der Name des Blobs wird auf "blob-sink" gesetzt.

• Die Werte für '**storage_account_name**' und '**storage_container_name**' werden identisch wie zuvor durch die Variablenreferenzierung abgelesen.

• **type:** Der Blob-Typ wird auf "Block" gesetzt, was bedeutet, dass es sich um einen Block-Type handelt, der für größere Datenmengen verwendet wird.

```terraform
resource "azurerm_storage_blob" "lime-blob0-source" {
  name                   = "blob-source"
  storage_account_name   = azurerm_storage_account.lime-sa01.name
  storage_container_name = azurerm_storage_container.lime-sc-source.name
  type                   = "Block"
  #source                 = "some-local-file.zip"
}
```

Abbildung 10: Codeblock für Blob Storage

In Abbildung 11 wird die **Virtual Machine** (VM) konfiguriert. Da die **VM** von mehreren anderen Ressourcen abhängig ist, werden auch das **azurerm_virtual_network**, das **azurerm_subnet** und **azurerm_network_interface** erstellt. Alle diese Ressourcen befinden sich in derselben Ressourcengruppe, und alle Variablen werden durch Referenzierung abgeleitet

```
resource "azurerm_virtual_network" "lime-vn" {
  name                = "reslime-network"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.lime.location
  resource_group_name = azurerm_resource_group.lime.name
}
resource "azurerm_subnet" "lime-sn" {
  name                 = "reslimesninternal"
  resource_group_name  = azurerm_resource_group.lime.name
  virtual_network_name = azurerm_virtual_network.lime-vn.name
  address_prefixes     = ["10.0.2.0/24"]
}
resource "azurerm_network_interface" "lime-ni" {
  name                = "reclimeni"
  location            = azurerm_resource_group.lime.location
  resource_group_name = azurerm_resource_group.lime.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.lime-sn.id
    private_ip_address_allocation = "Dynamic"
  }
}
resource "azurerm_windows_virtual_machine" "lime-vm" {
  name                = "reslimevm"
  resource_group_name = azurerm_resource_group.lime.name
  location            = azurerm_resource_group.lime.location
  size                = "Standard_F2"
  admin_username      = "adminuser"
  admin_password      = "E654550e#"
  network_interface_ids = [
    azurerm_network_interface.lime-ni.id,
  ]
  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }
  source_image_reference {
    publisher = "MicrosoftWindowsServer"
    offer     = "WindowsServer"
    sku       = "2016-Datacenter"
    version   = "latest"
  }
}
```

Abbildung 11: Codeblock für Virtual Machine

Anmeldung in Azure

Um den Zugriff auf das Cloud Ressourcen zu haben, es ist notwendig lokalen PC ins Azure zu identifizieren. Dafür werden wir Azure CLI verwenden.

Azure CLI ist eine Befehlszeilenschnittstelle, über das Entwickler eine Verbindung mit Cloud herstellen und Verwaltungsbefehle für Azure-Ressourcen ausführen können.

- Azure CLI instalieren

Shell

```
brew update && brew install azure-cli
```

- mit Azure CLI anmelden

Shell

```
az login
```

- Subscription ID zuweisen

Shell

```
az account set --subscription "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Abbildung 12: Anmeldung in Azure durch Azure CLI

Als Bestätigung erhaltet man folgende Nachricht in Terminal:

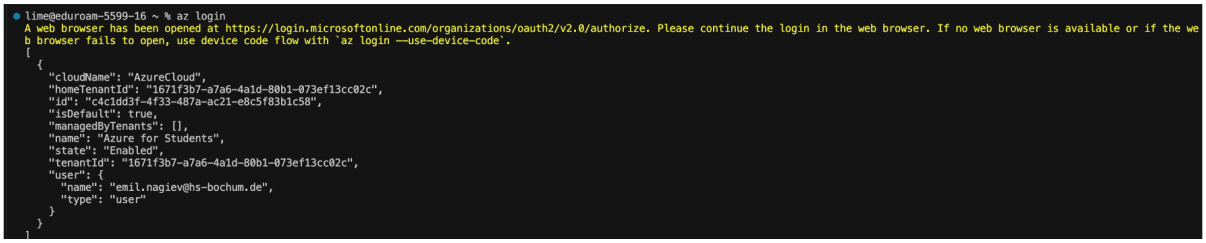

Abbildung 13: Anmeldebestätigung in dem Terminal

Infrastruktur erstellen

Um die geplante Infrastruktur zu erstellen, wird in Terraform zwei grundsätzliche bash-Befehle verwendet:

Mit `**terraform plan**` lässt sich der bereits angesprochene Ablaufplan erstellen, damit kann der Entwickler ablesen, welche Ressourcen erstellt, geändert oder zerstört werden.

Mit ` **terraform apply**` kann die Infrastruktur deployed werden.

```terraform
terraform plan

terraform apply
```

Abbildung 14: Codeblock für Deployment

5. Modellentwicklung

5.1 Implementierung der Funktionen

Grundsätzlich muss der Entwickler bei allen Bildverarbeitungsprozessen einfachem Vorgehen folgen.

- Preprocessing-Schritte:
  - Das Bild laden
  - Das Bild in eine graufarbige Ebene umwandeln
  - Das Bild mit einem Filter verschärfen
- Objekterkennug:
  - Das vorbereitete Bild analysieren
  - Die Ergebnisse anzeigen lassen

Da wir die OpenCV-Bibliotheken nutzen, werden wir die integrierte Funktionalität von OpenCV verwenden.

- `**cv2.imread()**`
  - Funktion zum Bildlesen
- - `**cv2.ctvColor()**`
  - Funktion zur Grauwertumwandlung
- - `**cv2.GaussianBlur()**`
  - Gaussian-Weichzeichnung
- - `**cv2.Canny()**`
  - Funktion zur Kantenerkennung

Zusätzlich muss noch eine Funktion zum Kantenerkennung geschrieben werden

- `**get_contours()**`

Mit der Funktion `cv2.approxPolyDP()` wird die Anzahl der Kanten gezählt, um anschließend durch die if-else-Statements richtige Entscheidungen zu treffen.

Jetzt muss man genauer Untersuchen, was eigentlich hinter die verwendeten CV-Funktionen steckt.

- **`cv2.ctvColor()`**

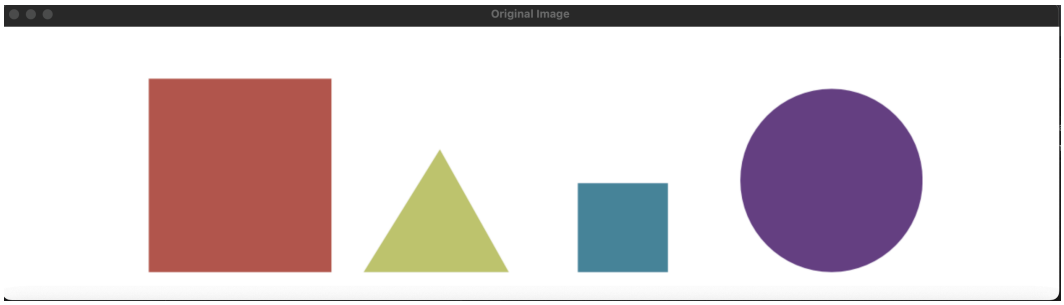

Abbildung 15: Eingangsbild

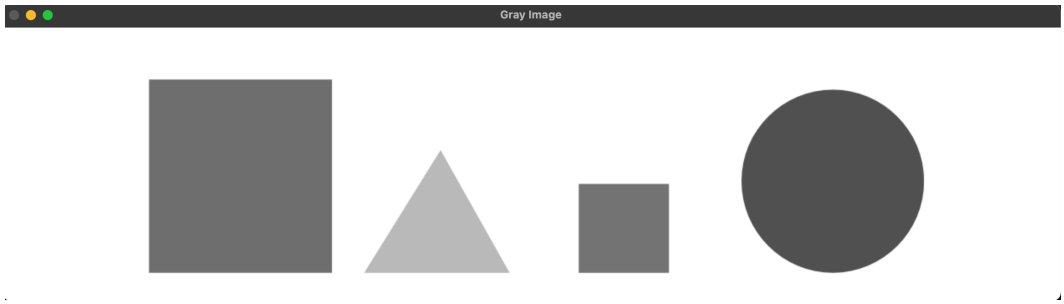

Abbildung 16: Bild in Graustufenformat

Die Funktion wandelt das Eingangsbild von einem Farbbildformat(RGB) in das Graustufenformat um. Als Ergebnis wird ein neues Bild erhalten, wessen Intensität den Graustufenwert eines Pixels im Originalbild repräsentiert. Dies ermöglicht die Verarbeitung und Analyse von Bildern basierend auf den Intensitätsunterscheidung anstelle der Farbinformationen.

- `**cv2.GaussianBlur()**`

Abbildung 17: Gebluertes Bild

Das Eingangsbild wird mit einem Gauß-Filter verarbeitet, um die Rauschreduktion und die Glättung der Kanten zu erreichen. Als Parameter soll die Größe des Kernels und ein Wert für Standartabweichung angegeben werden. Das Ergebnis ist wieder ein neues Bild, in dem das Raschen reduzieret und die Kanten weicher gemacht werden, um eine bessere Grundlage für die Kantenanalyse zu erschaffen.

Funktionalität von der Gaussian blur - Funktion kann als Faltungmatrix interpretiert werden:

1. Ursprüngliche Pixel: Die Zahlenwerte repräsentieren die ursprünglichen Pixelwerte des Bildes.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

2. Gaußian Kernel: Kann als korrespondierenden Werten für die Berechnung interpretiert werden.

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix}$$

3. Neu Pixel: Als Ergebnis wird ein neue Matrix erhalten.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Mathematisch gesehen passiert Folgendes: Jeder Wert des ursprünglichen Bildes wird mit dem entsprechenden Wert des Gauß'schen Kernels multipliziert, und schließlich werden alle Elemente addiert.

- `cv2.Canny()`

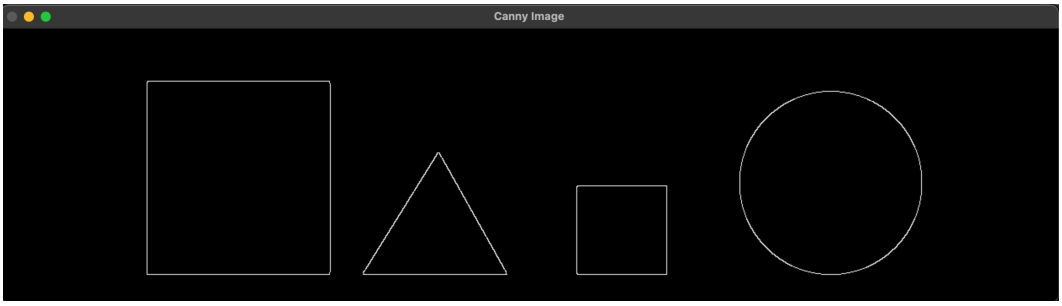

Abbildung 18: Kantenkarte des Bildes

Der Canny-Algorithmus nutzt die Gradienteninformationen des Bildes, um starke Kanten zu identifizieren und schwache Kanten durch Hysterese zu verfolgen. Das Ergebnis ist eine binäre Kantenkarte (`imageCanny`), bei der die Kanten als weiße Pixel markiert sind, während der Rest des Bildes schwarz ist. Diese Kantenkarte wird zur weiteren Verarbeitung und Visualisierung der Kanten verwendet.

6. Anwendung

6.1 Vorbereitung von VM

Sobald die virtuelle Maschine auf dem Azure-Server gestartet ist, können wir von unserem lokalen Rechner aus darauf zugreifen.

SSH-Tunnel aufbauen:

```
ssh -L 8080:localhost:8888 lime@20.160.227.224 jupyter notebook --no-browser
```

PowerShell

Abbildung 19: SSH-Tunnel zur VM

Hierbei sind '**lime@20.168.227.224**' der Kontoname und die IP-Adresse der gestarteten VM. Da wir nicht die 'klassische' Variante von Jupyter Notebook verwenden möchten, wird der Prefix --no-browser explizit angegeben.

Im Rahmen dieses Projekts wird VSCode als Entwickungsumgebung verwendet.

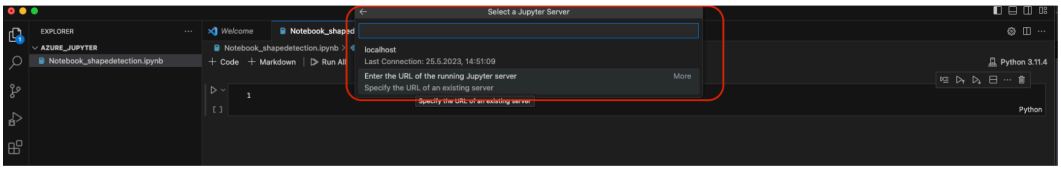

Abbildung 20: Python Kernel

VSCode ermöglicht es, laufende Python-Kernel zu nutzen, anstatt lokale Rechenleistungen zu verwenden. Jetzt sieht man das gewöhnliche Layout von Jupyter Notebook in unserer VSCode-Umgebung. Jetzt kann man entwickelte Modell implementieren und testen.

6.2 Implementierung

In Abbildung 21 werden die benötigte Python-Bibliotheken importiert.

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

✓ 0.5s Python

Abbildung 21: Codeblock 1

• **findContours():** Diese Funktion findet die Konturen im gegebenen Bild. Das Ergebnis sind die Konturen selbst und eine Hierarchie der Konturen.

• **contourArea():** Diese Funktion berechnet die Fläche der Kontur. Dadurch kann die Größe des erkannten Objekts bestimmt werden.

• **arcLength():** Diese Funktion berechnet den Umfang der Kontur und überprüft, ob die Kontur geschlossen ist.

• **approxPolyDP():** Hier wird die Anzahl der Ecken in der Kontur reduziert, um den Typ des Objekts zu bestimmen. Ein Dreieck hat beispielsweise drei Ecken, ein Rechteck vier.

• **Bestimmung des Objekttyps ():** Anhand der Anzahl der Ecken wird der Typ des Objekts bestimmt (Dreieck, Rechteck, Kreis oder "None", wenn das Objekt nicht erkannt wird).

• **boundingRect():** Diese Funktion erstellt ein Rechteck um das erkannte Objekt. Markierung des erkannten Objekts:

• **rectangle**(): Ein Rechteck wird um das erkannte Objekt gezeichnet.

• **putText():** Dann wird der erkannte Objekttyp auf das Bild geschrieben, wobei der Text aus der Variablen **objectType** entnommen wird.

```python
def get_contours(image):
    contours, hierarchy = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    for cnt in contours:
        area = cv2.contourArea(cnt)  #Diese Funktion wird verwendet, um die Fläche des Umrisses zu finden.
        print("Area: ", area)
        cv2.drawContours(imageContour, cnt, -1, (255,0,0), 3)  #Konturen zeichnen
        perimeter = cv2.arcLength(cnt, True)  #Überprüfen, ob die Konturen geschlosen sind
        print("Perimeter: ", perimeter)
        approx = cv2.approxPolyDP(cnt, 0.02*perimeter, True)  #Anzahl der Konturen zu finden
        print("Corner Points: ", len(approx))
        objCorner = len(approx)
        x,y,w,h = cv2.boundingRect(approx)  #erkannte Objekte gekenzeichnen

        if objCorner == 3:
            objectType = 'Dreieck'

        elif objCorner == 4:
            objectType = 'Rechteck'

        elif objCorner > 4:
            objectType = 'Creis'

        else:
            objectType = "None"

        cv2.rectangle(imageContour, (x, y), (x+w, y+h), (0, 255, 0), 2)  #Ein Rechteck wird um die identifizierten Objekte gezeichnet
        cv2.putText(imageContour, objectType, (x+(w//2)-10, (y - 20)), cv2.FONT_HERSHEY_COMPLEX, 0.5 , (0, 0, 0), 2)  #Beschriftung hinzufuegen
```

✓ 0.0s Python

Abbildung 22: Codeblock 2

In Abbildung 23 wird das Bild eingelesen mit der Funktion **imread()**.

```python
#image = cv2.imread("source/flaschendeckel.jpeg")
image = cv2.imread("source/shapes_v2.png")
imageContour = image.copy()

```

✓ 0.0s Python

Abbildung 23: Codeblock 3

In Abbildung 24 werden die Preprocessing-Funktionen, die in Kapitel 5 beschrieben sind, aufgerufen, und anschließend wird das Bild an die Funktion **get_contour** weitergegeben.

```python
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image_blur = cv2.GaussianBlur(image_gray, (7, 7), 1)
imageCanny = cv2.Canny(image_blur, 50, 50)
get_contours(imageCanny)
```

✓ 0.0s Python

Abbildung 24: Codeblock 4

In Abbildung 25 werden die einzelne Schritten angezeigt und beschriftet.

```python
# Alle Bilder werden aufeinander angezeigt.
cv2.imshow("Original Image", image)
cv2.imshow("Gray Image", image_gray)
cv2.imshow("Blur Image", image_blur)
cv2.imshow("Canny Image", imageCanny)
cv2.imshow("Contour Image", imageContour)
cv2.imwrite("source/shapeOutput.png", imageContour)
cv2.waitKey(0)
cv2.destroyAllWindows()

titles = ['original image','image in gray scale', 'canny image ', 'countour image']
images = [image, image_gray, imageCanny, imageContour]

```

Python

Abbildung 25: Codeblock 4

6.3 Testdurchlauf

Als Ergebnis erhalten wir vier Bilder:

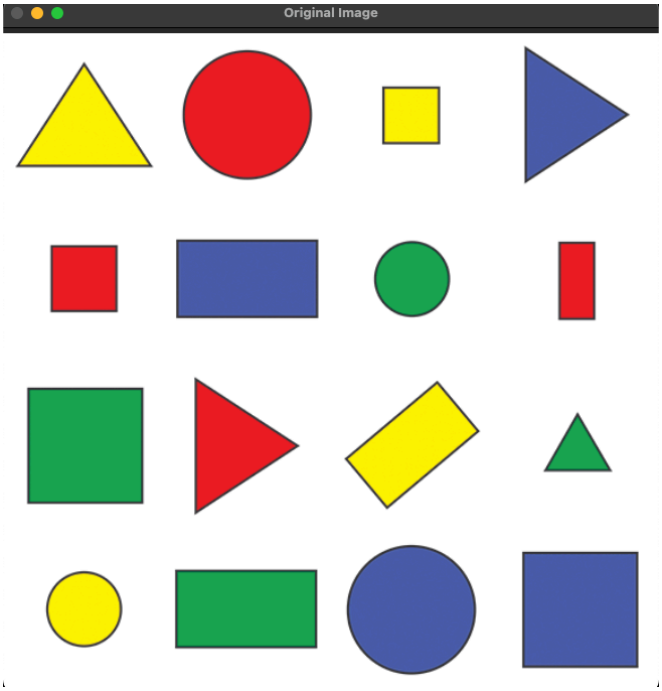

Abbildung 26: Eingangsbild

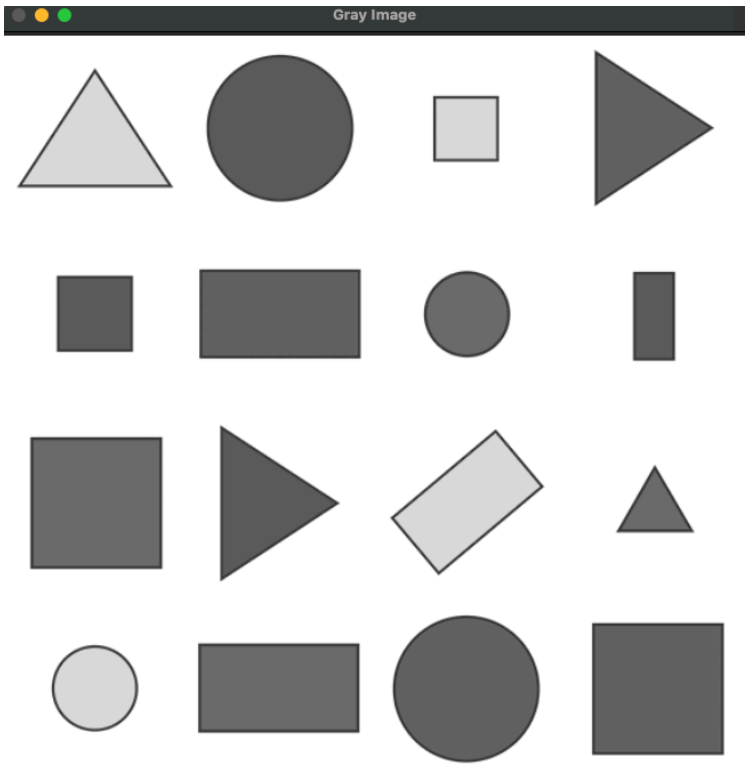

Abbildung 27: Das Bild in Graustufenform

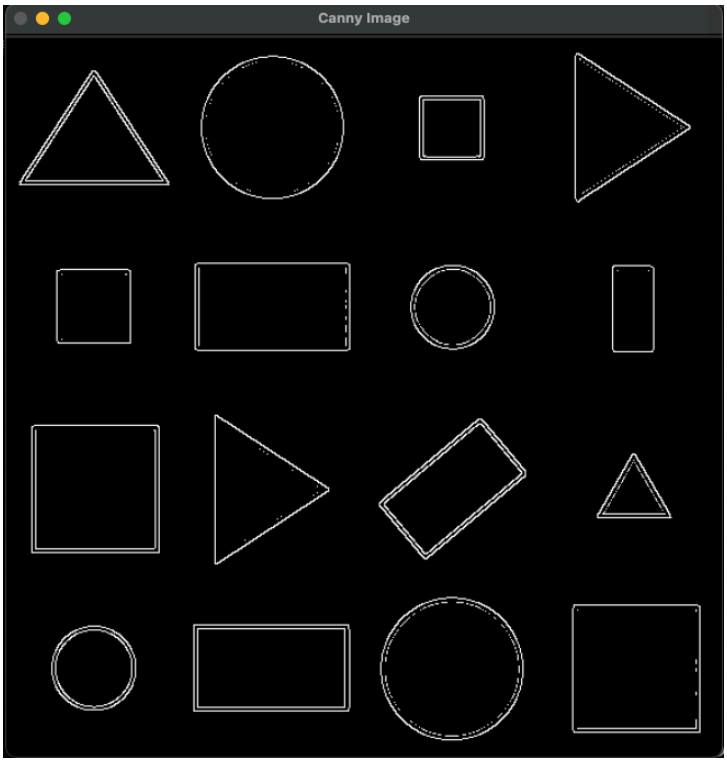

Abbildung 28: Kantenkarte des Bildes

Das letze Bild zeigt deutlich, dass alle geometrischen Figuren erfolgreich erkannt wurden.

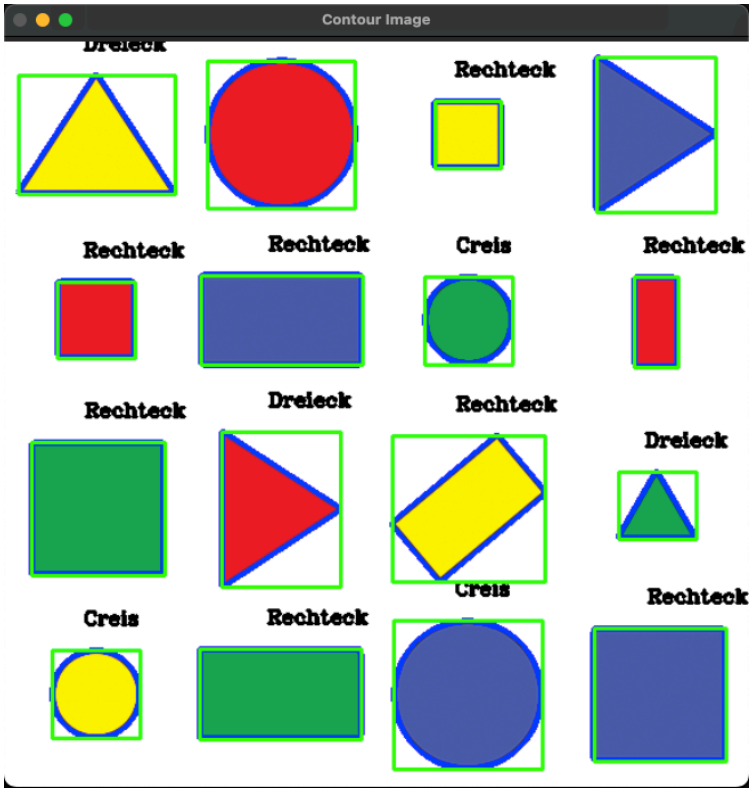

Abbildung 29: Ausgangsbild

Wenn aber ein Quadrat von einem einfachem Rechteck unterscheidet werden muss, wird es nicht funktionieren.

Als Beispiel wird ein anderes Bild untersucht.

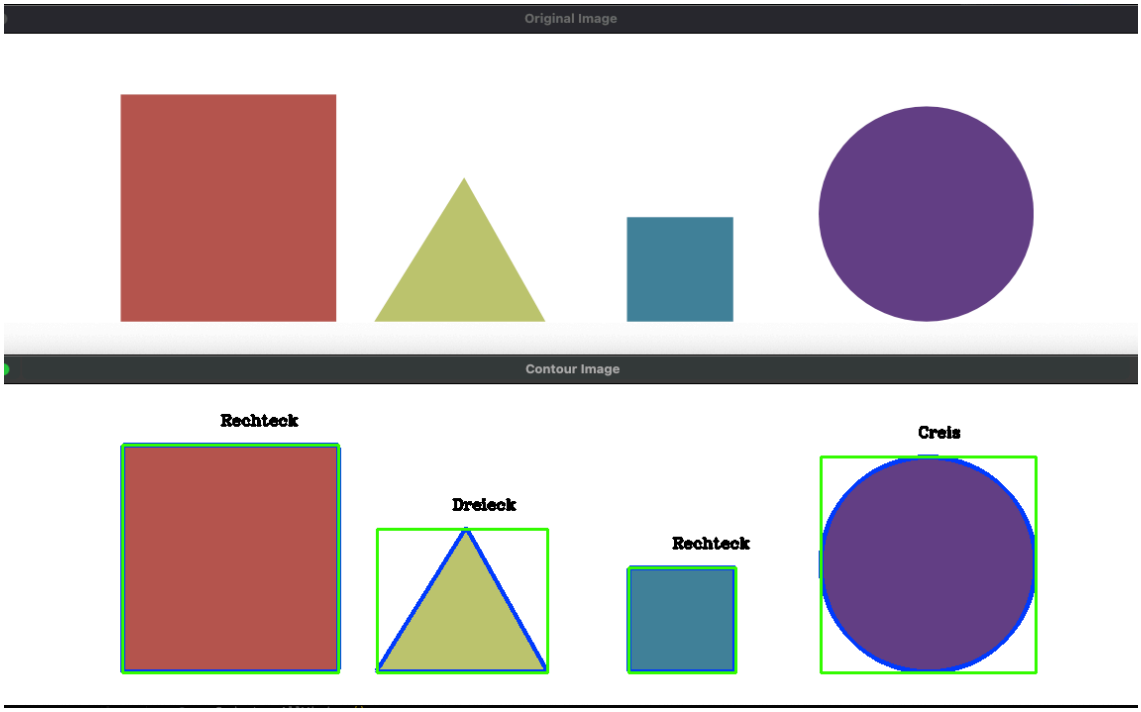

Abbildung 30: Eingangsbild 2

Eine weitere mögliche Funktion könnte in diesem Fall implementiert werden, nämlich die Unterscheidung zwischen Rechteck und Quadrat. Dazu könnte die Entscheidungslogik erweitert werden. Da Quadrat gleich lange Seiten hat, könnte diese Eigenschaft in der Funktion **get_contours()** implementiert werden. Eine Erweiterung sehen wir in Abbildung 31.

```python
elif objCorner == 4:
    aspectRatio = float(w)/float(h)
    if aspectRatio > 0.95 and aspectRatio < 1.05:
        objectType = 'Quadrat'
else:
    objectType = "Rechteck"
```

Abbildung 31: Feature code

6.4 Auswertung der Ergebnissen

Wie Abbildung 32 zeigt, wurden alle festgestellten Ziele erreicht. Mit den implementierten Funktionen wurden alle geometrischen Figuren mithilfe des OpenCV-Framework erfolgreich erkannt. Der Algorithmus kann Dreiecke, Rechtecke, Quadrate und Kreise voneinander unterscheiden.

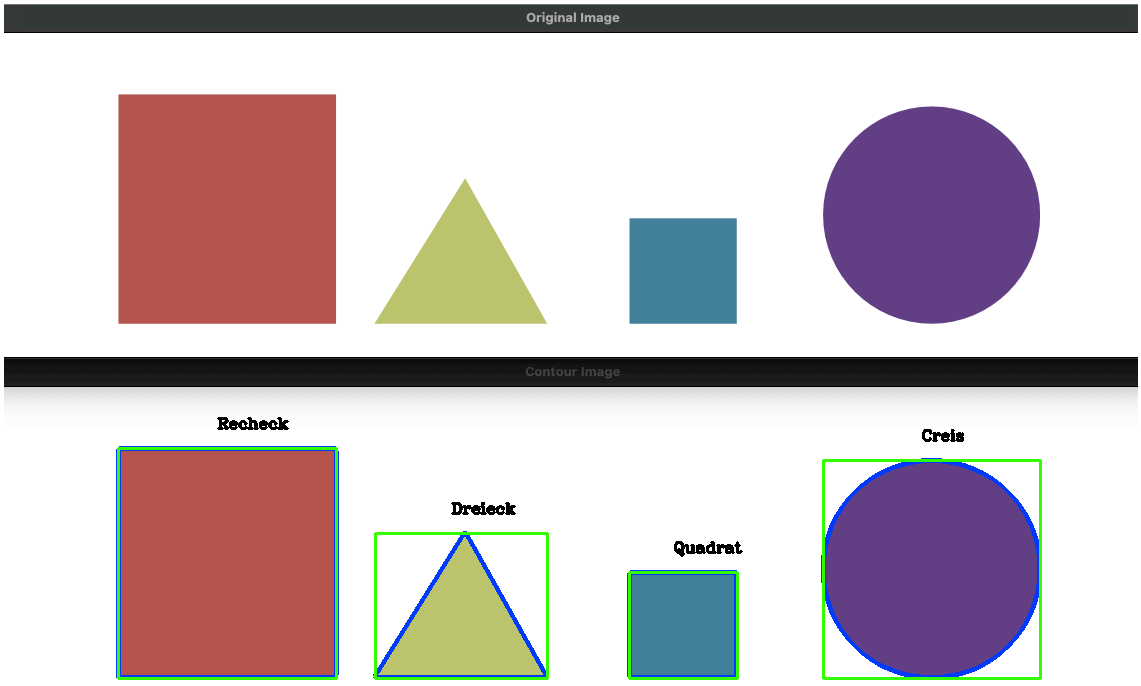

Abbildung 32: Auswertung des 2. Bild

7.Fazit

Im Rahmen dieses Entwicklungsprojekt haben wir uns mit der einigen grundlegenden Techniken von Bildverarbeitung auseinandergesetzt. Um die geometrische Objekte auf dem Bild erkennen zu können, war die interne Funktionalität von dem open-source Framework OpenCV verwendet worden. Die geometrische Figuren waren Anhang des implementierten Algorithmus erfolgreich erkennt. Außerdem wurden im Züge dieses Projekt die Fragen von Verwendung Cloud-Computing betrachtet. Als Ergebnis ist die vollständige Architekturmuster entworfen und die benötigten Ressourcen in Azure Cloud unter Verwendung des studentische Subscription deployed. Die Verwendung von Cloud Computing bietet eine hervorragende Möglichkeit, die Rechenleistung leicht zu skalieren. Dies hat für Unternehmen, die je nach Bedarf im Produktionsworkflow auf zusätzliche Hardware zugreifen können, oberste Priorität.

Da Source Control System ein unvermeidbare Teil des DevOps-Konzeptes ist, kann der gesamte Entwicklungsprozess im Git-Repository nachverfolgt werden. - https://github.com/veiganlime/shapedetection

8. Literaturverzeichnis

1. Offizielle Dokumentation von OpenCV, URL: https://docs.opencv.org/4.x/ (Stand: 01.09.2023)
2. Offizielle Dokumentation von Hashicorp, URL: https://developer.hashicorp.com/terraform?product_intent=terraform (Stand: 01.09.2023)
3. Offizielle Dokumentation von Terraform, URL: https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources (Stand: 01.09.2023)
4. Sunila Gollapudi. Learn Computer Vision using OpenCV. Apress ISBN-13 (electronic): 978-1-4842-4261-2
5. D. Sundararajan. Digital Image Processing. Springer ISBN 978-981-10-6113-4
6. Diagram Tool, URL: https://online.visual-paradigm.com/de/diagrams/features/azure-architecture-diagram-tool/
7. Statistik-Report zum Thema Cloud Computing, URL: https://de.statista.com/statistik/studie/id/22297/dokument/cloud-computing-statista-dossier/ (Stand: 04.10.2023)
8. Umfrage zu IT-Services, URL: https://www.statista.com/statistics/1321441/cloud-and-on-premises-data-storage-for-data-engineering-worldwide/ (Stand: 04.10.2023)