



**Robotika**  
**Manipulace pomocí robotické ruky**

31/10/2025

**Ondřej Vítá**

VITAONDR@FEL.CVUT.CZ

*Fakulta elektrotechnická*

*České vysoké učení technické v Praze  
Technická 2, 166 27 Praha*

**Josef Kahoun**

KAHOUJO1@FEL.CVUT.CZ

*Fakulta elektrotechnická*

*České vysoké učení technické v Praze  
Technická 2, 166 27 Praha*

**Šimon Pilc**

PILCSIMO@FEL.CVUT.CZ

*Fakulta elektrotechnická*

*České vysoké učení technické v Praze  
Technická 2, 166 27 Praha*

## Contents

<b>1</b>	<b>Zkratky a značení</b>	<b>3</b>
<b>2</b>	<b>Úvod</b>	<b>3</b>
2.1	Zadání úlohy . . . . .	3
2.2	Úprava zadání pro naši skupinu . . . . .	4
<b>3</b>	<b>Kalibrace kamery</b>	<b>4</b>
3.1	Kalibrační deska . . . . .	4
3.2	Kalibrace pomocí <i>cv2</i> . . . . .	4
<b>4</b>	<b>Hand-to-Eye kalibrace</b>	<b>5</b>
4.1	Získání transformace $T_{RC}$ a $T_{TG}$ . . . . .	5
4.2	Reprojekční chyba . . . . .	7
<b>5</b>	<b>Robot</b>	<b>9</b>
<b>6</b>	<b>Hardware</b>	<b>10</b>
6.1	Úchop robotického manipulátoru . . . . .	10
6.2	Desky . . . . .	11
<b>7</b>	<b>Detekce desek</b>	<b>12</b>
7.1	Načtení desky z csv souboru . . . . .	12
7.2	Detekce ve scéně . . . . .	12
7.3	Získání souřadnic pozic dílků . . . . .	13
<b>8</b>	<b>Kód</b>	<b>13</b>
8.1	camera_calibration . . . . .	14
8.2	robot_wrapper . . . . .	14
8.3	hand_to_eye_calib . . . . .	14
8.4	display . . . . .	14
8.5	Pomocné kódy . . . . .	15
<b>9</b>	<b>Testy</b>	<b>15</b>
<b>10</b>	<b>Závěr</b>	<b>16</b>

## 1 Zkratky a značení

V následujícím textu je užito následujících zkratek a značení:

- s.s. - soustava souřadnic
- $T_{AB}$  - transformace ze soustavy souřadnic  $B$  do soustavy souřadnic  $A$

## 2 Úvod

Tato technická zpráva popisuje řešení semestrální úlohy z předmětu Robotika (B3B33ROB1). Cílem námi vybrané úlohy je reorganizace kostek, která zahrnuje práci s kamerou a viděním, pomocí něhož se provede sběr a umístování kostek. Tedy 1. možnost ze zadání. Řešení zahrnovalo kalibraci kamery a kalibraci hand-to-eye, detekci desek a jejich pozic, návrh uchopovacího mechanismu a programování robota. Zpráva se zaměřuje na popis použitých metod a shrnutí dosažených výsledků.

### 2.1 Zadání úlohy

Pomocí průmyslového robotického manipulátoru přeneste všechny kostky z jedné desky do druhé. Vzorová deska je zobrazena na Obr. 1. Parametry desky pro umístění kostek jsou:

- Deska s otvory pro umístění kostek má rozměr  $240 \times 200$  mm.
- Deska je v protilehlých rozích označena ArUco značkami (DICT\_4X4\_50).
- Značky mají po sobě jdoucí ID.
- Střed ArUco značky s nižším ID je počátkem soustavy souřadnic na desce. Osa X je rovnoběžná s delší hranou desky, osa Y je rovnoběžná s kratší stranou desky.
- Osy směřují od ArUco značky s nižším ID ke značce s vyšším ID.
- Stěny kostek (otvorů) jsou rovnoběžné s osami soustavy souřadnic.
- Stěny sousedících kostek umístěných v otvorech jsou vzdáleny min. 40 mm.
- Pro každou desku je k dispozici CSV soubor, který definuje polohu otvorů v desce:
  - Sloupce jsou na řádku oddělené čárkou.
  - Oddělovač desetinných čísel je tečka.
  - Na každém řádku souboru jsou vždy dvě čísla.
  - Na prvním řádku jsou uvedeny ID ArUco značek na desce.
  - Na druhém a dalších řádcích jsou uvedeny souřadnice středu jednotlivých otvorů pro umístění kostek.
  - Souřadnice jsou uvedeny vždy v pořadí x, y.
  - Souřadnice jsou uvedeny v milimetrech.

Zadání mělo varianty A až D, naše skupina řešila variantu D:

V pracovním prostoru robota budou umístěny dvě desky, které mohou být umístěny v různých výškách a také mohou být natočeny o 7 nebo 15 stupňů okolo horizontálních os. Cílem je přenést všechny kostky z jedné desky na druhou. Kostky nejsou označeny a je jedno, která kostka bude vložena do kterého otvoru. Cvičící před spuštěním programu určí, na které desce se kostky nacházejí. Úkolem robota je poté bezpečně sebrat kostky z této desky a přesunout je na druhou desku. Deska s pružným podstavcem bude položena na rovině pracovní desky.

## 2.2 Úprava zadání pro naši skupinu

Původní zadání však v našem případě bylo upraveno a rozšířeno aby na projektu mohl pracovat tříčlenný tým. Změnou je, že místo dřevěných kostek bude manipulátor přemisťovat v CAD souborech specifikovaný plastový díl. S tím souvisí následující přidané úkoly:

- Návrh a výroba uchopovacího mechanismu manipulátoru uzpůsobeného pro plastový díl.
- Uzpůsobení desek ze kterých budou díly manipulátorem uchopovány.

## 3 Kalibrace kamery

### 3.1 Kalibrační deska

Pro přesný odhad polohy objektů ze scény je nutné kvalitně zkalibrovat kameru a získat matici kamery  $K$  a distorzní koeficienty  $dist\_coef$ , což jsou parametry kamery, které popisují, jak moc se zkresluje obraz. Pro kalibraci jsme využili desku ChArUco. Ta oproti desce se vzorem šachovnice má několik výhod, ale hlavně nevadí částečné zakrytí desky. Přesně jsme využili ChArUco desku s parametry:

- rozměr - 7x9
- velikost ArUco značky - 22 mm
- velikost čtverce - 30 mm

Použitá deska je zobrazena na obr. 1.

### 3.2 Kalibrace pomocí cv2

Pro kalibraci jsme nafotili fotky s různě natočenou a položenou kalibrační deskou, přičemž jsme se snažili, aby distribuce desky na fotkách byla co nejvíce rovnoměrná. Pro kalibraci jsme využili funkce z knihovny *cv2*. Prvně jsme ze všech fotek získali indexy a pixely rohů ArUco značek pomocí funkce *aruco.detectMarkers*. Následně jsme pro každou fotku využili funkci *aruco.interpolateCornersCharuco*, která vezme nalezené ArUco značky a vrátí jejich pozici se subpixelovou přesností. Po zpracování všech fotek se zavolá funkce *aruco.calibrateCameraCharuco*, která nám vrátí matici kamery a distorční koeficienty.

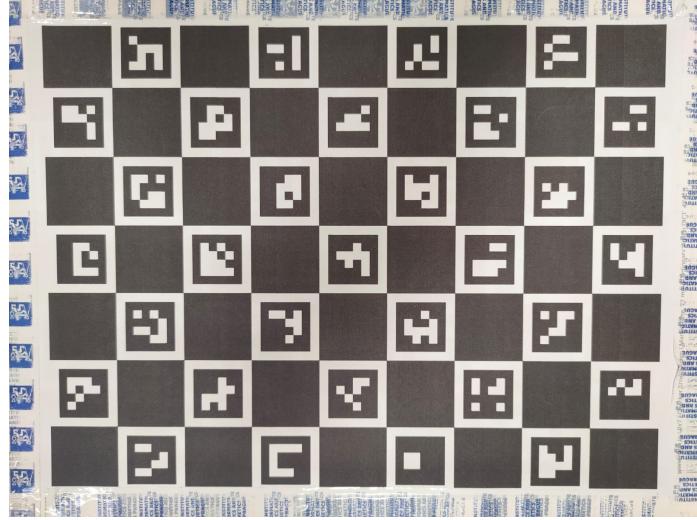


Figure 1: Použitá ChArUco deska

Po kalibraci jsme získali následující výsledky:

$$K = \begin{bmatrix} 4720.25 & 0 & 926.91 \\ 0 & 4710.67 & 646.41 \\ 0 & 0 & 1 \end{bmatrix}, \quad (1)$$

$$dist\_coef = [-0.22 \ -0.85 \ 0.0006 \ 0.004 \ 6.83]. \quad (2)$$

#### 4 Hand-to-Eye kalibrace

Abychom mohli ovládat robota pomocí kamery, potřebujeme získat transformaci ze souřadnicového systému kamery do s. s. robota. To řešíme pomocí rovnice

$$A^i X = Y B^i, \quad (3)$$

kde v případě hand-to-eye  $A^i = T_{CT}^i = FK^i$  je transformace ze souřadnic cíle (target) do souřadnic kamery<sup>1</sup>,  $X = T_{TG}$  je transformace ze souřadnic cíle do souřadnic gripperu,  $Y = T_{CR}$  je transformace ze souřadnic robota do souřadnic kamery a  $B^i = T_{RG}^i$  je transformace z gripperu do robota získaná z dopředné kinematiky. Potřebné transformace jsou znázorněny na obr. 2.

##### 4.1 Získání transformace $T_{RC}$ a $T_{TG}$

Pro vyřešení rovnice musíme získat co nejvíce měření transformací  $T_{CT}^i$  a  $T_{RG}^i$ . Jako target jsme použili vlastní ArUco značku přidělanou na gripper robota, viz obr. 3. Nyní můžeme změřit obě transformace,  $T_{RG}^i$  z dopředné kinematiky, a  $T_{CT}^i$  pomocí funkcí knihovny *cv2 aruco.detectMarkers* a *solvePnP*, které ve scéně najdou ArUco značku a vrátí její souřadnice

---

1. Horní index  $i$  značí, že tyto transformace jsou proměnné v čase.

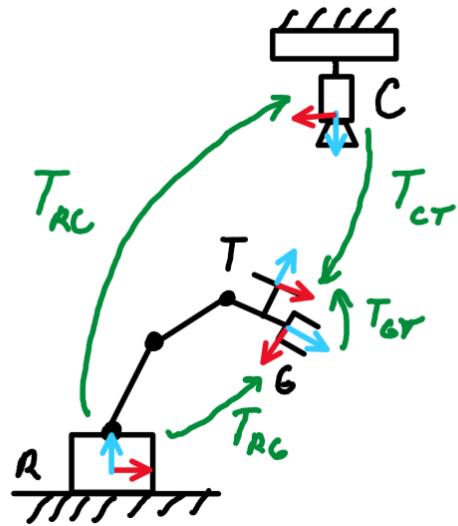


Figure 2: Znázornění použitých transformací při hand-to-eye kalibraci



Figure 3: Target (ArUco marker) přidělaný na gripper robota

v kamerovém s. s. Těchto dvojic měření je potřeba co nejvíce, proto jsme vytvořili program, který iterativně prochází předdefinovanou část prostoru, a pokud kamera vidí ArUco značku, uloží si dané transformace. Po získání těchto korespondujících transformací využijeme funkce `calibrateRobotWorldHandEye` pro vypočítání výsledných transformací. Tímto

jsme získali následující transformace ( $t$  (m) značí translační vektor,  $r$  rotační):

$$t_{RC} = \begin{bmatrix} 0.471 \\ -0.0001 \\ 1.178 \end{bmatrix}, \quad (4)$$

$$r_{RC} = \begin{bmatrix} 2.24 \\ 2.18 \\ -0.04 \end{bmatrix}, \quad (5)$$

$$t_{GT} = \begin{bmatrix} -0.064 \\ 0 \\ 0.0085 \end{bmatrix}, \quad (6)$$

$$r_{GT} = \begin{bmatrix} 2.09 \\ 2.21 \\ 0.15 \end{bmatrix}, \quad (7)$$

## 4.2 Reprojekční chyba

Jelikož naše zadání vyžaduje větší přesnost, musíme dále zpřesnit  $T_{RC}$ . Toho dosáhneme pomocí minimalizace reprojekční chyby.

Pokud vidíme ve scéně naší ArUco značku, můžeme zjistit, na jaké pixely se promítou rohy této značky tak, že je nalezneme na obrázku. Také to ale můžeme vypočítat z transformací a z projekce kamery. Jelikož známe velikost použité ArUco značky, můžeme napsat souřadnice všech rohů v s.s. targetu (střed s.s. je uprostřed značky). Pokud  $s$  je rozměr ArUco značky, pak levý horní roh má souřadnice  $[-s/2, s/2, 0]$ , pravý horní roh souřadnice  $[s/2, s/2, 0]$ , apod. Na obr. 2 jsou znázorněny známé transformace souřadnic, z hand-to-eye kalibrace máme transformace z robota do kamery a z gripperu do targetu (ArUco značky), z přímé kinematiky pak transformaci z robota do gripperu. Každý roh nyní můžeme transformovat do s.s. kamery podle

$$t_C = T_{CR} \cdot T_{RG} \cdot T_{GT} = T_{RC}^{-1} \cdot T_{RG} \cdot T_{GT}. \quad (8)$$

Pro objekt v kamerových souřadnicích platí:

$$\begin{bmatrix} u_h \\ v_h \\ w_h \end{bmatrix} = K \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = K \cdot t_C, \quad (9)$$

kde  $u_h, v_h, w_h$  jsou homogenní souřadnice příslušného pixelu a  $x_c, y_c, z_c$  jsou souřadnice bodu v kamerovém s. s. Hodnotu pixelů poté můžeme získat jako  $u = \frac{u_h}{w_h}$  a  $v = \frac{v_h}{w_h}$ . Tuto transformaci do pixelů dělá funkce `cv2.projectPoints`. Souřadnice pixelu můžeme tedy získat z funkce nalezení ArUco značky v obrazu, nebo přes transformace. Je jasné, že ideálně by hodnota pixelů měla být stejná, ovšem jelikož kalibrace nikdy není perfektně přesná, můžeme pro každý roh každého měření provedeného v 4.1 zavést reprojekční chybu jako

$$e_{ijx} = (u_{ij} - u_{ij}^T)^2, \quad (10)$$

$$e_{ijy} = (v_{ij} - v_{ij}^T)^2, \quad (11)$$

kde  $i$  je index měření,  $j$  je index rohu, tedy 0,1,2 nebo 3,  $x/y$  značí osy a horní index  $T$  značí, že se jedná o pixely získané z transformací.

Nyní můžeme zavést optimalizační úlohu, která bude minimalizovat tuto chybu přes všechna měření, neboli:

$$\min_{t_{RC}, r_{RC}, t_{GT}, r_{GT}} \sum_{i=0}^N \sum_{j=0}^3 e_{ijx} + e_{ijy}, \quad (12)$$

kde  $t$  jsou translační vektory a  $r$  jsou rotační vektory. Celkově tedy minimalizuje přes 12 proměnných, ale jelikož rotační vektory se moc nemění a chyba je převážně způsobená translačními, minimalizujeme jenom přes ně (tedy 6 proměnných). Takto sepsanou úlohu minimalizujeme pomocí knihovny *scipy* a funkce *least\_squares*, která minimalizuje nelineární úlohu nejmenších čtverců, což je přesně náš případ.

Tímto jsme získali nové transformace, které zmenšily reprojekční chybu na 63 % původní a získali jsme následující translace (v metrech):

$$t_{RC} = \begin{bmatrix} 0.470 \\ -0.001 \\ 1.158 \end{bmatrix}, \quad (13)$$

$$t_{GT} = \begin{bmatrix} -0.0643 \\ -0.003 \\ 0.0103 \end{bmatrix}, \quad (14)$$

## 5 Robot

Pro splnění zadání jsme si vybrali jednoho ze tří nabízených robotů, jedná se o CRS 97, kterého můžete vidět na obrázku 4. Na tomto robotovi probíhala většina našich testů a vývoj programu.

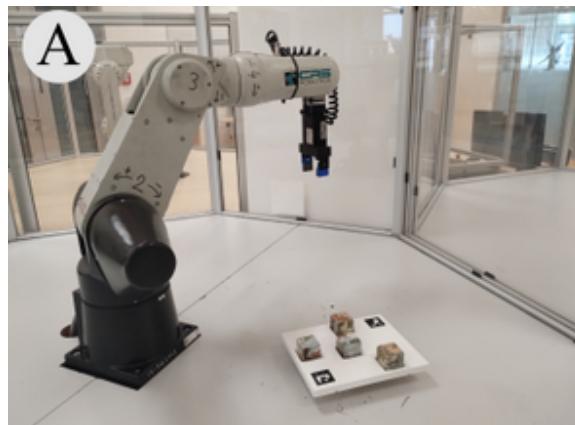


Figure 4: Robotický manipulátor CRS 97

Z detailnějšího hlediska se jedná o šestiosého robota s gripperem, na který lze namontovat libovolné packy pro uchopování. Této vlastnosti jsme využily při návrhu mechanismu pro uchopení plastového dílku 6.1.

Pro manipulaci s robotem nám byla poskytnuta Python knihovna *ctu\_crs*. Na tuto knihovnu jsem navázali Python Class Robot 8.2.

## 6 Hardware

### 6.1 Úchop robotického manipulátoru

Pro úchop plastového dílu bylo třeba modifikovat prsty manipulátoru, tedy pouze poslední článek uchopovacího mechanismu. Jelikož díl ze zadání musí být zvedán v orientaci jako je na obr. 5. Prsty byly navrženy aby využívaly největší možné styčné plochy vnitřku modelu.

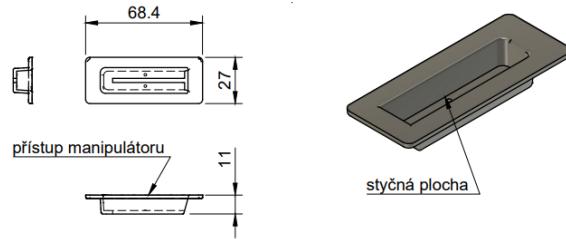


Figure 5: Plastový díl, u bokorysu vyznačena orientace.

Princip uchopení spočívá ve vložení k sobě plně sevřených prstů do vnitřku dílu, jeho prohloubené části. Manipulátor prsty následně roztáhne, čímž uchopí a bude držet díl. Díl je puštěn opět sevřením prstů k sobě.

Po první iteraci se design ustálil na ten, který je na obr. 6. Místo styčných ploch bylo ve 3D modelu připraveno pro nalepení přibližně 1 mm tlusté gumové vrstvy. Před styčnými plochami byla přidána plocha pod úhlem s horizontální rovinou, aby při nepřesném zarovnání středu dílu se středem uchopovacího mechanismu došlo k zarovnání během vkládání prstů do dílu. U předešlé iterace, která měla prsty konci zarovnanými s horizontální rovinou, došlo k zastavení a zaseknutí o okraj dílu.

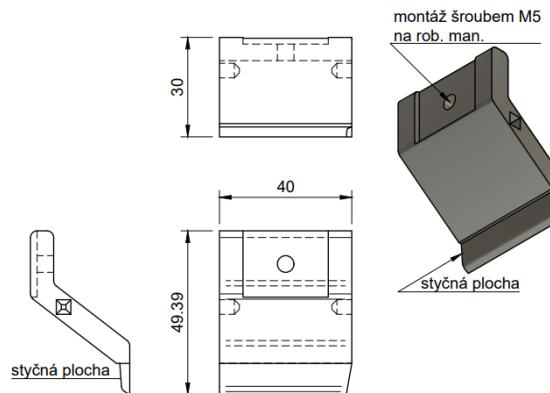


Figure 6: Z čelního pohledu na robota levý prst uchopovacího mechanismu manipulátoru

Rozměry, hlavně délka (na obr. 6 vertikální rozměr u čelního pohledu) nových prstů se též liší od původních, což bylo kompenzováno v softwaru.

Po vytisknutí na 3D tiskárně z PETG byla gumová vrstva ve formě kusu duše z pneumatiky jízdního kola přilepena na styčné plochy. To propůjčuje jistou adaptabilitu celému úchopu.



Figure 7: Fotka vyrobených prstů

## 6.2 Desky

Jelikož plastový díl se nevejde do děr pro kostky na deskách, nejjednodušší uskutečnitelným řešením bylo vytvořit pro plastový díl vložku na obr. 8, která zapadne na místo kostky (spodní část modelu, s kruhovým výstupkem, který slouží k zarovnání středu díry na střed vložky) a udržela střed plastového dílu na stejném místě jako je střed díry pro kostku v souřadnicích X a Y. Z výškového hlediska vložky vystupují nad rovinu desky, což bylo kompenzováno v softwaru **společně** s délkou prstu.

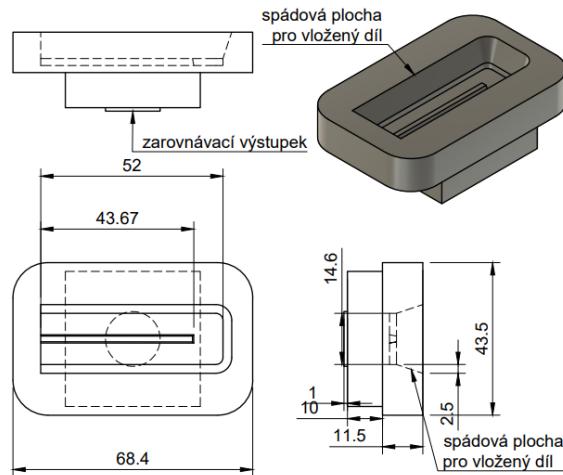


Figure 8: Vložka do desky uzpůsobená pro plastový díl

Dle původního designu podložky byly také upraveny okraje otvoru vložky tak, aby měly spádový úhel. Tato úprava pomůže dílu zapadnout do správné a vždy stejné pozice i když je manipulátorem vložen nepřesně.

## 7 Detekce desek

Každá deska, na kterou se budou dílky pokládat (nebo se brát) má pevně dané rozměry, a to 240x200 mm. V levém spodním rohu a v pravém vrchním jsou umístěny 2 ArUco značky o velikosti 36 mm sloužící pro detekci. V každé desce jsou otvory na dílek. Nákres typické desky je na obr. 9. Informace o indexech ArUco značek a rozmištění otvorů jsou připojeny v csv souboru.

Práce se samotnými deskami se dělí na 3 části, nejprve je nutné načít informace o desce

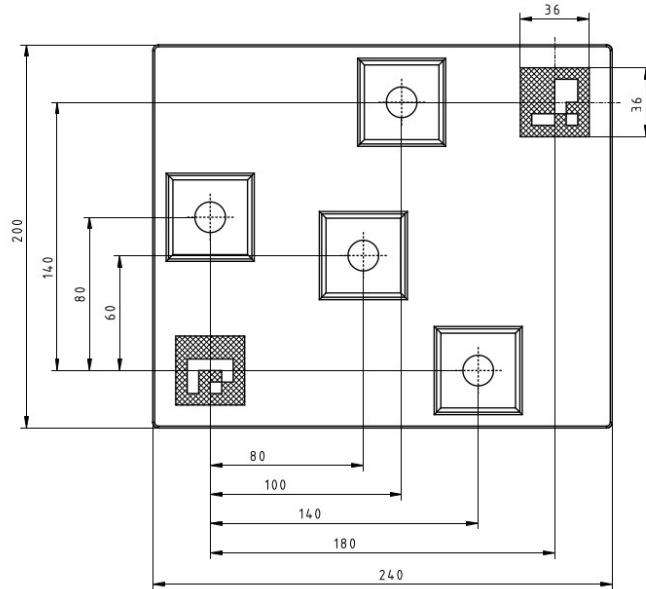


Figure 9: Příklad desky

z přiloženého csv souboru, poté desku nalézt ve scéně, a nakonec získat souřadnice pozic, kam se má robot pohnout, aby vzal/pustil dílek.

### 7.1 Načtení desky z csv souboru

Načtení informací je přímočaré, použili jsme knihovnu *csv*, která má naimplementovaný reader csv souborů. První řádek souboru obsahuje vždy indexy ArUco značek, další řádky poté souřadnice středu otvorů od středu ArUco značky umístěné v levém dolním rohu, tedy první číslo je vzdálenost podél delší hrany, druhé podél kratší. Vzdálenosti jsou také vyobrazeny na obr. 9.

### 7.2 Detekce ve scéně

Pro detekci desky ve scéně jsme využili funkci z knihovny *cv2 solvePnP*, která slouží k nalezení transformace z s.s. objektu do s.s. kamery (Perspective-n-Point problem). Tato funkce pomocí korespondencí z s.s. objektu a 2D scény získá danou transformaci. Jako střed s.s. desky jsme zvolili střed levé spodní ArUco značky, s osou x směřující podél delší hrany dovnitř desky, osa y směřuje podél kratší hrany dovnitř desky a osa z směřuje ven z desky tak

aby dokončila pravotočivou soustavu. Jako korespondenční body pro PnP vybereme rohy ArUco značek. Souřadnice rohů v 2D scéně získáme pomocí funkce *aruco.detectMarkers* z knihovny *cv2*. Souřadnice rohů v s.s. desky získáme jednoduše, jelikož značky mají definovanou velikost a vzdálenost od sebe a leží v jedné rovině, budou souřadnice rohů levé spodní značky (od levého horního rohu po směru hodinových ručiček)  $[-36/2, 36/2, 0]$ ,  $[36/2, 36/2, 0]$ ,  $[36/2, -36/2, 0]$  a  $[-36/2, -36/2, 0]$ . Souřadnice rohů druhé ArUco značky budou stejné, jen se k souřadnici x přiřete 180 mm (vzdálenost středu ArUco značek v x-ové ose) a k souřadnici y 140 mm. Tímto máme všechny korespondence, a jelikož z 3.2 známe matici kamery a distorzní koeficienty, můžeme vyřešit PnP problém. Ten nám vrátí translační a rotační vektor, ze kterého můžeme vytvořit SE(3) transformaci z desky do kamery  $T_{CB}$ .

### 7.3 Získání souřadnic pozic dílků

Aby mohl robot vzít/položit délku, potřebujeme pozici daného bodu v s.s. robota a danou rotaci gripperu, tedy transformaci. Z těchto informací poté můžeme pomocí inverzní kinematiky získat souřadnice kloubů nutné k vykonání pohybu. Ze sekce 4 známe transformaci z kamery do robota  $T_{RC}$  a z předchozího kroku transformaci z desky do kamery  $T_{CB}$ . Transformaci z každého držáku na délku do s.s. desky vytvoříme jednoduše, jelikož jsme osy zavedli podél hran, můžeme rovnou využít posun v osách x a y tak, jak jsme je získali z csv souboru. Jelikož držák na délku má střed výše než je rovina desky, je osa z posunutá o konstantní ofset 20 mm. Jelikož ale osa z s.s. gripperu směřuje ven, musí osa z s.s. držáku směřovat dolů do desky, proto k transformaci přidáme rotaci okolo osy x o  $180^\circ$ . Dále jelikož orientujeme držák jen jedním směrem, přidáváme rotaci okolo osy z o  $-90^\circ$ . Pokud posun středu držáku podél delší hrany označíme  $p_1$  a podél kratší strany  $p_2$ , pak celková transformace z s.s. jednoho držáku do s.s. desky je

$$T_{BH} = T_x(p_1) \cdot T_y(p_2) \cdot R_x(180) \cdot R_z(-90), \quad (15)$$

kde matice  $T_x(\alpha)$  jsou transformační matice s nulovou rotací a s translací v definované ose (zde x) o  $\alpha$  a  $R_x(\alpha)$  jsou transformační matice s nulovou translací a s rotaci okolo definované osy (zde x) o  $\alpha$  stupňů.

Finální transformaci pozice gripperu v s.s. robota tedy dostaneme jako

$$T_{RH} = T_{RC} \cdot T_{CB} \cdot T_{BH}. \quad (16)$$

## 8 Kód

V této sekci se nachází uvedení do kódové stránky daného zadání. Pro přehlednost jsou zde vypsány důležité a složitější funkce, které přispívají k fungování našeho programu.

Náš program je strukturován do několika složek. Hlavní z těchto složek je *lib*, která obsahuje všechny pomocné funkce. Zbylé složky obsahují programy pro pohyb robota *move\_commands*, pro kalibraci robota *calibration* a pro testování funkcí a teoretických znalostí jsme také vytvořili složku *tests*, v níž se nalézají všechny potřebné programy pro otestování funkčnosti kódu.

## 8.1 camera\_calibration

Pro úspěšné splnění zadání je nutno zkalibrovat kameru, pomocí níž zjišt'ujeme polohu předmětů. Prvním krokem pro kalibraci je zjištění distorzních koeficientů a matice kamery. Pro tyto účely máme Python Class CameraCalib, ve které jsou naimplementovány funkce potřebné pro kalibraci kamery.

Důležitou částí je funkce *calibrate\_using\_charuco*. Tato funkce se nakonec stala hlavní součástí našeho kalibračního postupu. Pro přesnější kalibraci jsme začali využívat charuco chess board, která umožňuje důkladnější kalibraci. Hlavními výhodami je, že charuco chess board nemusí být celá zaznamenána ve snímku, díky čemuž se lépe odhadují distorzní koeficienty na okrajích zorného pole kamery. Hlavním podílem v tomto kódu je použití knihovny *openCV*, pomocí níž jsme vyřešili velkou část kalibrace.

## 8.2 robot\_wrapper

Součástí zadání byla také knihovna umožňující komunikaci s robotem, pro snazší práci s robotem jsme vytvořili *robot\_wrapper*. Jedná se o součást námi vytvořené knihovny, přesněji Python Class Robot, ve které se odvoláváme na Python Class CRS97 z knihovny *ctu\_crs*.

V Class Robot jsou vytvořeny funkce pro snazší použití funkcí pohybu robota. Pro rychlejší manipulaci jsme také přidali funkce, které se pohnou přímo do daného místa na pracovní ploše. *move\_to\_gate*, která robota velice rychle pohne před bránu pracovního prostoru. Například v sekci 2.2 je zmínka o na míru vyrobených součástkách, které je nutno před zahájením provozu namontovat na gripper, pro tento účel slouží program *move\_to\_gate*, který robota velice rychle pohne před bránu pracovního prostoru. Více informací lze nalézt v přiložené dokumentaci kódů.

Dalším účelem Robot Class je usnadňovat pohyb, jsou zde implementovány funkce, které zajišt'ují pohyb robota na zadané souřadnice a případné výpočty k tomu potřebné.

## 8.3 hand\_to\_eye\_calib

Pro uchopení součástky robota jsme nejdříve museli zkalibrovat robota do souřadnic kamery. Tuto funkci zajišťuje Python Class HandEyeCalib a program, *hadn\_eye\_calib\_with\_moves*. V tomto programu se odehrávají dva hlavní úkony, pohyb robota v co nejvíce uniformním rozložení v oblasti zorného pole kamery a snímání gripperu robota. Výsledkem tohoto procesu jsou transformace z kamery do robota, umožňující synchronizovanou práci robota a kamery. Detailní vysvětlení kalibrace kamery a robota najeznete v sekci ??.

## 8.4 display

Jedná se o pomocný vizualizační soubor. Pro testovací účely jsme vytvořili menší Python Class, pomocí níž jsme schopni si snadno zobrazit všechny body, které máme k dispozici. Tyto funkce se hodí především během zdokonalování kódu a pro lepší pochopení fungování programu. Zároveň jsme měli možnost se nejdříve podívat na pozici, do níž jsme plánovali robota umístit, tímto způsobem jsme zamezili zbytečným nárazům robota do podložky.

Ukázku výstupu pro klasickou úlohu přenesení ze dvou rovných podložek můžete vidět na obrázku 10.

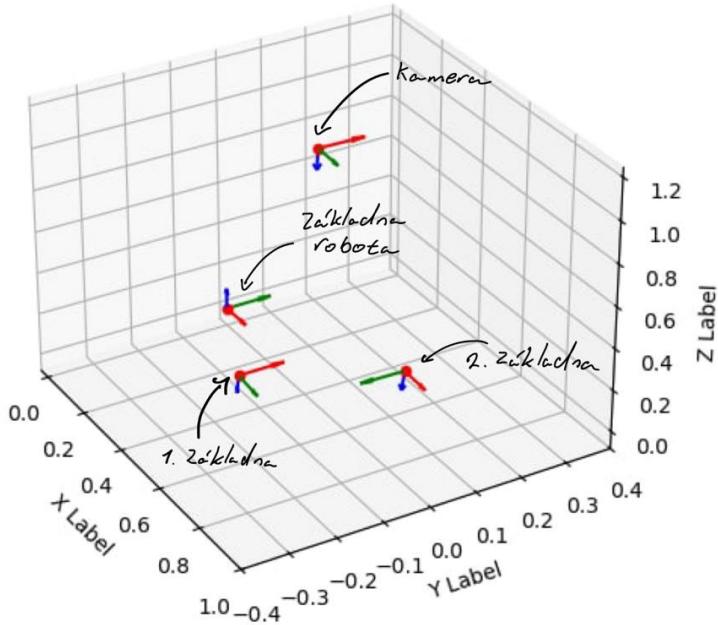


Figure 10: Ukázka fungování Display class

## 8.5 Pomocné kódy

V rámci urychlení naší práce jsem využily předem naimplementovaných kódů z předchozích úkolů v tomto semestru. Jedná se o struktury  $SE3$ ,  $SO3$ ,  $SE2$ . Funkce potřebné v našem kódu jsou naimplementovány především ve složce utils, která schraňuje funkce jak pro změnu reprezentace transformací, tak pro zápis a čtení dat.

## 9 Testy

V rámci testování jsme zjišťovali především správnou funkčnost našich programů, mnohdy se jednalo pouze o kontrolu, zda program proběhne bez vyvolání nějaké chyby.

Při testování pohybu robota jsem také velice důkladně musel otestovat přesnost robota a hlavně správnou navigaci robota, abychom se vyhnuli zbytečným kolizím robota s podložkou, případně s jinými předměty v manipulačním prostoru.

Veškeré testovací programy najeznete v přiloženém kódu společně s dokumentací pro dané programy.

Základem správného fungování robota je kalibrace kamery a hand-to-eye kalibrace. Tyto kalibrace jsme testovali sbíráním a pokládáním jediné kostičky, především v krajiných bodech, kde se nejvíce projevila distorze kamery. Pro detailnější popis se můžete podívat do sekce 3.2 a 4.

V rámci zpřesnění úchopu dílku jsme vymysleli funkci ‘wiggle wiggle‘, pomocí které dílek nejen snáze zvedneme, ale zároveň se dílek sám automaticky vystředí. Tato funkce nám umožnila sbírat a pokládat dílky s vysokou úspěšností. Funkce pohne robotem o malý kousek v kladném a záporném směru osy  $x$  a  $y$  v s.s. gripperu.

## 10 Závěr

Pro pečlivou kalibraci jsme vytvořili 78 fotografií pro určení matice kamery, kde jsem se snažily co nejvíce zachytit distorzní parametry kamery pro co nejpřesnější výpočty poloh. Následně jsem robota nechali projít přibližně 3600 pozic, ze kterých bylo použitelných 1500 obrázků pozic robota.

Ani po velice důkladné kalibraci jsme nebyli schopni dosáhnout dostatečné přesnosti v krajinách bodech zorného pole kamery. Uchýlili jsme se tedy k použití jiných metod zpřesnění. Minimalizace reprojekční chyby přinesla zlepšení, i když ne tak velké, jak bychom očekávali, což mohlo být způsobeno nepřesnou maticí kamery. Nepřesnosti jsme úspěšně kompenzovali ”vrtícím se“ pohybem naší funkcí ’wiggle wiggle‘

Překvapivé výsledky přineslo pořizování více snímků za účelem zbavit se outlierů. Při výpočtu pozice desek z obrázku jsme využili více snímků, přesněji 15, z nichž jsme vy-početly medián vektorů rovin desek. Tímto způsobem jsme dostali konzistentní výpočet pozic na desce a eliminovaly vychýlené hodnoty pozic.

Celkově jsme byli schopni přenést 4 dílky z jedné platformy do druhé, jak pro případ zvýšených, tak nakloněných desek.