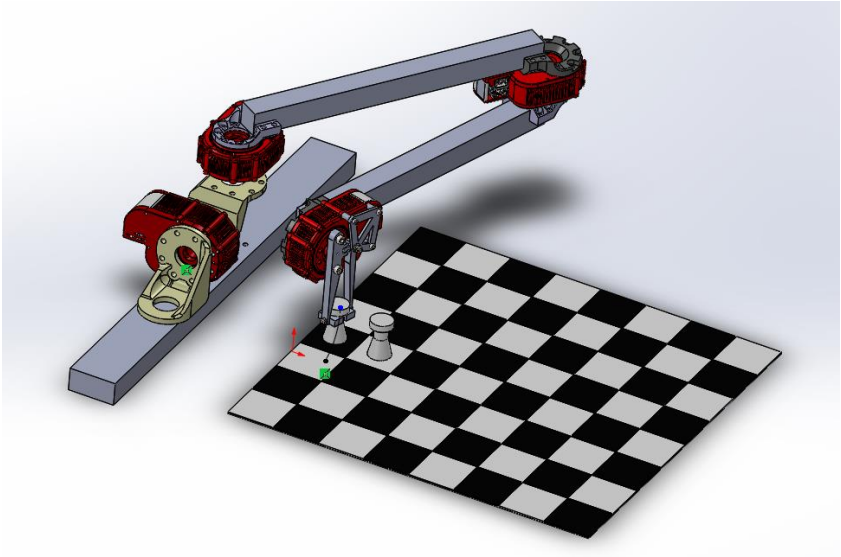
Schachroboter



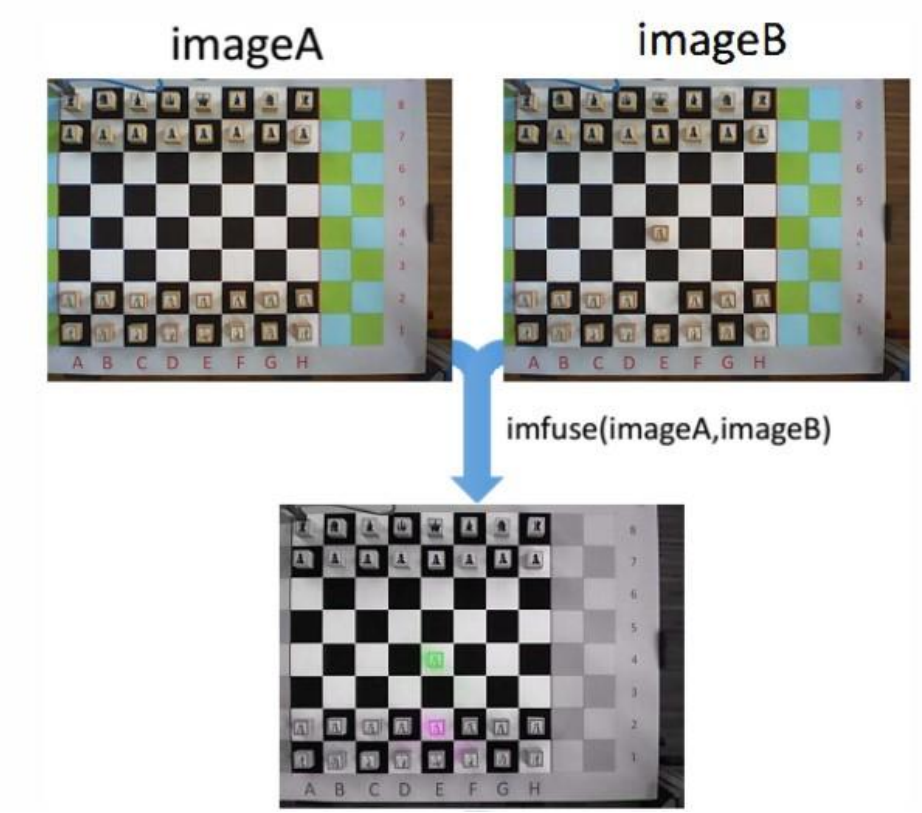
Marie Fieweger, Lennart Köhnke, Sebastian Kuhn, Kai Liu, Quan Pham

-

Seminar Entwicklung modularer robotischer Systeme

Stand der Technik

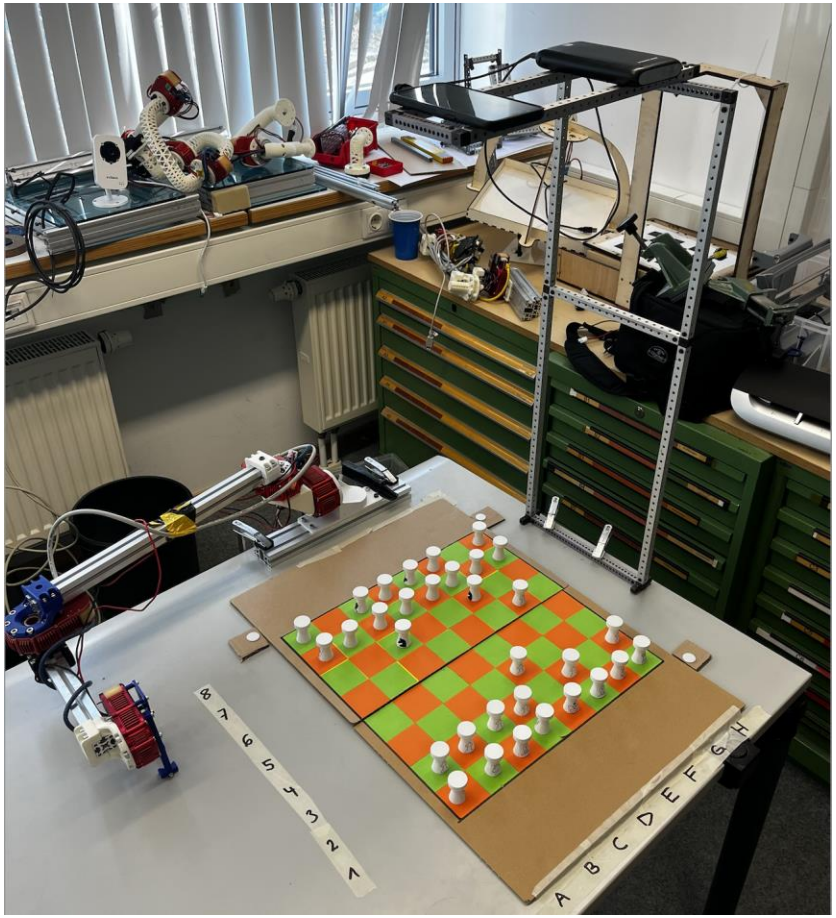
- Hauptkomponenten eines Schachroboters:
  - Roboterarm (mit Greifer)
  - Kamera
  - Software: Schachprogramm (Chessengine), MATLAB
  - Schachbrett, Figuren
- Roboterarm: Industrielle Roboterarm (z.B. KUKA)
- Computervision für Zugerkennung:
  - Methode 1: Differenz zwischen Bilder vor und nach dem Zug [1]
  - Methode 2: Erkennung einzelner Figuren



Beispiel Zugerkennung durch Verschmelzung der Bildern vor und nach dem Zug (Golz *et al.* 2015)

Versuchsaufbau

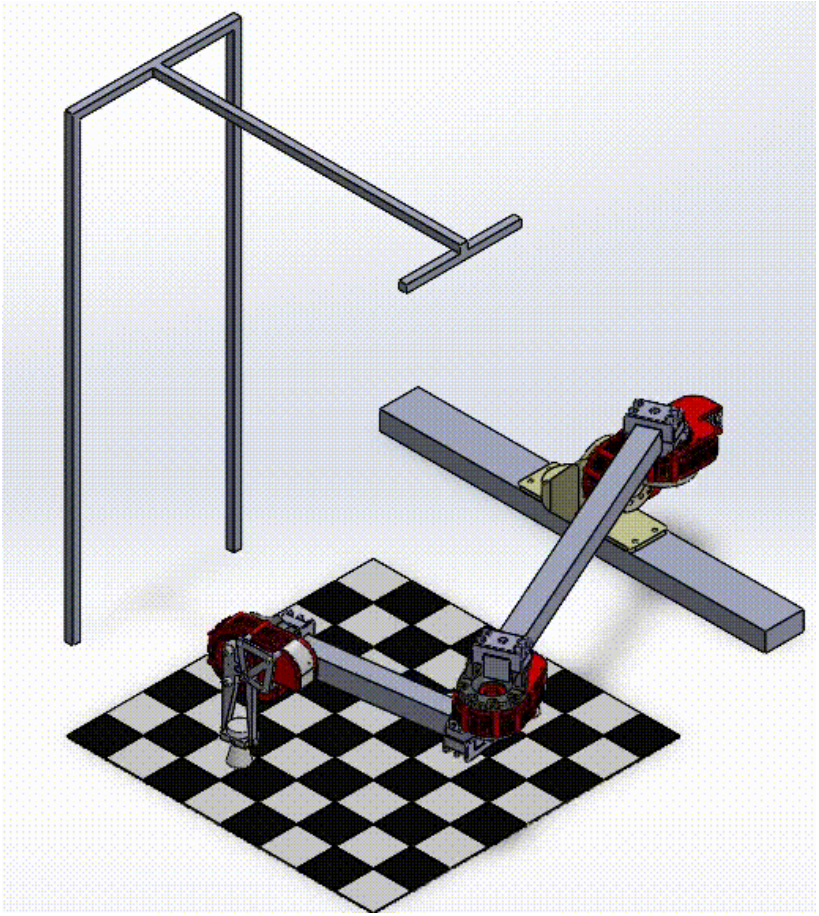
- Roboterarm:
  - HEBI-Module
  - Links
  - Brackets
- Kamera: Handy mit Verbindung zu MATLAB
- Schachbrett
  - Orange und grün: bessere Bilderkennung
- Figuren



Versuchsaufbau

Umsetzung Roboterarm

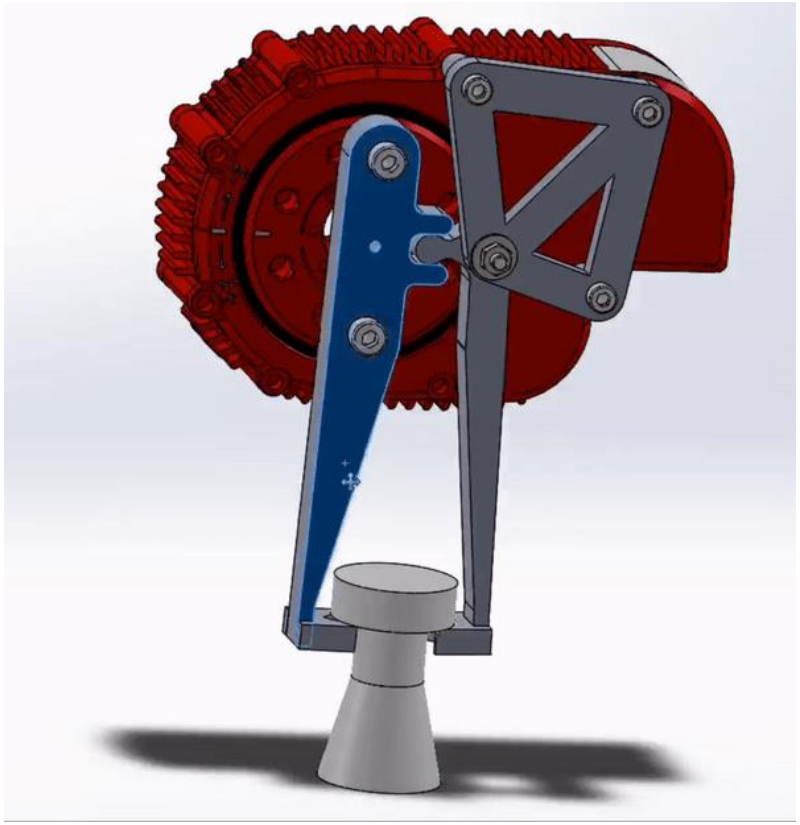
- Kippen 2R Arm mit 4 Gelenken
- Länge der Verbindungen mind. 40cm
- Höhe der Kamera ca. 75 cm
- Höhe der Basis für den Arm ca. 1cm



Animation des Kippenden 2R Arms mit 4 Gelenken

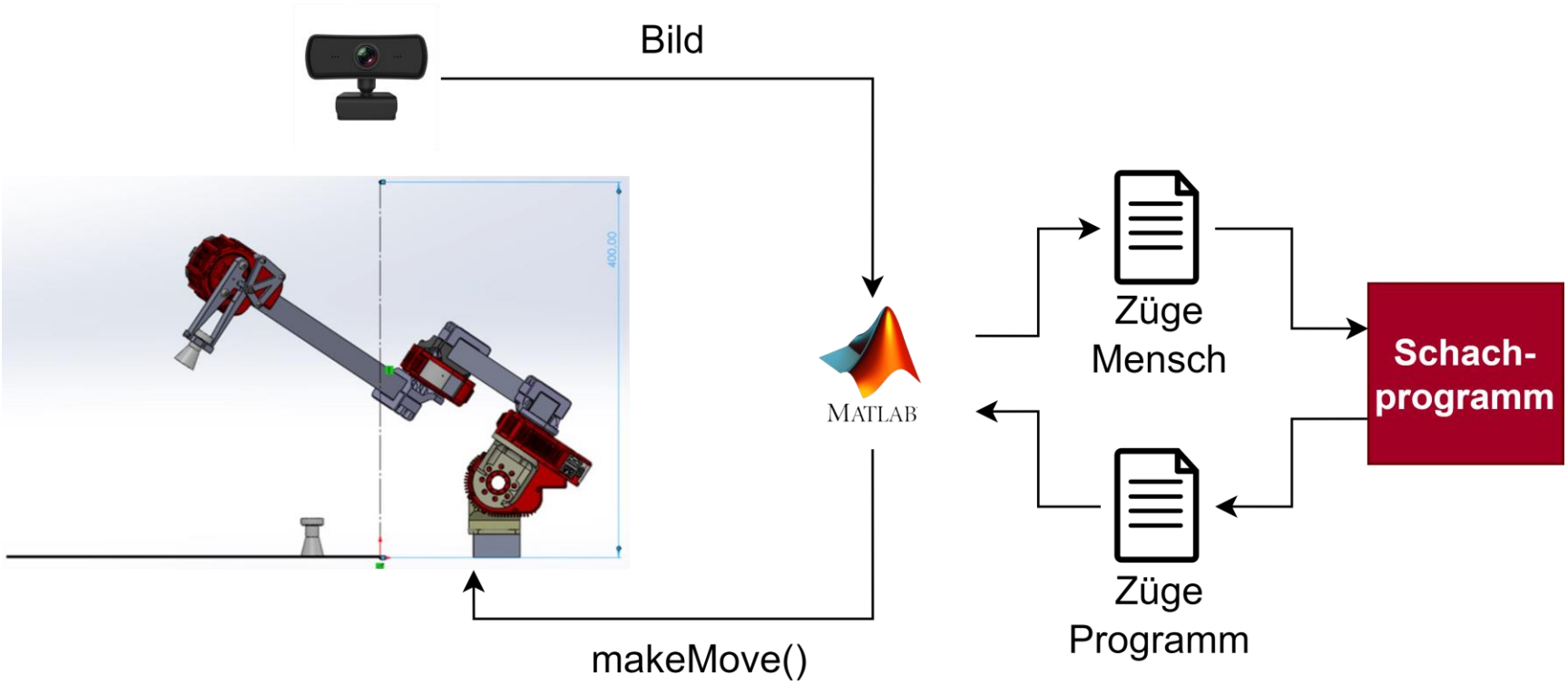
Umsetzung Greifer

- Greift Figur auch ohne exakte Positionierung
- Verzahnungsmechanismus --> simple Kopplung der zwei Seiten
- Leichtes Spiel, aber unproblematisch
- 3D-gedruckte und verschraubte Teile
- Erkennung ob Figur gegriffen wurde: über Drehmoment



Animation des Greifers

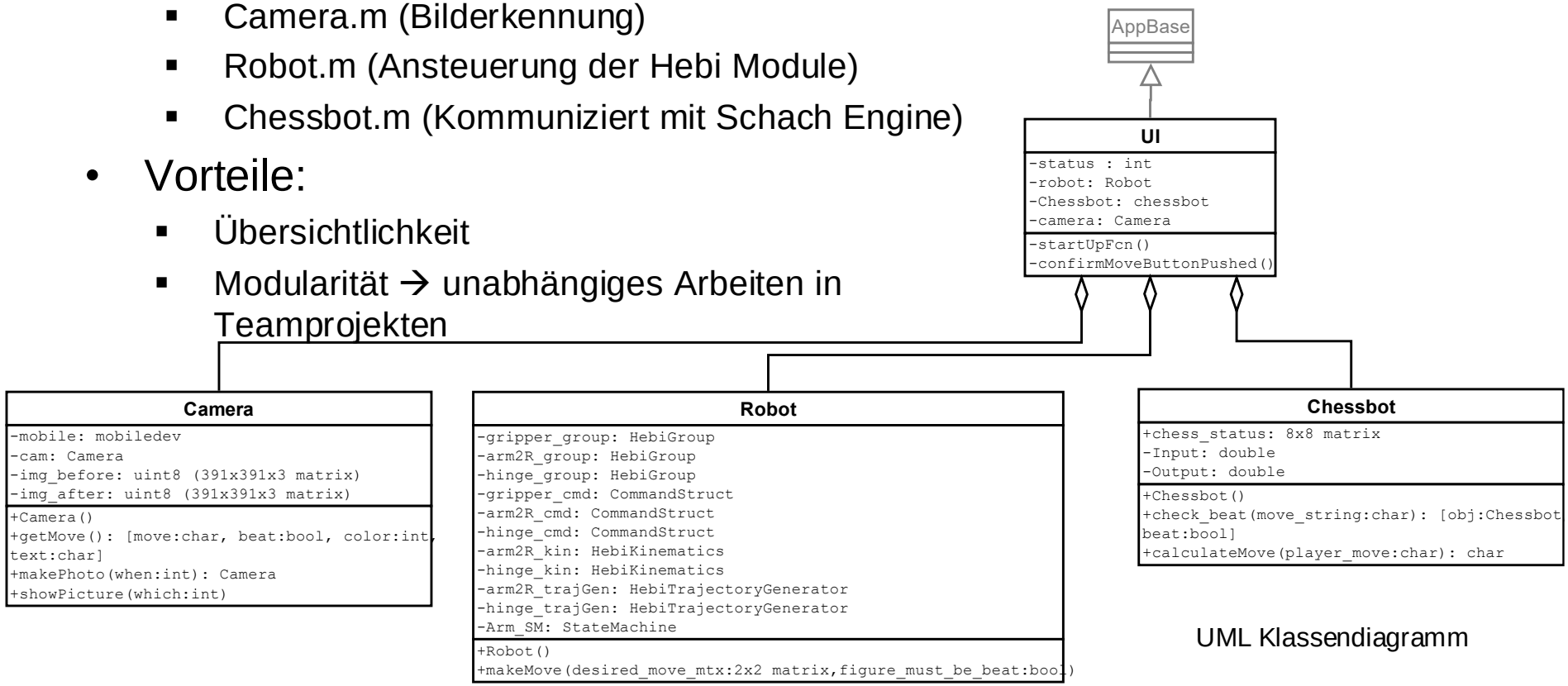
Schnittstelle Hardware - Software



Strukturdiagram des Spielablaufs

Umsetzung Software - Objektorientierung

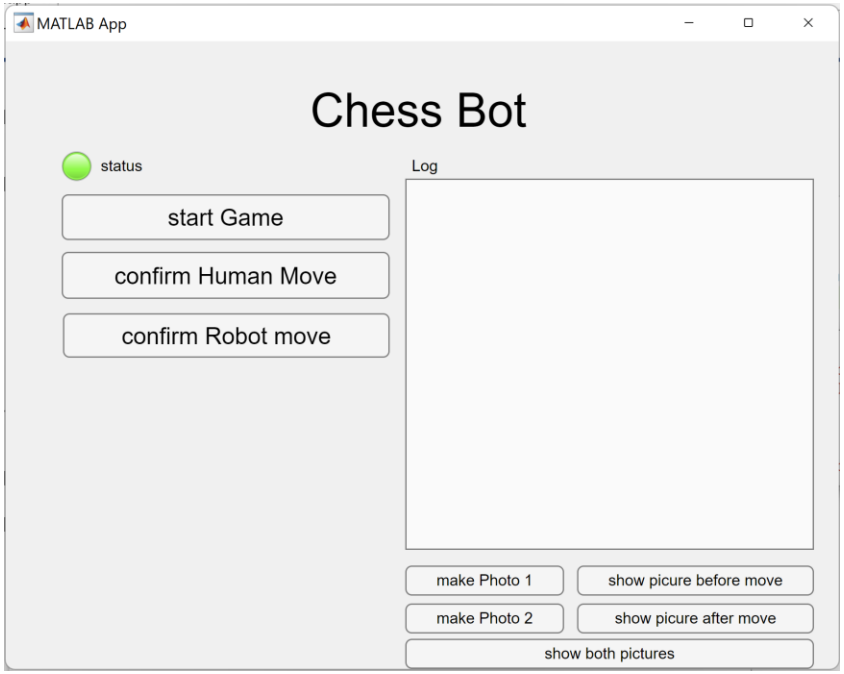
- Objekt Orientierter Ansatz, Unterteilung in folgende Klassen:
  - UI.mlapp (Benutzerschnittstelle, koordiniert Programmablauf)
  - Camera.m (Bilderkennung)
  - Robot.m (Ansteuerung der Hebi Module)
  - Chessbot.m (Kommuniziert mit Schach Engine)
- Vorteile:
  - Übersichtlichkeit
  - Modularität → unabhängiges Arbeiten in Teamprojekten



UML Klassendiagramm

Umsetzung Software – MATLAB App

- Benutzerschnittstelle ist als MATLAB App implementiert



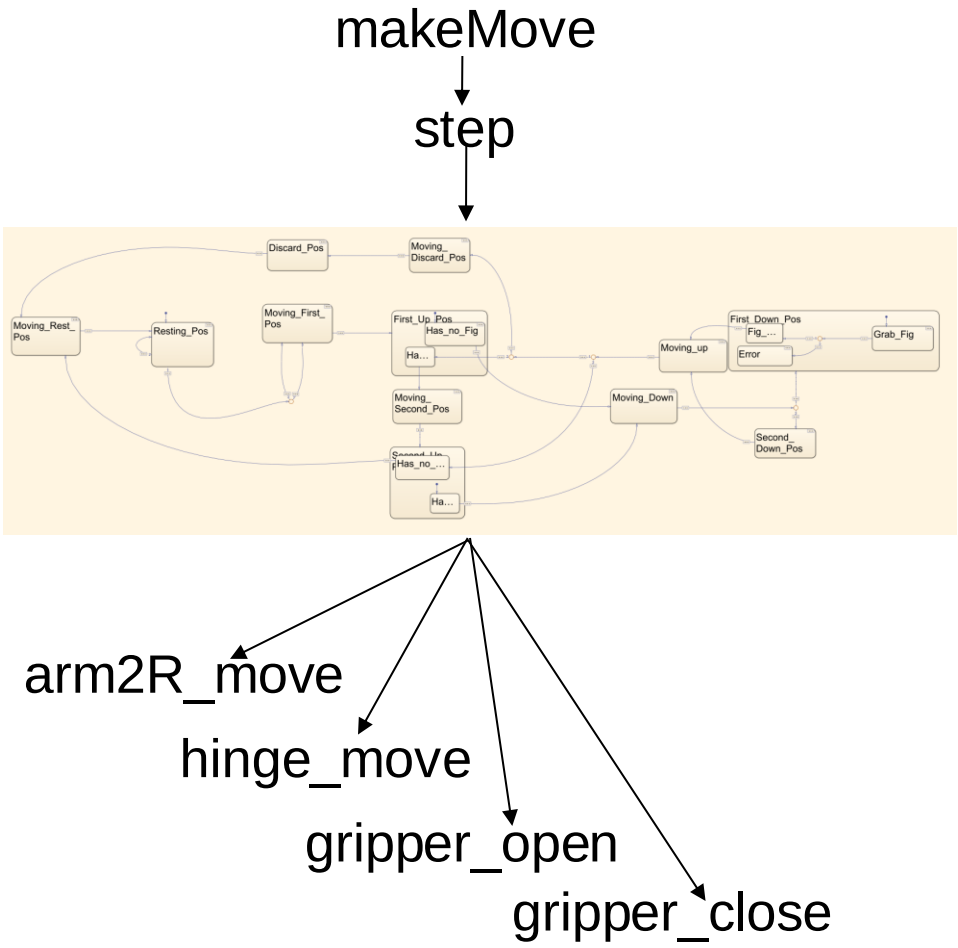
Chess Bot App

```
% Button pushed function: makePhoto2Button
function makePhoto2ButtonPushed(app, event)

    try
        app.camera = app.camera.makePhoto(1);

        [move, beat, ~, text] = app.camera.getMove();
        if(size(move) == [1 1])
            app.human_move = char(move);
            app.human_beat = beat;
            app.updateLog(text);
        else
            app.updateLog('Kein eindeutiger Zug erkannt');

        end

    catch
        app.updateLog('Fehler beim erkennen des Spielfeldes');
    end
end
```

Aufruf von Code durch Callback
Funktionen der Buttons

Umsetzung Software Roboterarm

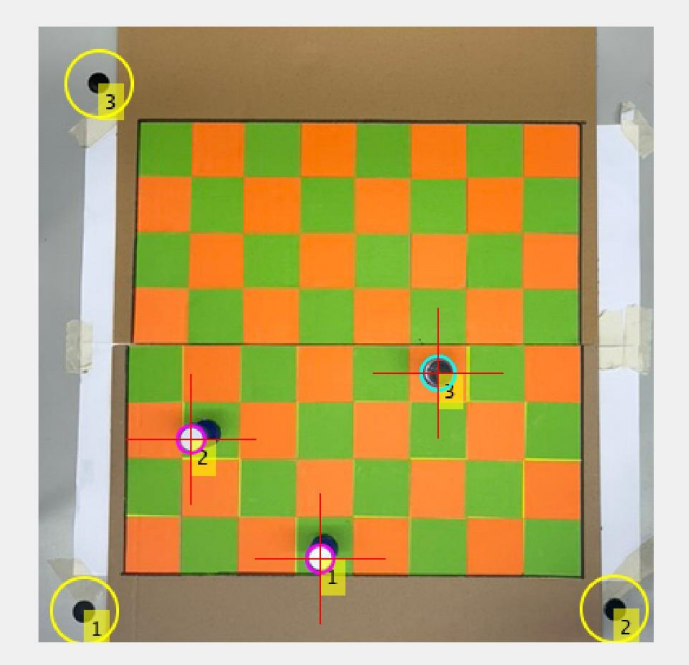
- Bewegungsablauf beim Ziehen muss gesteuert werden
- State Machine (MATLAB Stateflow)
- Wird von makeMove-Funktion aufgerufen und aktiviert Funktionen für Hinge, 2R-Arm und Greifer
- Eigenes Set von Variablen --> mit Robot-Klasse verknüpft
- Zyklische Abfrage über step()
- Ziehen mit und ohne Schlagen einer Figur



Umsetzung Computer Vision

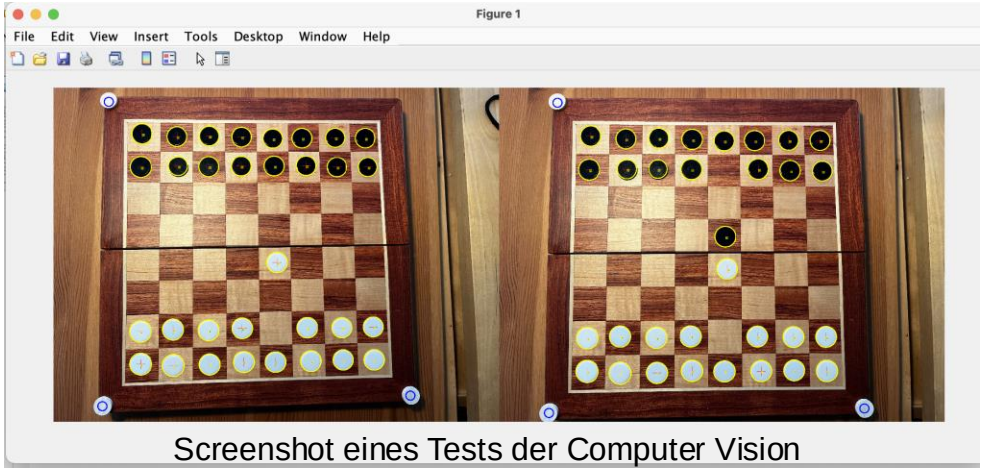
Ansatz: Erkennung der Figuren durch kreisförmige Marker

- Entfernen von Duplikaten mit Hilfe der Euklidischen Distanz
- Unterscheidung der Farbe durch Kontrastermittlung
- Erkennung des KooSys anhand der Extrempunkte
- Erstellen einer Status Matrix
- Erkennung des Zugs durch Differenz zweier Matrizen
- Identifizierung durch Switch Cases



Durch den Computer Vision Algorithmus markiertes Bild des Schachbretts

Umsetzung Computer Vision



Screenshot eines Tests der Computer Vision

```
board_state_prev =

     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     3     0     0     0
     0     0     0     0     0     0     0     0
     3     3     3     3     0     3     3     3
     3     3     3     3     3     3     3     3
```

```
board_state =

     1     1     1     1     1     1     1     1
     1     1     1     1     0     1     1     1
     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     3     0     0     0
     0     0     0     0     0     0     0     0
     3     3     3     3     0     3     3     3
     3     3     3     3     3     3     3     3
```

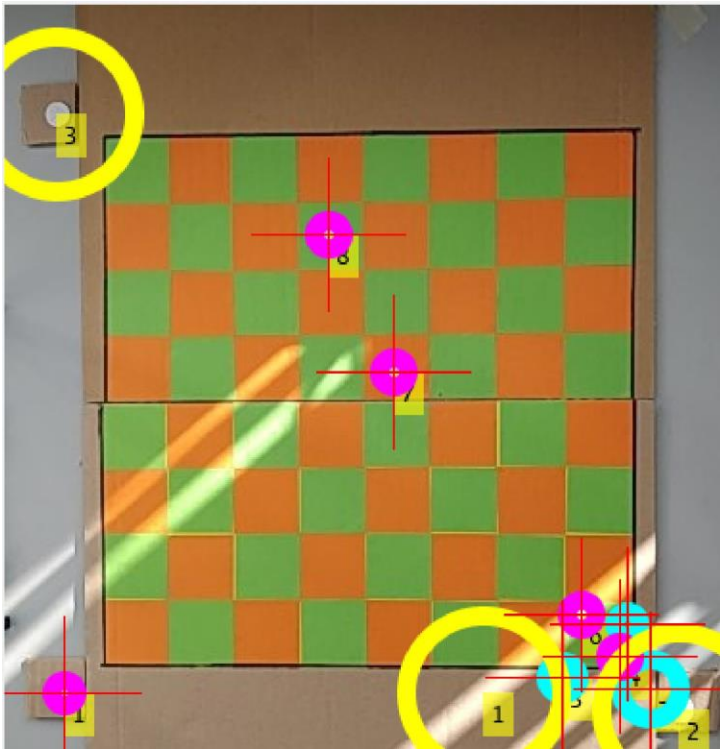
```
move =

    "e7e5"


'BLACK MOVES from e7 to e5'
```

```
board_move =

     0     0     0     0     0     0     0     0
     0     0     0     0    -1     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

```
switch max
case 3
% white moves to an empty field

case 2
% white beats a black figure

case 1
% black moves to an empty field

case 0

   switch min
   case 0
   % nothing has changed

   case -2
   % black beats a white figure
```

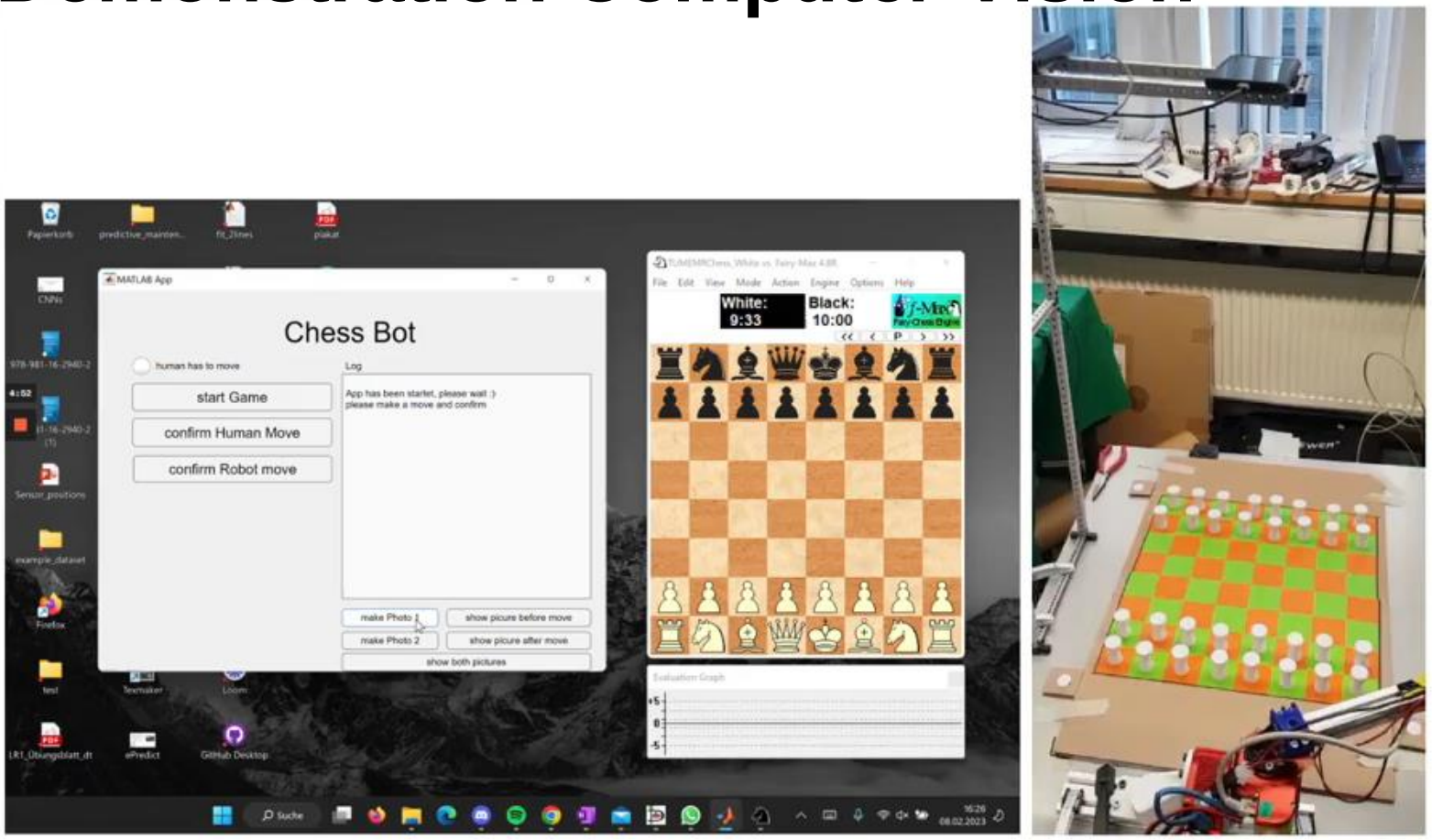
Diskussion und Aussicht

- Fehleranfälligkeit der Computer Vision
  →ArUco Marker
- Genaue Absetzen der Figuren
  → Magnete
- Stabilität der Figuren → Schrauben
- Verwendung einer anderen Kinematik
- Fehlerfreie Integration des Roboterarms



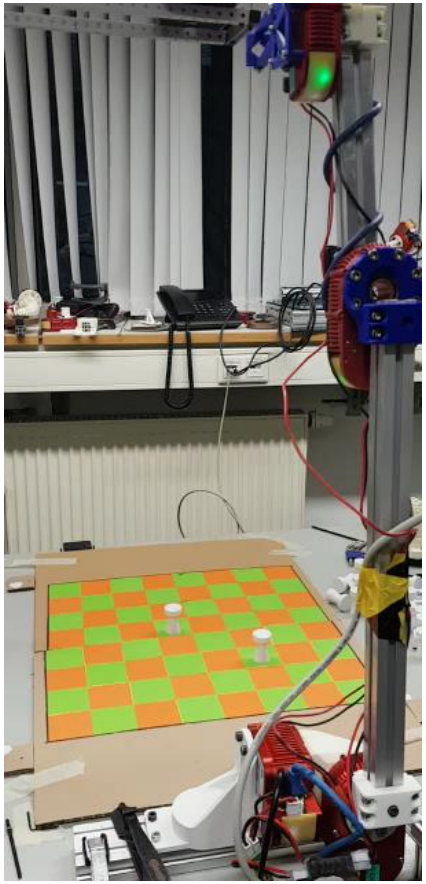
Fehleranfälligkeit bei schlechten Lichtverhältnissen

Demonstration Computer Vision



Erkennen des Schachspieles (Züge des Roboters werden manuell ausgeführt)

Demonstration Roboterarm



Roboterarm schlägt Schachfigur

Quellen

[1] J. Golz and R. Biesenbach, "Implementation of an autonomous chess playing industrial robot," *2015 16th International Conference on Research and Education in Mechatronics (REM)*, Bochum, Germany, 2015, pp. 53-56, doi: 10.1109/REM.2015.7380373