BEISPIEL 2 - MATLAB BASICS

VU Einführung in Visual Computing, TU Wien, SS 2021 04.03.2021

1 Übersicht

Das zweite Beispiel besteht aus 4 Teile: Im ersten Teil ist es Ziel sich mit Grundlagen von MATLAB und der MATLAB-Hilfe vertraut zu machen und eigene Versionen von built-in MATLAB Funktionen zu implementieren. Im zweiten Teil dieses Übungsbeispiels soll die Theorie bzgl. Graphikpipelines und Objektrepräsentation praktisch umgesetzt werden. Der dritte Teil macht Sie mit den grundlegenden Kenntnissen im Umgang mit Bildern und Filtern in MATLAB vertraut und der vierte Teil repräsentiert ein praktisches Anwendungsbeispiel bzgl. der Theorie der Transformationen.

Bewertung	
MatlabBasics.m :	5 Punkte
Triangles.m :	5 Punkte
Images.m :	5 Punkte
Transformations.m :	5 Punkte
Maximal erreichbare Punkte :	**20 Punkte**

Abzugeben ist eine ZIP-Datei via TUWEL, die folgende 4 Dateien beinhaltet:
MatlabBasics.m, Triangles.m, Images.m, Transformations.m

1.1 Allgemeine Informationen zur Abgabe

Es ist sicherzustellen, dass MATLAB beim Ausführen der abgegebenen Dateien keine Fehlermeldungen ausgibt, das Skript abstürzt oder ähnliches. **Wir können keinen fehlerhaften bzw. nicht ausführbaren Code bewerten! Dateinamen, Methodensignaturen und die Bezeichnungen von vorgegebenen Variablen dürfen auf keinen Fall verändert werden!** Ergebnisse müssen in den vorgegebenen Variablen abgespeichert werden! Ein Missachten dessen kann dazu führen, dass für Teilaufgaben keine Punkte vergeben werden.
Es liegt in Ihrer Verantwortung, rechtzeitig **vor der Deadline** zu kontrollieren, ob Ihre Abgabe funktioniert hat. **Kontrollieren Sie bitte Ihre Abgabe bzgl. folgender Punkte:**

- Werden die hochgeladenen Dateien angezeigt?
- Können Sie die Dateien herunterladen und öffnen oder im Browser anzeigen?
- Haben Sie die richtigen Dateien abgegeben?

Sollte der Upload Ihrer Dateien nicht funktionieren, können Sie uns via TUWEL-Forum kontaktieren oder eine E-Mail an **evc@cg.tuwien.ac.at** senden.

1.2 Allgemeine Hinweise

MATLAB-Hilfe verwenden: Um mehr Informationen zu einem MATLAB-Befehl zu erhalten, tippe Sie *help MATLAB-Befehl* in das Command Window von MATLAB. Alternativ dazu kann der Textcursor über dem *help MATLAB-Befehl* positioniert und “F1” gedrückt werden.

Debuggen:

- **Breakpoints setzen:** Setzt man einen Breakpoint in einem MATLAB-Script und führt es dann aus, so kann man im „Workspace“ des Hauptfensters auf alle bis zu diesem Breakpoint erzeugten Variablen zugreifen, sie ausgeben oder verändern.
- **Variablen ausgeben:** Lässt man am Ende einer Zeile/Berechnung in einem MATLAB-Script das Semikolon weg, so wird das Ergebnis der entsprechenden Zeile/Berechnung beim Ausführen des Scripts automatisch im „Command Window“ des Hauptfensters ausgegeben. Bitte nach Möglichkeit das Semikolon vor dem Abgeben wieder hinzufügen.

Internet & Tutorials verwenden: Es gibt jede Menge an MATLAB Tutorials im Internet, speziell wenn es um Debugging geht. Ein Beispiel hierfür wäre **Debug a MATLAB Program**[1] von MathWorks.

Hier noch ein Hinweis: Der **MATLAB Primer**[2] enthält einige nützliche Informationen.

[1] Debug a MATLAB Program:
https://de.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html

[2] MATLAB Primer: https://de.mathworks.com/help/matlab/getting-started-with-matlab.html

2 Basics

In der ersten MATLAB-Aufgabe sollen Sie sich mit ein paar Grundlagen von MATLAB, sowie der MATLAB-Hilfe vertraut machen und gleichzeitig ein wenig Vektorrechnung bzw. Matrizenrechnung wiederholen. Sie werden dieses Wissen bei den weiteren Beispielen und auch beim Test benötigen.

Laden Sie die Datei MatlabBasics.m! Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren in dieser Datei wieder. Die Stellen, an denen Sie Ihre Lösung implementieren sollen, sind jeweils mit einem *TODO* markiert. Seien Sie sich bei der Implementierung der folgenden Aufgaben stets sicher, dass Sie genau wissen, was in welcher Zeile passiert. **Beim Abgabegespräch müssen Sie Ihren Code im Detail erklären können um Punkt dafür zu bekommen. Auch wenn Ihre abgegebene Lösung zu 100% richtig ist, bekommen Sie erst die Punkte wenn Sie diese einwandfrei erklären können.**

2.1 Basis-Datenstrukturen in MATLAB erzeugen

1. Erzeugen Sie einen Zeilen-Vektor v_1 (1) und einen Spalten-Vektor v_2 (2) mit jeweils 3 Werten, sowie eine Matrix *M* (3) mit 3 Zeilen und 3 Spalten. Entnehmen Sie dabei die Werte für Vektoren und Matrix Ihrer Matrikelnummer (8-stellig) in folgender Reihenfolge: Angenommen Ihre Matrikelnummer lautet ABCDEFGH, so sei:

$$v_1 = \begin{bmatrix} D & A & C \end{bmatrix} \tag{1}$$

$$v_2 = \begin{bmatrix} F \\ B \\ E \end{bmatrix} \tag{2}$$

$$M = \begin{bmatrix} D & B & C \\ B & G & A \\ E & H & F \end{bmatrix} \tag{3}$$

   Setzen Sie für ABCDEFGH die Ziffern Ihrer Matrikelnummer ein! Editieren Sie die mit *TODO* markierten Zeilen entsprechend.

2. Erzeugen Sie aus der Matrix *M* eine Sequenz, die von der kleinsten Ziffer Ihrer Matrikelnummer bis zur größten Ziffer Ihrer Matrikelnummer im Intervall 0.25 läuft! Verwenden Sie für diese Aufgabe den Sequenz-Operator ':' und die Funktionen *min* und *max*. **Verwenden Sie direkt die Matrix M! Beispiel:** Die Matrikelnummer 01233120 soll folgenden Vektor v_3 (4) erzeugen:

$$v_3 = \begin{bmatrix} 0.0 & 0.25 & 0.5 & 0.75 & 1.0 & 1.25 & 1.5 & 1.75 & 2.0 & 2.25 & 2.0 & 2.25 & 2.5 & 2.75 & 3.0 \end{bmatrix} \tag{4}$$

   Um mehr Informationen zur Verwendung des Sequenz-Operators zu erhalten, tippen Sie *help :* ins Command Window von MATLAB.

3. Erzeugen Sie eine Matrix $M_{15\times 9}$ (5) mit 15 Zeilen und 9 Spalten! Die Matrix soll wie ein Schachbrett aufgebaut sein, dessen weiße Felder mit 0 gefüllt sind und dessen schwarze Felder mit der Matrix *M* gefüllt sind. Entsprechend ist ein weißes bzw. schwarzes Feld 3×3 Zellen groß. Dabei sollen die Ecken des Schachbrettes schwarz sein (d.h. 3×3 Zellen mit der Matrix gefüllt).

   Wichtig: Dieser Teil des Beispiels (das Erstellen der fertigen Schachbrett-Matrix) soll direkt auf die Matrix *M* zugreifen und dabei maximal 7 Zuweisungsoperationen verwenden. Es dürfen keine Schleifen oder Rekursionen verwendet werden.

$$M_{15\times 9} = \begin{bmatrix} D & B & C & 0 & 0 & 0 & D & B & C \\ B & G & A & 0 & 0 & 0 & B & G & A \\ E & H & F & 0 & 0 & 0 & E & H & F \\ 0 & 0 & 0 & D & B & C & 0 & 0 & 0 \\ 0 & 0 & 0 & B & G & A & 0 & 0 & 0 \\ 0 & 0 & 0 & E & H & F & 0 & 0 & 0 \\ D & B & C & 0 & 0 & 0 & D & B & C \\ B & G & A & 0 & 0 & 0 & B & G & A \\ E & H & F & 0 & 0 & 0 & E & H & F \\ 0 & 0 & 0 & D & B & C & 0 & 0 & 0 \\ 0 & 0 & 0 & B & G & A & 0 & 0 & 0 \\ 0 & 0 & 0 & E & H & F & 0 & 0 & 0 \\ D & B & C & 0 & 0 & 0 & D & B & C \\ B & G & A & 0 & 0 & 0 & B & G & A \\ E & H & F & 0 & 0 & 0 & E & H & F \end{bmatrix} \tag{5}$$

   Als Zuweisungsoperation gilt jeder Befehl, in dem = (ist-gleich) als Zuweisung, und nicht als Vergleich vorkommt.

   Hinweis: Auf den Mathworks-Seiten über **Matrix-Indexing**[3] und **Array-Manipulation**[4] finden Sie einige nützliche Befehle. (Tipp: *repmat*)

2.2 Implementieren von Built-In Funktionen

Implementiere Sie eigene Versionen der folgenden built-in MATLAB-Funktionen: *****, **.***, **cross** und **dot**.

ACHTUNG: Es ist nicht erlaubt, die Funktionen *cross*, *dot*, *mtimes* oder '.*' zum Lösen dieser Aufgabe zu verwenden, sie sollen selbst implementiert werden. Außerdem darf der '*'-Operator nur verwendet werden, um zwei einzelne Werte miteinander zu multiplizieren. Es dürfen zum Lösen dieser Aufgabe Schleifen oder Rekursionen verwendet werden.

Hinweis: Um Ihre Lösung auf Korrektheit zu überprüfen, können Sie einfach Ihr Resultat mit dem Resultat vergleichen, das Ihnen die entsprechende built-in MATLAB-Funktion liefert.

[3] Matrix-Indexing: http://de.mathworks.com/help/matlab/math/matrix-indexing.html
[4] Array-Manipulation: https://de.mathworks.com/help/matlab/matrices-and-arrays.html

1. Implementieren Sie die Funktion *dotProduct()*, die das Skalarprodukt von zwei 3-Element-Vektoren berechnet! Die Funktion soll das Ergebnis *result* zurückliefern.

2. Implementieren Sie die Funktion *crossProduct()*, die das Kreuzprodukt zweier 3-Element-Vektoren berechnen soll! Die Funktion soll das Ergebnis als Zeilen-Vektor *result* zurückliefern.

3. Implementieren Sie die Funktion *vector_X_Matrix()*, die einen 3-Element-Vektor von links mit einer 3×3-Matrix multipliziert! Die Funktion soll das Ergebnis als Zeilen-Vektor *result* zurückliefern.

4. Implementieren Sie die Funktion *Matrix_X_vector()*, die einen 3-Element-Vektor von rechts mit einer 3×3-Matrix multipliziert! Die Funktion soll das Ergebnis als Spalten-Vektor *result* zurückliefern.

5. Implementieren Sie die Funktion *Matrix_X_Matrix()*, die zwei 3×3-Matrizen miteinander multipliziert! Dabei soll die Matrix *M* von links auf *M2* multipliziert werden (*M* × *M2*). Die Funktion soll das Ergebnis *result* zurückliefern.

6. Implementieren Sie die Funktion *Matrix_Xc_Matrix()*, die zwei 3×3-Matrizen komponentenweise miteinander multipliziert! Das heißt, jeder Wert der Matrix *M* wird mit jenem Wert der Matrix *M2* multipliziert, der die gleichen Koordinaten hat. Die Funktion soll das Ergebnis *result* zurückliefern.

2.3 Hinweise für das Abgabegespräch

Beim Abgabegespräch müssen Sie Ihren Code erklären können und genau wissen, was die von Ihnen verwendeten Befehle tun. Sie müssen auch die Theorie hinter den nachprogrammierten Befehlen kennen (z.B. was ist ein dot-product, wofür wird es verwendet und was sagt es aus).

3 Triangles

Theorie in Vorlesung: *Graphikpipeline + Objektrepräsentation*

In dieser MATLAB-Aufgabe sollen Sie sich mit den wichtigsten Primitiven in der Computergrafik befassen - den Dreiecken. Dieses Wissen wird bei den weiteren Beispielen und auch beim Test benötigt.

Laden Sie die Datei Triangles.m. Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren im MATLAB-File wieder. Die Stellen, an denen Sie Ihre Lösung implementieren sollen, sind jeweils mit einem *TODO* markiert.

3.1 Triangles

1. Erzeugen Sie 3 Vektoren *P1* (6), *P2* (7) und *P3* (8), welche die 3 Punkte eines Dreiecks repräsentieren, mit folgenden Koordinaten:

$$P1 = \begin{bmatrix} (1+C) & -(1+A) & -(1+E) \end{bmatrix} \tag{6}$$

$$P2 = \begin{bmatrix} -(1+G) & -(1+B) & (1+H) \end{bmatrix} \tag{7}$$

$$P3 = \begin{bmatrix} -(1+D) & (1+F) & -(1+B) \end{bmatrix} \tag{8}$$

   Wobei ABCDEFGH wie im Beispiel 2.2-Basics die Matrikelnummer ist.

2. Erzeugen Sie nun die 3 Seiten-Vektoren des Dreiecks: *P1P2*, *P2P3* und *P3P1*.
   P1P2 soll von *P1* nach *P2* zeigen.
   P2P3 soll von *P2* nach *P3* zeigen.
   P3P1 soll von *P3* nach *P1* zeigen.

3. Berechnen Sie die Längen der Seiten des Dreiecks. Die Länge eines Vektors wird auch *euklidische Norm* eines Vektors genannt.

   Achtung: Die MATLAB-Funktion *norm* darf zur Lösung dieser Aufgabe nicht verwendet werden, da ihre Funktionalität nachimplementiert werden soll. Sie können aber Ihre Ergebnisse mit den Ergebnissen, die Ihnen die norm-Funktion liefern würde, vergleichen.

4. Berechnen Sie den Normalvektor der Dreiecks-Fläche und den normalisierten Normalvektor! Speichern Sie die Ergebnisse in *normal* und *normalized_normal*. Zum Lösen dieser Aufgabe ist es nun auch erlaubt die *MATLAB*-Funktionen *dot*, *cross* und *norm* zu verwenden. **Achtung: Ein Normalvektor und ein normalisierter Vektor sind nicht das Gleiche!**

5. Berechnen Sie die Fläche des Dreiecks mit der Hilfe des Normalvektors und speichern Sie das Ergebnis in der Variable *Area*! Zum Lösen dieser Aufgabe ist es nun auch erlaubt die *MATLAB*-Funktionen *dot*, *cross* und *norm* zu verwenden.

   Hinweis: Hat die Richtung des Normalvektors hier einen Einfluss?

6. Berechnen Sie die 3 Winkel des Dreiecks in Grad! Nennen Sie die Winkel *alpha* beim Eckpunkt *P1*, *beta* bei *P2* und *gamma* bei *P3*. Built-in MATLAB-Funktionen dürfen zum Lösen dieser Aufgabe verwendet werden.

   Hinweis: Beachten Sie die Richtung der Vektoren! Verwenden Sie die MATLAB-Funktion *acosd*, um den Arkuskosinus in Grad zu berechnen. Überprüfen Sie Ihre Lösung! Ist die Summe Ihrer 3 errechneten Winkel korrekt? Speichern Sie die Summe der Winkel in *angles_sum*, das Maximum der Winkel in *angles_max* und das Minimum der Winkel in *angles_min*. Das arithmetische Mittel der Winkel speichern Sie in *angles_avg* (siehe Befehl *mean*).

3.2 Hinweise für das Abgabegespräch

Beim Abgabegespräch müssen Sie Ihren Code erklären können, und genau wissen, was die von Ihnen verwendeten Befehle tun. Sie müssen auch die Theorie hinter den nachprogrammierten Befehlen kennen (z.B. was ist ein dot-product, wofür wird es verwendet und was sagt es aus).

4 Images

Dieses Beispiel soll grundlegende Kenntnisse im Umgang mit Bildern und Filtern in MATLAB erläutern.

Laden Sie die Files Images.m und mandrill_color.jpg.

Wichtig: Beide Dateien müssen im selben Ordner liegen! Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren im MATLAB-File wieder. Die Stellen, an denen Sie Ihre Lösung implementieren sollen, sind jeweils mit einem *TODO* markiert.

4.1 Image Basics

Hinweis: Zum Lösen dieser Beispiele werden Funktionen der **Image Processing Toolbox**[5], wie beispielsweise *fspecial* benötigt. Funktionen auf den Mathworks-Seiten für **Read, Write, Display, and Modify Images**[6] könnten zusätzlich hilfreich sein.

1. Laden Sie ein Bild in MATLAB! Der Pfad ist in der Variable *input_path* gespeichert.
2. Konvertieren Sie das in Punkt 1 geladene Bild in double-Werte zwischen 0 und 1 (d.h. 0 wird auf 0 abgebildet, 255 wird auf 1 abgebildet)!

   Achtung: Für diese Aufgabe dürfen **keine Schleifen oder Rekursionen** verwendet werden.
3. Vertauschen Sie im Bild aus Punkt 2 den Rot- mit dem Grün-Kanal! Das Ergebnis soll in der Variable *image_swapped* abgelegt werden und so aussehen wie in Abbildung 1 a) dargestellt.

a) image_swapped b) image_masked

Abbildung 1: Beispiele für image_swapped und image_masked

4. Zeigen Sie das Bild aus Punkt 3 am Bildschirm an.

[5] Image Processing Toolbox: https://de.mathworks.com/help/images/index.html
[6] Read, Write, Display, and Modify Images: https://de.mathworks.com/help/matlab/images_btfntr_-1.html

5. Speichern Sie das Bild aus Punkt 3 im PNG-Format auf die Festplatte. Der Zielpfad ist in der Variable *output_path* gespeichert.

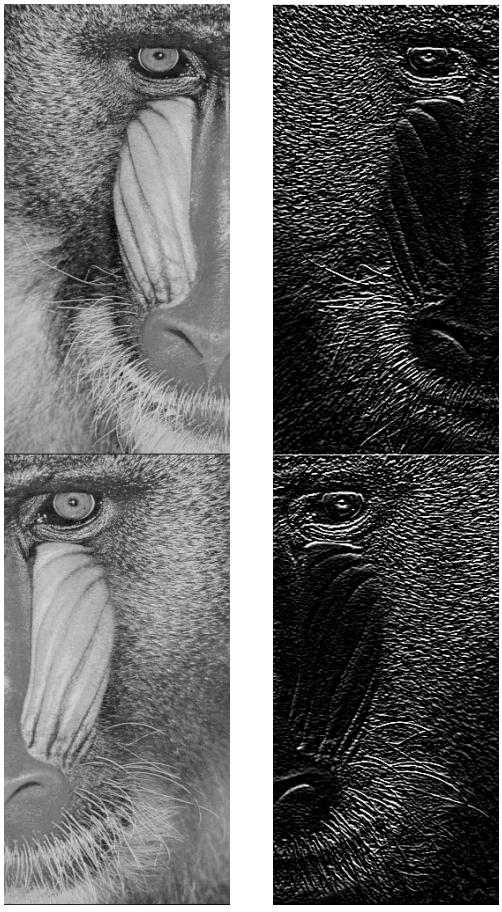
6. Erstellen Sie ein zwei-dimensionales Bild aus logical-Werten in dem jedes Pixel markiert ist, bei dem der Grün-Kanal im Bild aus Punkt 2 größer oder gleich 0.7 ist! Das Ergebnis soll in der Variable *image_mark_green* abgelegt werden.

   Achtung: Für diese Aufgabe dürfen **keine Schleifen oder Rekursionen** verwendet werden.

7. Setzen Sie im Bild aus Punkt 2 alle Pixel auf Schwarz, die im logical-Bild aus Punkt 6 markiert sind (also die Pixel, bei denen der Grün-Kanal im Bild größer-gleich 0.7 ist)! Das Ergebnis soll in der Variable *image_masked* abgelegt werden und so aussehen wie in Abbildung 1 b).

   Achtung: Für diese Aufgabe dürfen **keine Schleifen oder Rekursionen** verwendet werden. Hilfreiche Funktion: *repmat*

8. Wandeln Sie das Bild aus Punkt 2 in ein Graustufenbild um und transformieren Sie dieses von 512×512 nach 1024×256 (**ACHTUNG:** MATLAB adressiert Matrizen durch ”Zeile × Spalte”)! Dabei soll das Bild in der Mitte zerschnitten werden und der rechte Teil unten an den linken Teil angefügt werden. Das Ergebnis soll in der Variable *image_reshaped* abgelegt werden und so aussehen wie in Abbildung 2 a).



a) image_reshaped b) image_edge

Abbildung 2: Beispiele für image_reshaped und image_edge

4.2 Filter und Faltungen

Theorie in Vorlesungen: *Local Operations, Edge Filtering*

1. Erstellen Sie mit *fspecial* einen Gauss-Filter der Größe 5 $\times$ 5 mit σ =2.0.

2. Implementieren Sie die Funktion *evc_filter*, welche ein RGB-Eingabebild (*input*) mit einem 5$\times$5 Filterkern (*kernel*) filtert. Für alle Pixel, die außerhalb des Bildes liegen, soll [0,0,0] angenommen werden. Das Ergebnis der Funktion soll in etwa so aussehen wie Abbildung 3.

Abbildung 3: image_convoluted

Achtung: Für diese Aufgabe **dürfen Schleifen oder Rekursionen verwendet werden**, jedoch nicht die Funktionen *imfilter*, *filter2* und *conv2*, oder ähnliche Funktionen die sich um die Faltung kümmern. **Hint:** Das Ausgabebild muss immer die gleiche Größe haben wie das Eingabebild.

3. Erstellen Sie ein Kantenbild, in dem nur die horizontalen Kanten des Bildes aus 4.1 Punkt 8 zu sehen sind. Hierzu soll ein Sobel-Filter verwendet werden, wie er in den Vorlesungsunterlagen zu finden ist. Für diese Aufgabe darf *imfilter* verwendet werden.

   Achtung: Für dieses Beispiel nicht *evc_filter* verwenden. Das Ergebnisbild soll die gleiche Größe wie das Eingabebild haben. Wie Pixel außerhalb des Bildes behandelt werden, ist Ihnen überlassen. Entscheide Sie sich aber für eine übliche Randbehandlung[7] (*zero padding*, *border replication*, ...) Das Ergebnis soll in der Variable *image_edge* abgelegt werden und in etwa so aussehen wie Abbildung 2 b).

   Hilfreiche Funktion: *imfilter*

[7] https://www.mathworks.com/help/images/imfilter-boundary-padding-options.html

4.3 Hinweise für das Abgabegespräch

Beim Abgabegespräch müssen Sie Ihren Code erklären können und genau wissen, was die von Ihnen verwendeten Befehle tun. Sie müssen auch die Theorie hinter den nachprogrammierten Befehlen kennen (z.B. was ist ein Gauß-Filter, wofür wird er verwendet und wie schaut eine Gauß-Kurve aus).

5 Transformationen

Theorie in Vorlesung: *Transformationen*

In diesem Beispiel sollen Sie sich mit den Grundlagen der Transformationen und Transformationsmatrizen bekannt machen.

Laden Sie das File Transformations.m! Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren im MATLAB-File wieder. Die Stellen, an denen Sie Ihre Lösung implementieren sollen, sind jeweils mit einem *TODO* markiert.

Vorlesungsunterlagen: In den Vorlesungsunterlagen sind weitere hilfreiche Informationen zu den Transformationen zu finden.

5.1 Transformationen und Plotten

1. Implementieren Sie die drei MATLAB-Funktionen *mtranslate*, *mrotate* und *mscale*, welche 3×3 Basis-Transformationsmatrizen zurückgeben sollen. Details zu den Parametern finden Sie im Kommentar zur jeweiligen Funktion. Hilfreiche Funktionen: *sind*, *cosd*

2. Vervollständigen Sie die Funktion *display_vertices*, sodass diese die Vertices anzeigt! Wenn diese Methode korrekt implementiert ist, sollte das erste Bild (Original Quad) wie in Abbildung 4 aussehen. Hilfreiche Funktion: *plot*

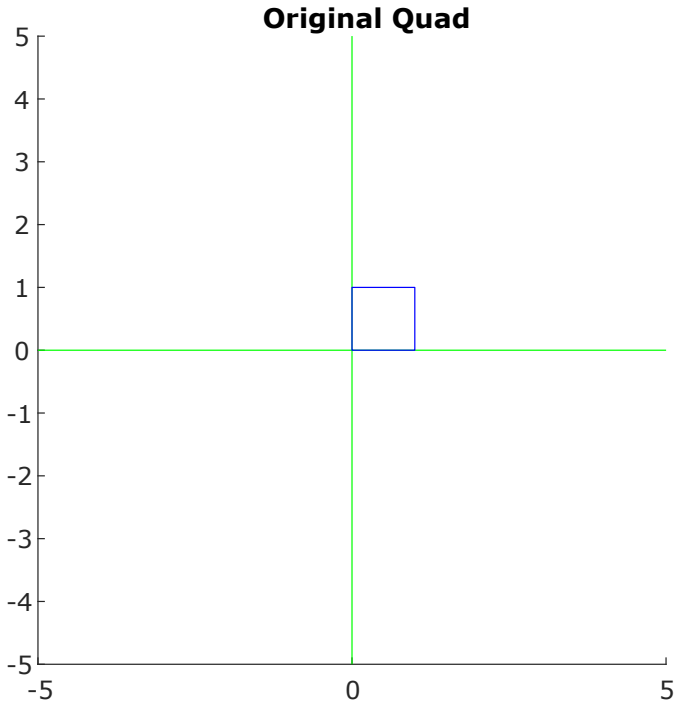


Abbildung 4: Original Quad

3. Vervollständigen Sie die Funktion *transform_vertices*, welche eine Liste von Vertices und eine Transformationsmatrix als Eingabe nimmt und jeden Vertex mit dieser Matrix transformiert! Wenn diese Methode korrekt implementiert ist, sollte das zweite Bild (Rotated Quad) wie in Abbildung 5 aussehen.

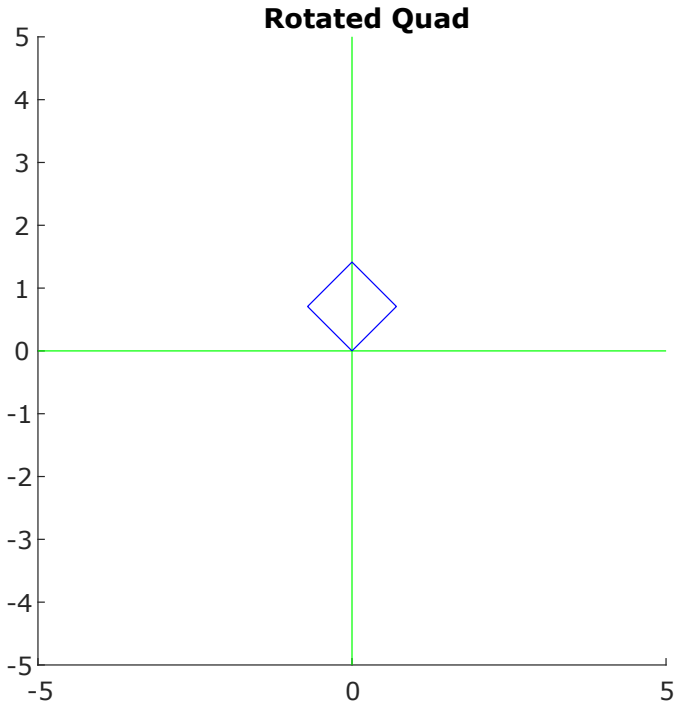


Abbildung 5: Rotated Quad

4. Benutzen Sie die zuvor implementierten Funktionen, um die folgenden vier Ziel-Bilder (siehe Abbildungen 6, 7, 8, 9) zu erstellen! Nutzen Sie die Matrix-Funktionen aus Punkt 1, um eine geeignete Transformation zu erstellen. Transformieren Sie anschließend die Vertices in quad und zeige n Sie diese an. Die jeweiligen Ergebnis-Vertices sollen in den Variablen *image1_vertices*, *image2_vertices*, *image3_vertices* und *image4_vertices* gespeichert werden.
   Wichtig: Die Funktion *transform_vertices* darf nur einmal pro Bild aufgerufen werden. Erstellen Sie also zuerst eine Matrix, die bereits alle Transformationen enthält und wenden Sie diese dann erst an!
   Tipp: Durch hovern über die Eckpunkte in den Plots kann deren Position angezeigt werden und mit den (auf zwei Kommastellen gerundeten) Zielwerten aus der Angabe verglichen werden.

5.2 Hinweise für das Abgabegespräch

Beim Abgabegespräch müssen Sie Ihren Code erklären können und genau wissen, was die von Ihnen verwendeten Befehle tun. Sie müssen auch die Theorie hinter den nachprogrammierten Befehlen kennen (z.B. warum braucht man bei 2D Punkten eine 3×3 Matrix)!

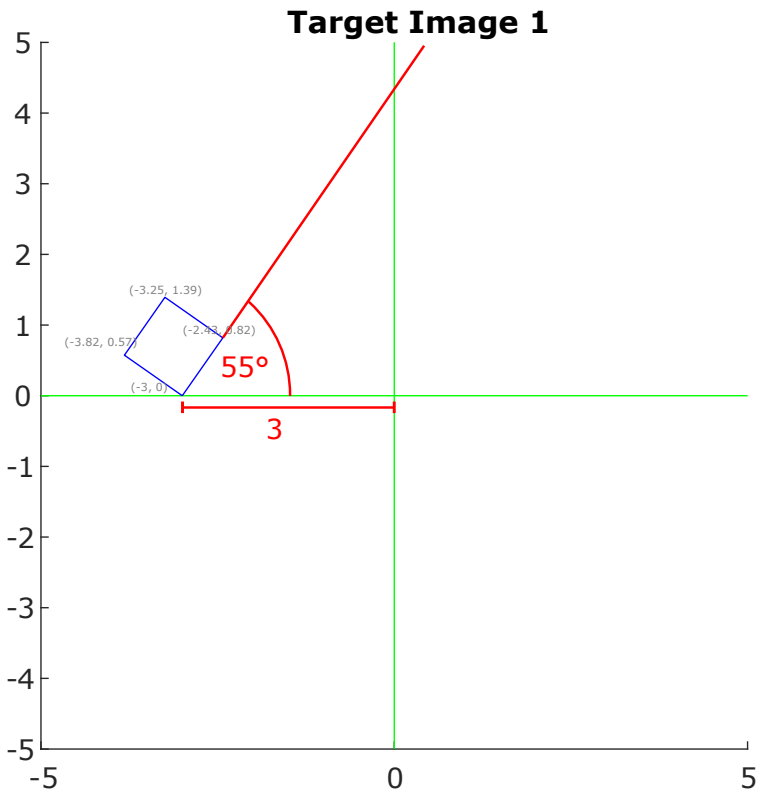


Abbildung 6: Target Image 1

Abbildung 7: Target Image 2

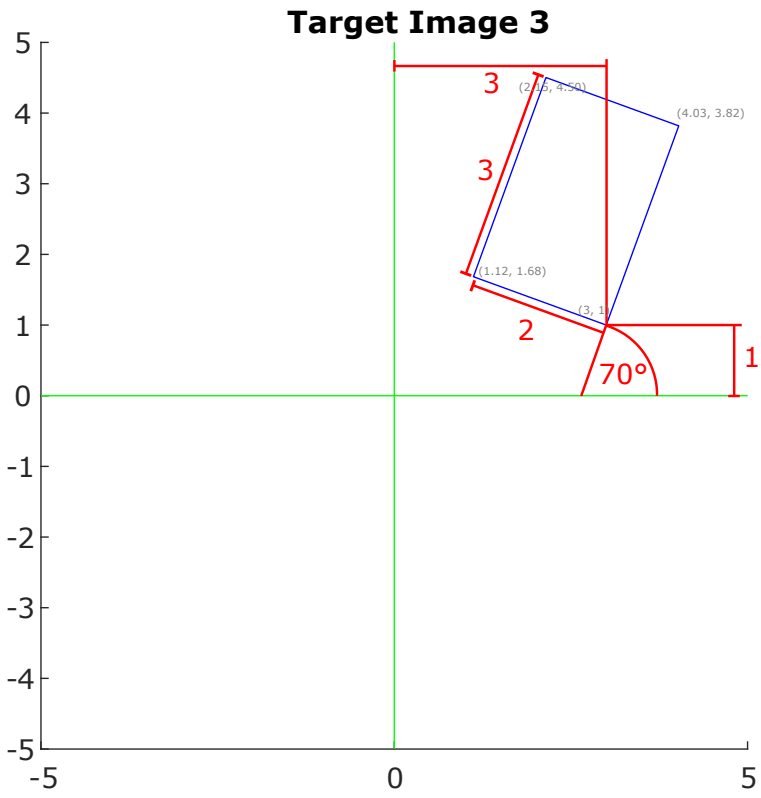


Abbildung 8: Target Image 3

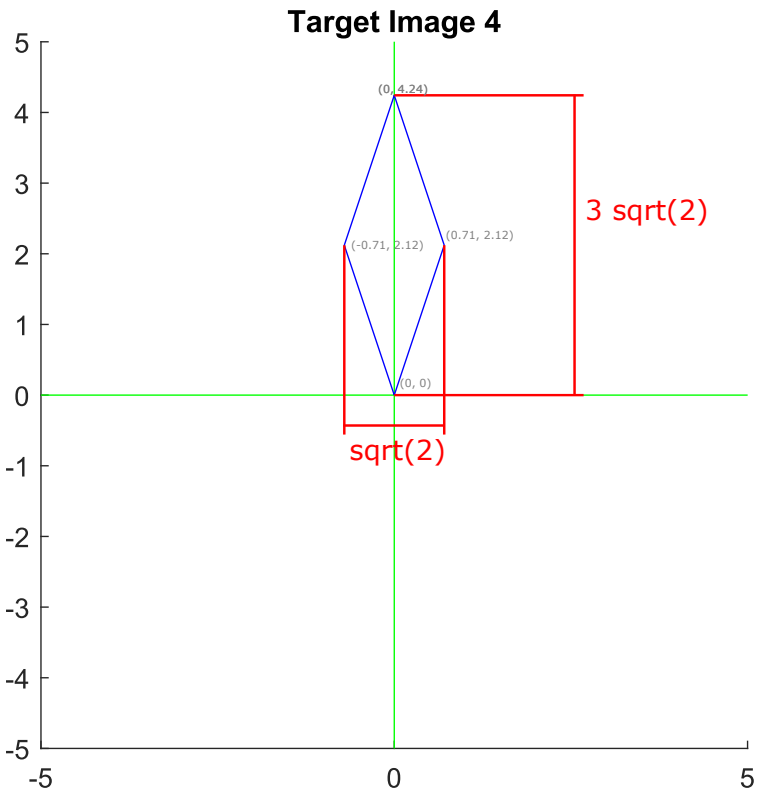


Abbildung 9: Target Image 4