

# Úklid pracovního prostoru

Aleš Trna, Minh Hoang Tran

08/01/2024

**Popis**— Předmětem této práce je popis řešení závěrečné semestrální úlohy z předmětu Robotika.

**Klíčové pojmy**— Robot, manipulátor, počítačové vidění

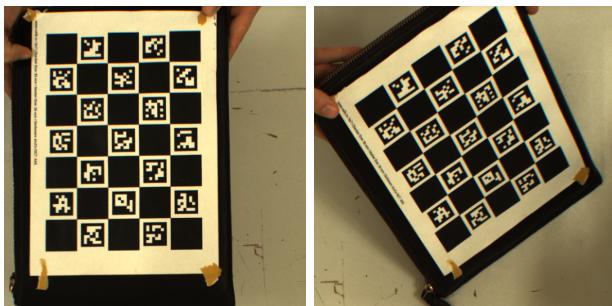
## I. ZADÁNÍ ÚLOHY

V pracovním prostoru robota CRS A465 jsou rozmístěny kostky o konstantní velikosti označené značkami typu *Aruco*. Robot je umístěn v kleci, která vymezuje jeho pracovní prostor. Ke kleci je staticky připevněna kamera, která snímá část pracovního prostoru. V pracovní scéně umístěny krabice, které slouží k uložení kostek. Úkolem je správně roztržit kostky s *Aruco* značkami tak, aby kostky označené stejnými značkami byly uloženy na stejném místě (do stejné krabice).

## II. VIDĚNÍ

### A. Kalibrace kamery

Abychom mohli detektovat, kde se kostky nachází, musíme provést kalibraci kamery. Kalibrací kamery zjistíme distorzi, kterou poté budeme moci kompenzovat, pro jednoznačné a přesné identifikování polohy *Aruco* značky vůči kameře. Princip kalibrace kamery spočívá v nafocení vzorků o známé velikosti a tvaru a následném porovnávání zkreslení výsledného obrázku s realitou. Pro jednoduchost se při této úloze obvykle používá Šachovnice o konstantním rozmeru čtverce. Pro naše řešení byla použita šachovnice s *Aruco* značkami na bílých polích o jiném, fixním rozmeru díky které byla získána přesnejší data. Pro co nejpřesnejší odhad distorze a parametrů kamery je třeba nafotit co nejvíce vzorků z co nejvíce poloh a natočení šachovnice vůči kameře.



Obrázek 1: Plochá a nakloněná Charuco deska

Na nafocených vzorcích byla detekována *Charuco* deska pomocí funkce *interpolateCornersCharuco* z knihovny *cv2*, díky

které byla získána jednoznačná poloha rohů šachovnice a *Aruco* značek. Tato data byla použita pro funkci *CalibrateCameraCharuco* ze stejné knihovny. Výsledkem byla matice kamery

$$\mathcal{C} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

Kde:

- $f_x, f_y$  jsou ohniskové vzdálenosti v konkrétních osách
- $c_x, c_y$  jsou optické středy

Zároveň byly získány distorční koeficienty, díky kterým můžeme kompenzovat distorzi kamery.

### B. Hand to Eye kalibrace

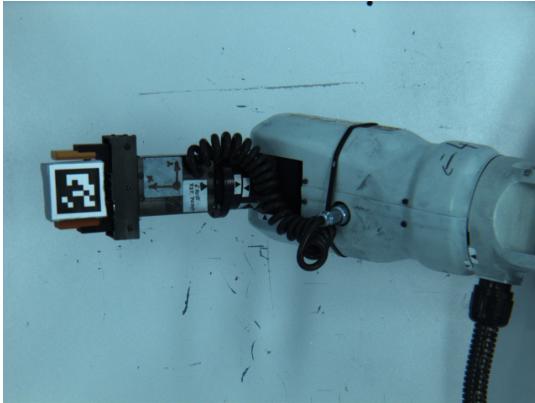
Ze zadání úlohy není jasné, kde přesně se kamera vůči robotovi nachází. Je mnoho způsobů, jak to zjistit. Nejjednodušší je změřit polohu kamery vůči pevně zvolenému bodu. Tato metoda je ale pro naše použití příliš nepřesná. Dalším způsobem je vytvoření homografické matice, která by nám mapovala scénu kamery do roviny. Předmětem této práce je však řešení nerovnninného problému, proto tento způsob není vhodný. Protože máme již zkalibrovaný obraz z kamery dokážeme díky funkcím knihovny *cv2* zjistit polohu objektů. Díky senzorům v robotu známe přesnou polohu chapadla vůči základně robota. V našem konkrétním případě není kamera připevněna na chapadle robota, ale na konstrukci v konstantní poloze vůči robotu. Řešíme tedy "Hand to Eye" problém. Na jeho řešení můžeme využít opět knihovnu *cv2*, konkrétně funkci *CalibrateRobotWorldHandEye*. Do chapadla robota vložíme objekt o známých rozměrech, u něž můžeme zjistit jeho polohu vůči kameře. V našem případě využijeme kostku s *Aruco* značkou. S robotem budeme jezdit v zorném poli kamery a zaznamenávat scénu pomocí kamery a k ní korespondující polohu chapadla vůči základně robota. Poté na fotkách detekujeme polohu *Aruco* značek a data vložíme do [zmíněné funkce](#). Řešíme tedy rovnici

$$\mathbb{A}_i \mathbb{X} = \mathbb{Y} \mathbb{B}; \quad (2)$$

Kde:

- $\mathbb{A}_i$  je  $i$ -tá naměřená transformace chapadla vůči bázi
- $\mathbb{X}$  je konstantní transformace značky vůči chapadlu robota

- $\mathbb{B}_i$  je  $i$ -tá odhadnutá transformace značky vůči kameře
- $\mathbb{Y}$  je konstantní transformace kamery vůči bázi robota.



Obrázek 2: Kostka s Aruco značkou uchopená (s konstantní transformací) v chapadle robota

Hledáme tedy transformaci  $\mathbb{Y}$ , pro její výpočet použijeme [výše zmíněnou funkci](#).

### C. Ověření správnosti transformace

Po zjištění polohy kamery vůči základně robota můžeme zjistit polohu kostek vůči robotu. Využijeme k tomu rovnici

$$T_{B \rightarrow K} = T_{B \rightarrow C} \cdot T_{C \rightarrow K} \quad (3)$$

Kde:

- $T_{B \rightarrow K}$  je transformace kostky vůči bázi robota
- $T_{B \rightarrow C}$  je transformace kamery vůči bázi robota
- $T_{C \rightarrow K}$  je transformace kostky vůči kameře
- Násobení transformací odpovídá aplikování levé transformace na transformaci pravou

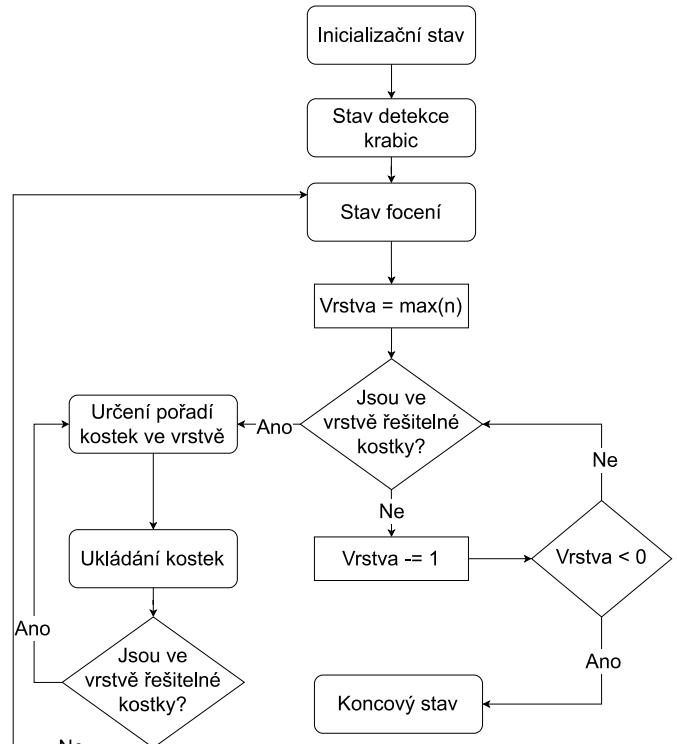
Transformaci  $T_{C \rightarrow K}$  získáme pomocí `cv2.aruco` funkcí `detectMarkers` a `estimatePoseSingleMarkers`. Jako ověření přesnosti kalibrace porovnáme změřenou a spočtenou polohu kostek vůči bázi robota. A provedeme relativní pohyb kostkou v pracovním prostoru. Dalším způsobem ověření bylo měření známých vzdáleností mezi více kostkami.

## III. NÁVRH ŘEŠENÍ

Program je navrhnut jako stavový automat. Jeho zjednodušený vývojový diagram můžeme vidět na obrázku 3. Jednotlivé stavy algoritmu podrobně rozobereme v této kapitole.

### A. Inicializační stav

Robot provede inicializaci chlapadla a absolutní kalibraci polohy. Poté odjede do bezpečné pozice - tzv. "soft home".



Obrázek 3: Vývojový diagram popisující základní stavy stavového automatu

### B. Stav focení

Robot se dostane do takové polohy, aby jeho ramena, nebo chlapadlo nezasahovalo do zorného úhlu kamery. Protože je kamera na fixním místě, můžeme pro toto použít absolutní polohu robota. Poté se zachytí fotografie scény pomocí kamery na rámu robota. Tato fotografie je převedena do grayscale a podrobena funkcí `cv2`, které na ní detekují Aruco značky. Výstupem těchto funkcí je absolutní poloha značek vůči bázi robota. Polohy těchto značek jsou diskretovány v ose  $z$ , za účelem možnosti řešit problém po jednotlivých vrstvách - to znamená že kostky v první vrstvě dostanou index 0, kostky ve druhé 1 atd...

Pokud jde o první stav focení od začátku běhu programu provede se zároveň **Stav detekce krabic**.

### C. Stav detekce krabic

Protože id Aruco značek krabic jsou odlišné od id značek kostek a tato id jsou předem známá, můžeme použít detekci krabic na základě id značky. Při prvním focení programu se označí všechny detekované id z množiny id pro krabice jako "krabice". Předpokládá se přitom, že jsou všechny dostupné krabice viditelné v zorném úhlu kamery od samého začátku běhu programu. K těmto krabicím jsou postupně přiřazovány id kostek, které byly pozorovány. Pokud je v pracovním prostoru méně druhů kostek a více krabic budou všechny kostky stejného druhu uloženy do stejných krabic a některé krabice zůstanou volné. V druhém případě bude program přiřazovat kostky ke krabicím s nejnižším počtem druhů již přiřazených kostek. Všechny další kostky se stejným id budou

přiřazeny do stejné krabice. Po určení id krabic se uloží jejich pozice vůči bázi robota a kostky, jež do dané krabice patří se přesouvají na příslušné ( $x, y$ ) souřadnice.

Aby se předešlo detekci již roztržděných kostek v krabicích, vytvoří se kolem původní polohy pomyslný kruh se středem v id krabice a poloměrem 10 cm v rovině  $xy$ . Pokud souřadnice ( $x, y$ ) některé nově detekované značky leží uvnitř jednoho z těchto kruhů, nebude znova tříděna. Poloměr 10 cm jsme zvolili díky předem známému rozměru krabic.

#### D. Třídění podle vrstev

Pokud známe pozice krabic, můžeme se přesunout k třídění jednotlivých kostek. Po detekci a diskretizaci kostek v ose  $z$  se vyberou se kostky z nejvyšší detekované vrstvy a začne se řešit konkrétní vrstva. Zároveň se uloží nejvyšší vrstva kostky při startu programu. V rámci bezpečného, bezkolizního pohybu robota všechny manipulace v rovině  $xy$  probíhají nad touto vrstvou. Pokud současná nejvyšší vrstva neobsahuje kostku která by mohla být uchopena, začne se hledat řešení o vrstvě níže. Tento proces se opakuje buď do nalezení řešitelné kostky, nebo dosažení nulté vrstvy.

#### E. Pořadí kostek v jednotlivé vrstvě

Máme vybrané kostky z jedné konkrétní vrstvy kterou hodláme roztržít. Nyní musíme určit pořadí, v jakém budeme kostky z dané vrstvy brát, a zároveň určit, které kostky jsou řešitelné. Jako první filtr použijeme matici vzdálenosti, kterou sestavíme jako:

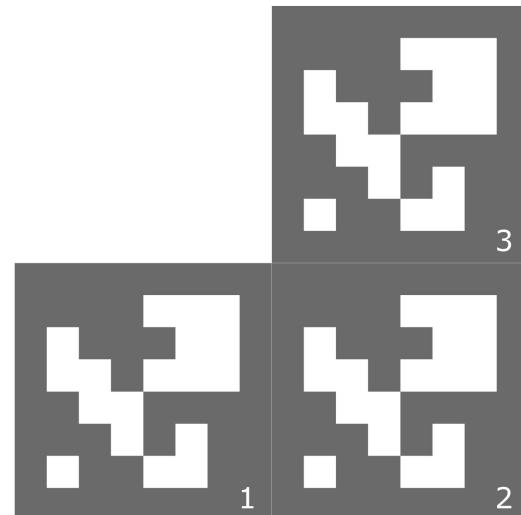
$$\mathbb{D} = \begin{pmatrix} d_{1,1} & d_{1,2} & \dots & d_{1,n} \\ d_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ d_{n,1} & \dots & \dots & d_{n,n} \end{pmatrix} \quad (4)$$

kde  $d_{i,j}$  je vzdálenost mezi kostkou s indexem  $i$  kostkou a indexem  $j$

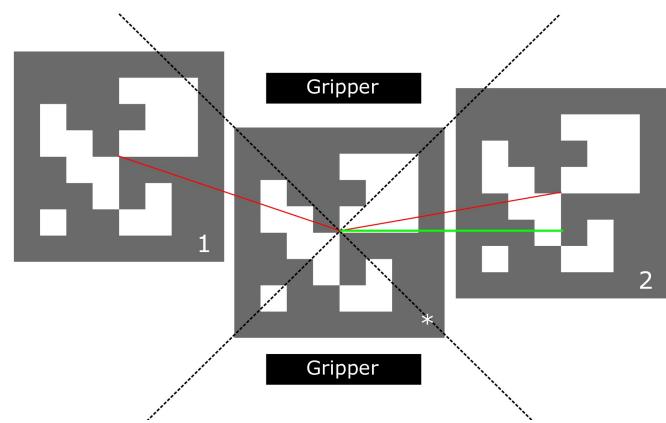
Pokud na **matici vzdáleností** provedeme sumaci řádků (nebo sloupců, matice je symetrická), dostaneme součet vzdáleností všech kostek od kostky na indexu  $i$ . Kostky budeme třídit právě podle tohoto kritéria. Budeme brát kostky od "nejvzdálenější" a pokud bude kostka uchopitelná roztrždíme ji. Takto budeme postupovat až do té doby, než roztrždíme všechny kostky. Jak ale zjistíme, že je kostka uchopitelná a v jakém úhlu ji máme chytit?

V našem řešení jsme využili toho že Aruco markery jsou nesouměrné a můžeme tedy zjistit jejich rotaci vůči ose procházející kamerou. U každé kostky najdeme v **matici vzdáleností** kostky, které jsou jejímu středu blíže než 70 mm. Vytvoříme si dočasné "pole orientací", do kterého budeme ukládat relativní pozice kostek v blízkosti. U všech těchto kostek spočítáme vektor transformace z naší kostky a vynormujeme jej. Spočítáme skalární součin tohoto vektoru s orientací naší kostky (jednotkovým vektorem ve směru osy  $x$ ) a jeho absolutní hodnotu porovnáme s hodnotou 0.70, která odpovídá skalárnímu součinu jednotkového vektoru  $\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$  - tedy jednotkové ose prvního a třetího kvadrantu.

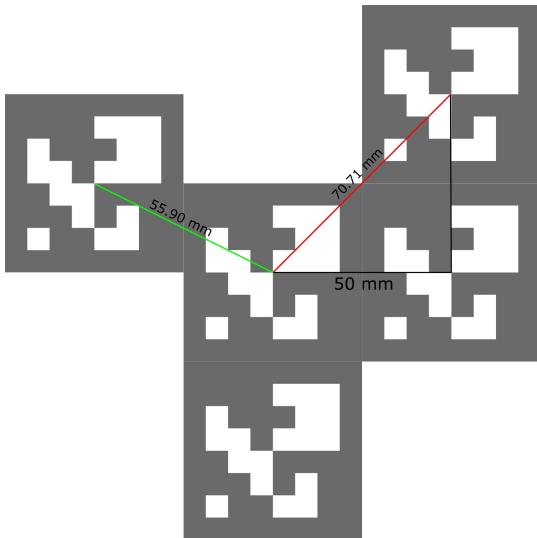
Pokud je hodnota našeho skalárního součinu vyšší, uložíme do pole orientací 1, pokud ne, uložíme nulu. Toto provedeme pro všechny kostky, které jsou v okolí právě zkoumané kostky menší než 70 mm. Pokud pole orientací obsahuje samé '1' víme, že kostku musíme chytit z pohledu kamery "svisle", pokud samé '0', musíme kostku uchopit "vodorovně". Pokud pole obsahuje jak '0' tak '1', víme, že kostku v současnosti nemůžeme uchopit (jako například kostku 2 na obrázku 4). Buď musíme nejdříve odstranit ostatní kostky, nebo jde o neřešitelnou úlohu. Tato logika je vidět na obrázku 5. Protože absolutní hodnota skalárního součinu kostky "\*" s kostkami 1 a 2 je větší, než 0.7, je kostka "\*" uchopitelná "svisle". Toto je však pouze ukázkový příklad pro demonstraci logiky úchopu. Pokud by měl robot reálně řešit tuto situaci, roztržil by nejdříve kostky 1 nebo 2, protože součet vzdáleností od ostatních kostek je pro ně větší, než u kostky "\*".



Obrázek 4: Uchopitelné kostky 1,3 a neuchopitelná kostka 2 v dané konfiguraci kostek



Obrázek 5: Logika úchopu kostky. Pro kostku "\*" se provádí skalární součiny normovaných "červených vektorů" s normovaným "zeleným vektorem"



Obrázek 6: Rozměry aruco kostek

#### F. Třídění kostky

Nyní, když máme určené pořadí, v jakém budeme kostky třídit, můžeme začít se samotným tříděním. Ze stavu určujícího pořadí máme přesnou polohu kostek a pořadí, ve kterém je máme brát. Robot tedy podle tohoto pořadí nejdříve uchopí kostku na konkrétní pozici, vyjede s ní nad nejvyšší detekovanou vrstvu při prvním focení a přemístí do předem určené krabice. Do které krabice konkrétní kostka půjde je popsáno [zde](#). Pokud leží kostka mimo dosah robota, je vypsána chybová hláška, robot odjede do pozice "soft home" a je ukončen běh programu. Kostky jsou vždy uchopovány z vrchu, kolmo na Aruco značku.

Po roztržení jedné kostky pokračuje podle pořadí daném ve stavu určujícího pořadí kostek do té doby, než všechny kostky roztrží, nebo všechny zbývající kostky jsou neuchopitelné. Poté se přesune do stavu focení.

#### IV. KOMPLIKACE PŘI ŘEŠENÍ ÚLOHY

##### A. Neřešitelné vrstvy a virtuální kostky

Pokud robot není schopen vyřešit nějakou vrstvu, bude podle obrázku 3 postupovat do nižší vrstvy, ve které bude dále postupovat podle výše popsaného algoritmu. Do této vrstvy však musíme doplnit "virtuální kostky", o kterých robot neví, protože na nich stojí "neřešitelné" kostky z vyšší vrstvy. Tyto kostky označíme jako rovněž neřešitelné, ale zbytek vrstvy budeme řešit podle stejného postupu. Na konci nám zbydou opět tyto "virtuální" kostky plus jakékoli další "skutečné" kostky z této vrstvy. z nich opět vyrobíme "virtuální kostky", které přidáme do nižší vrstvy.

#### V. PŮVODNÍ ZPŮSOBY ŘEŠENÍ, OD KTERÝCH SEŠLO

##### A. Opuštění zorného úhlu kamery

Původně byla v návrhu řešení pro opuštění zorného úhlu kamery při focení použita poloha kloubů  $[0 \ 0 \ 0 \ 0 \ 0 \ 0]$ , za účelem rychlosti řešení byla tato poloha předělána do fixní rotace nultého kloubu, který je

schopen násobně větší rychlosti, než všechny ostatní klouby robota. Před implementací tohoto řešení panovaly obavy, zda velké zrychlení v jednom kloubu nemohou rozkalibrovat robota - tedy zda moment síly nebude moc velký. Naštěstí se prokázala tato obava jako neopodstatněná a toto efektivnější řešení mohlo být použito.

#### B. IKT a dosah robota

Původně bylo řešení navrhnuto pouze na ukládání do krabic s tím, že kostka je ve kterékoli fázi pohybu vodorovná s rovinou stolu v pracovním prostoru robota. Protože jsme však krabice umisťovali do rohů zorného úhlu kamery, kde se občas stávalo, že poloha bylo mimo dosah robota, byla přidělána funkce o ukládání v jakémkoliv úhlu. Priorizována je opět poloha "kolmo na pracovní stůl", nicméně pokud neexistuje konfigurace robota, ve které by této polohy byl schopen dosáhnout, vyhledá ve všech konfiguracích takovou polohu, aby do krabice byl schopen kostku umístit.

#### VI. ZÁVĚR, DISKUZE A MOŽNOSTI LEPŠÍHO ŘEŠENÍ

Na této úloze jsme si vyskoušeli úlohu z praktické robotiky. Většina problémů při řešení vznikla při nastavování "vidění". To, že tomu bylo tak i při takto jednoduché úloze objasňuje, proč se v moderní robotice jakýkoliv problém, který jde udělat bez vidění řeší bez vidění. Nejpřekvapivější věc pro nás byla, že při vyšším počtu vzorků při kalibraci kamery a HandEye kalibraci bylo dosaženo horších výsledků než s menším počtem vzorků.

Při testování jsme zjistili, že naše řešení kolabuje na testech, když se jedná o problém se 7 a více vrstvami, protože kamera nedokáže přesně odhadnout polohu kostky v ose  $z$ . Současné řešení by se dalo vylepšit o kolizní funkci, která by umožňovala robotu hledat efektivnější trajektorie, a nezvedat kostky nad úroveň té nejvyšší. Zároveň by se mohlo implementovat ukládání kostek do mřížky do krabic. Při testování jsme také přišli na to, že by možná bylo lepší použít při počítání pořadí třídění kostek v dané vrstvě vždy jen kostky, které ještě nejsou roztržděné. Tímto by se sice zvýšila výpočetní náročnost algoritmu, nicméně výsledky by byly nejspíše optimálnější. Dala by se také použít jiná metrika, např. třídit kostku nejbližše od současné polohy robota.