

実習の目的

- 教材用ロボットを用いて、実際のティーチング作業を学ぶ
- ティーチング作業でロボットが担える作業を学ぶ
- 画像処理技術に触れる
- 実装・ハンズオンを通して技術的な提案・問題解決能力を養う

今日の実習内容

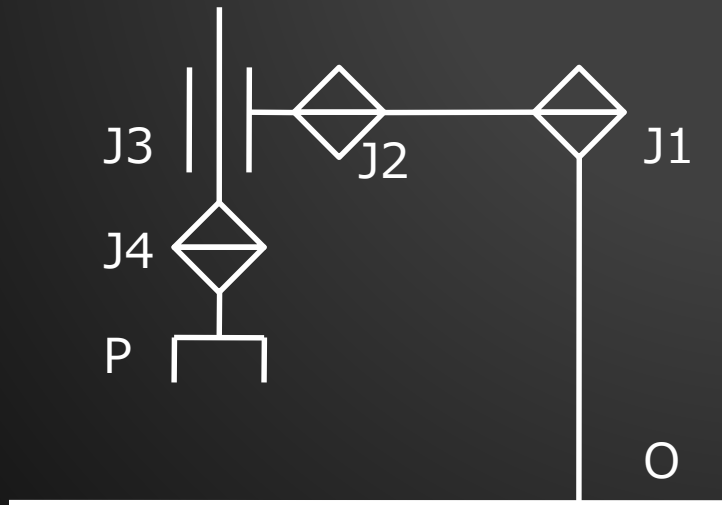
- スカラロボットをティーチング（教示）して動かす
 - 手先位置と姿勢を考慮したティーチング
- ワークの積み上げとパレタイジングをティーチングする
 - ティーチングする姿勢のコツを掴む
 - 作業あたりターンアラウンドタイムを意識してみる（どうティーチングすると効率的で早いか）
- 画像処理（撮像・色抽出）をおこなう
- 画像処理を用いて自律パレタイジングをプログラミングしてみる

おさらい ～ティーチング作業について～

- 手先の位置（と姿勢）を指定してロボットを動かすこと
 - 手先位置を指定する方法
（これだけでは手先の姿勢は決まらない）
 - 関節の角度を指定して一意な姿勢を得る方法
 - 手先姿勢を指定して、それを実現しうる関節の角度を得る方法

スカラロボットの構造

- 手先位置はJ1～J3で決定
 - 回転・回転・並進の機構をもつ
 - ハンドは常に鉛直下向き
 - 手先（ハンド）の姿勢はJ4で最終決定

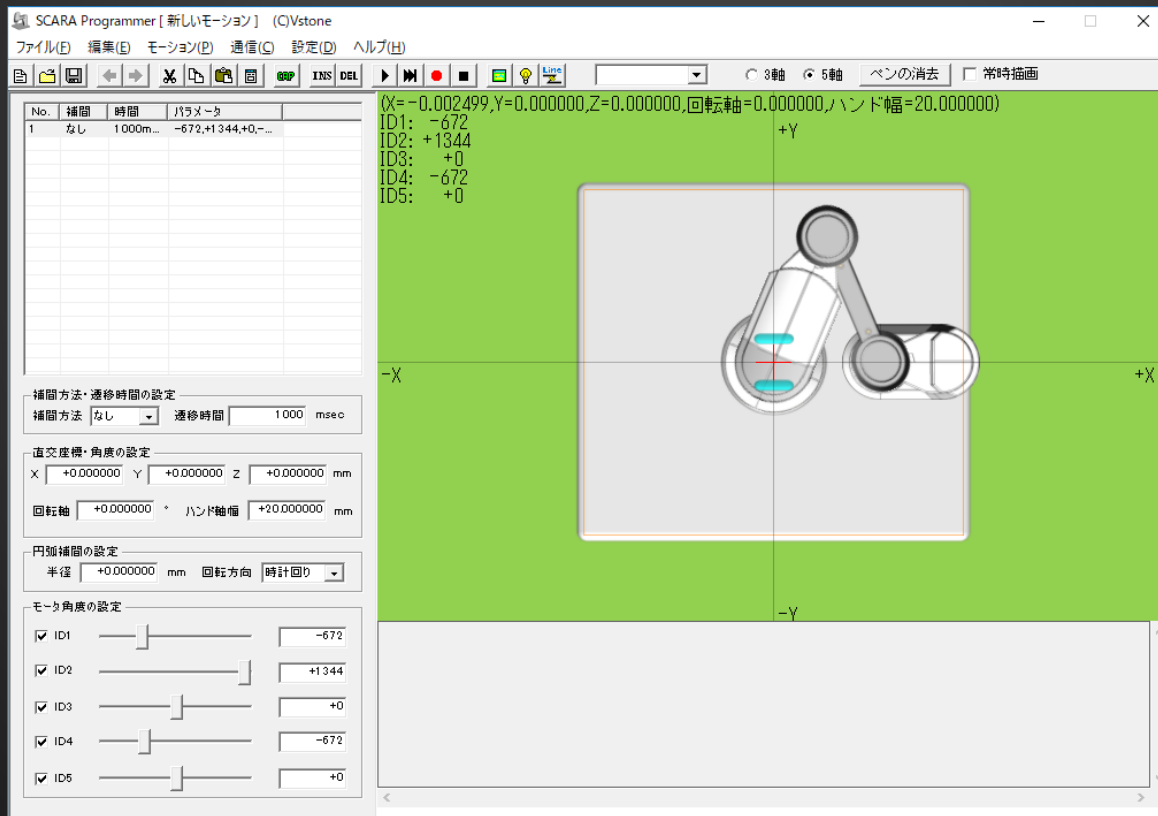


さっそく動かしてみる

- 事前準備（ドライバのインストール）
 - CP2110_4_Windows.exeを起動して実行する.
（必要に応じて再起動する）
- 制御アプリケーションを起動する
 - 「SCARAProgrammer」を起動する
 - 起動するとシミュレータ画面が表示される

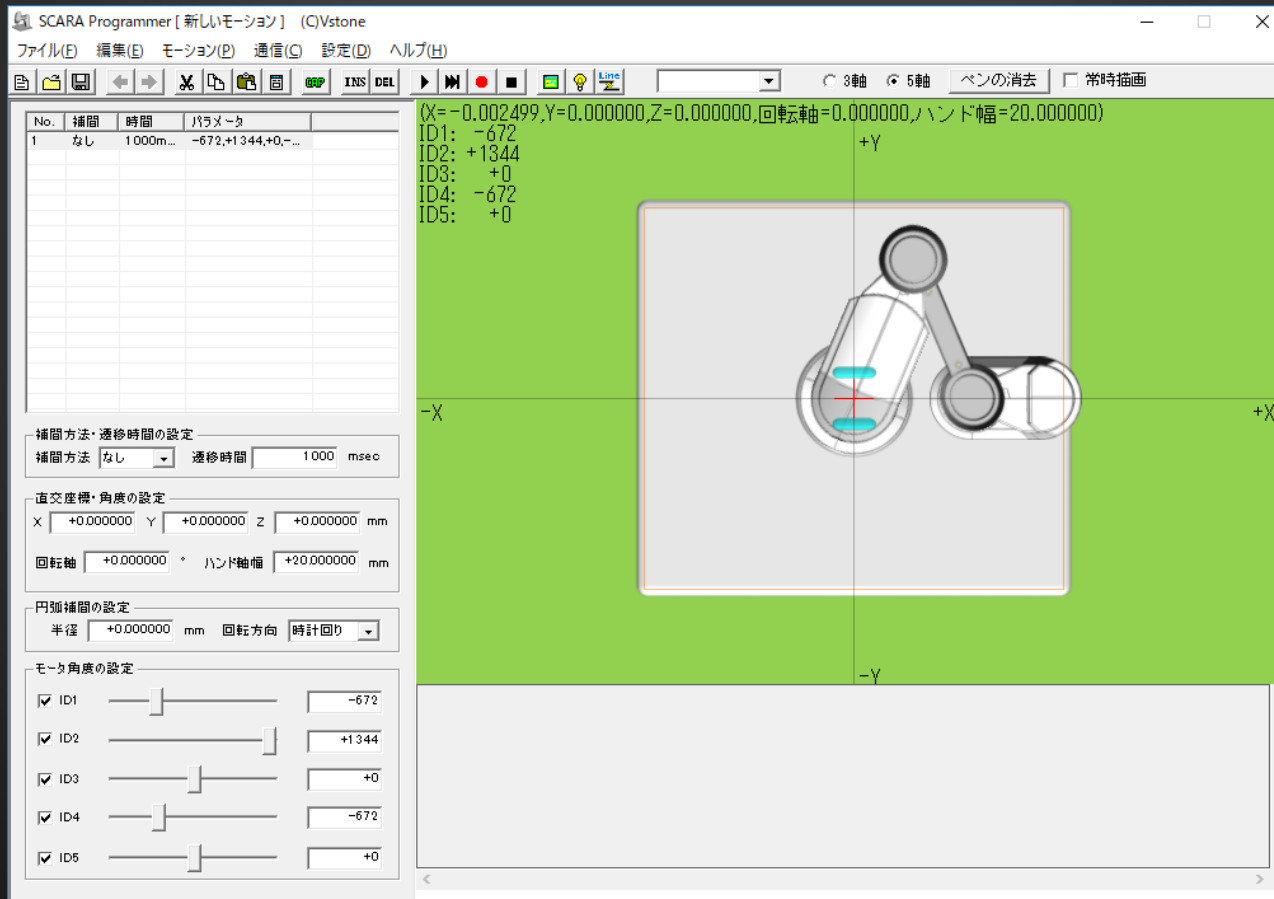
作業手順

- 制御アプリケーションを起動する
 - デスクトップ上の「SCARAProgrammer」を起動する
 - 起動するとシミュレータ画面が表示される



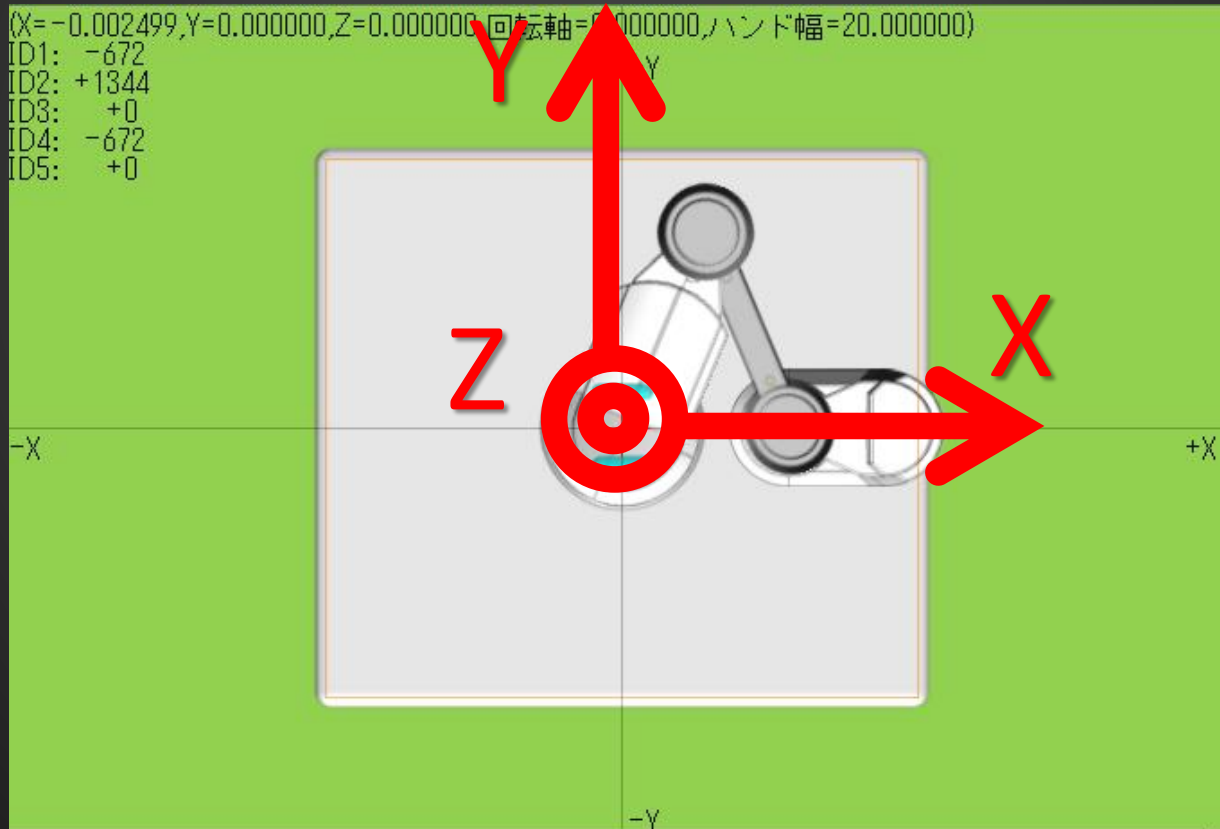
作業手順

- 制御アプリケーションを起動する
 - デスクトップ上の「SCARAProgrammer」を起動する
 - 起動するとシミュレータ画面が表示される



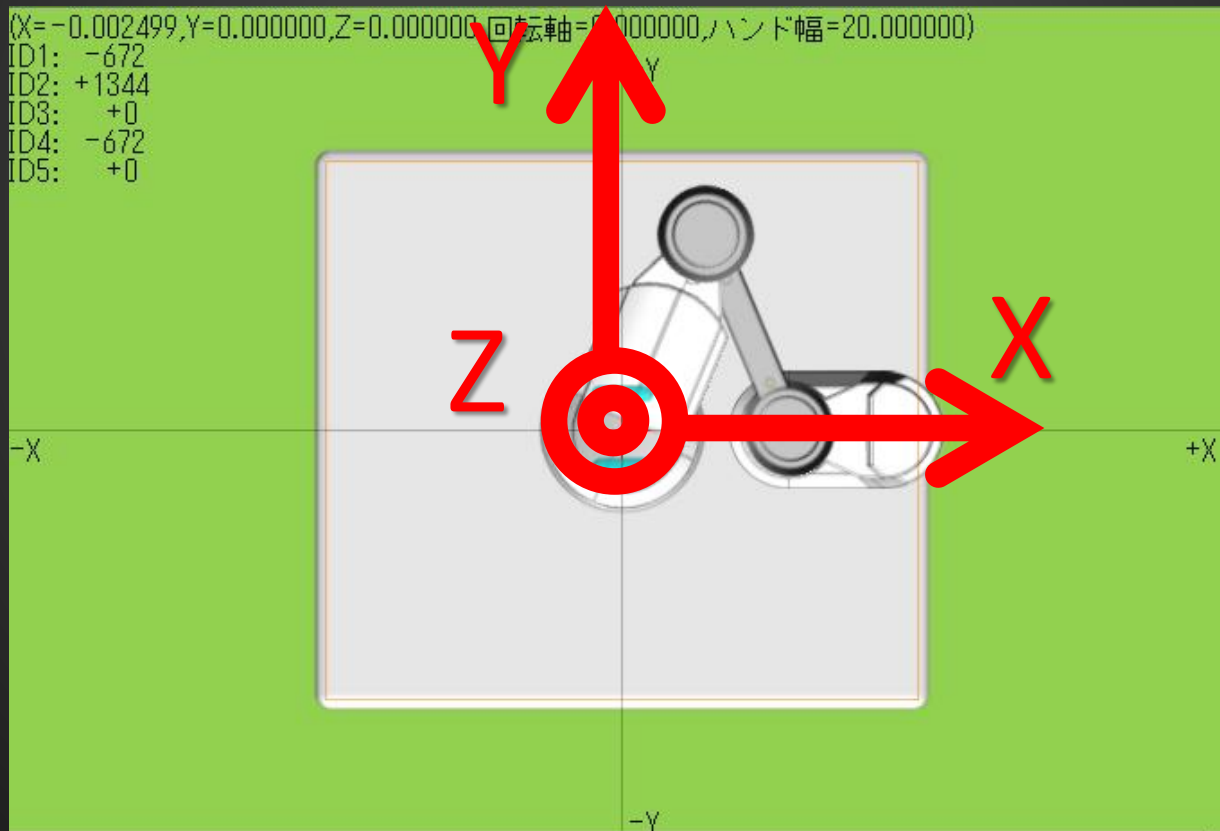
座標の取り方

- 右手系（鉛直上向きを+Z）



座標の取り方

- 右手系（鉛直上向きを+Z）
 - 原点は盤面中央なので注意

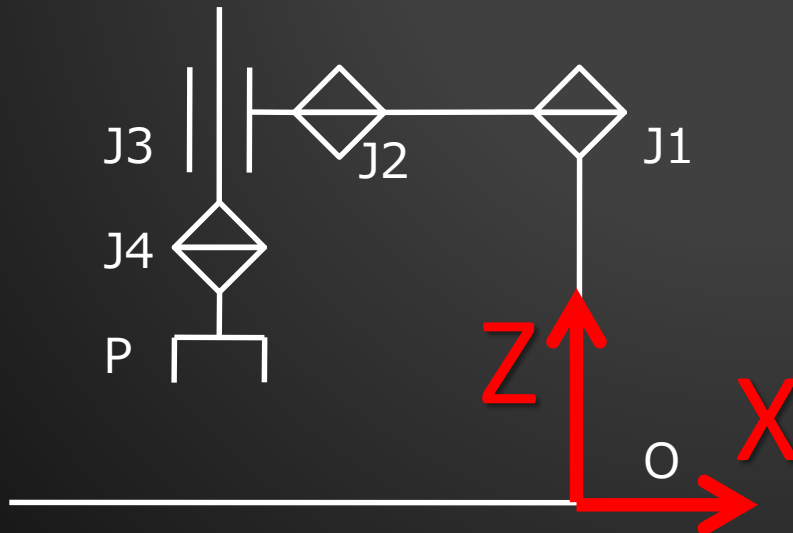


順運動学方程式の定義

- 関節角度から手先位置を算出する
 - Z軸はJ3のみによって決まる

$$x = l_1 \cos \theta_1 + l_2 \cos (\theta_1 + \theta_2)$$

$$y = l_1 \sin \theta_1 + l_2 \sin (\theta_1 + \theta_2)$$



順運動学方程式の定義

- 関節角度から手先位置を算出する
 - Z軸はJ3のみによって決まる

$$x = \ell_1 \cos \theta_1 + \ell_2 \cos (\theta_1 + \theta_2)$$

$$y = \ell_1 \sin \theta_1 + \ell_2 \sin (\theta_1 + \theta_2)$$

このロボットのパラメータ

$$\ell_1 = 80.0 \text{ [mm]}$$

$$\ell_2 = 80.0 \text{ [mm]}$$

ティーチング実習

- 事前に配布したロボットの取扱説明書に記載された課題に取り組んでいただきます

画像処理と 自律パレタイジング

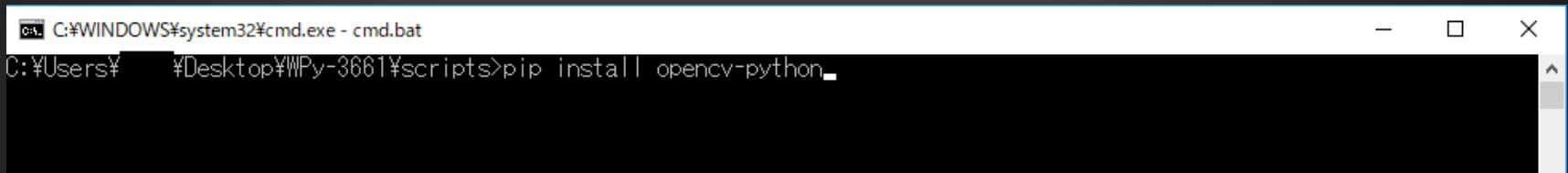
実習課題

- カメラで撮影した把持対象物（ワーク）をロボットで把持して特定の場所に搬送する
 - カメラからの画像取得
 - 画像処理・対象物抽出技術
 - ※今回は色による抽出を行う
 - 座標変換アルゴリズム（代数学）
 - カメラで撮影したワークの座標をロボットの座標系に変換する
 - 座標指定によるロボット制御
 - TCP/IP通信に関する知識
 - Pythonを用いたプログラミング技術

opencv-pythonのセットアップ

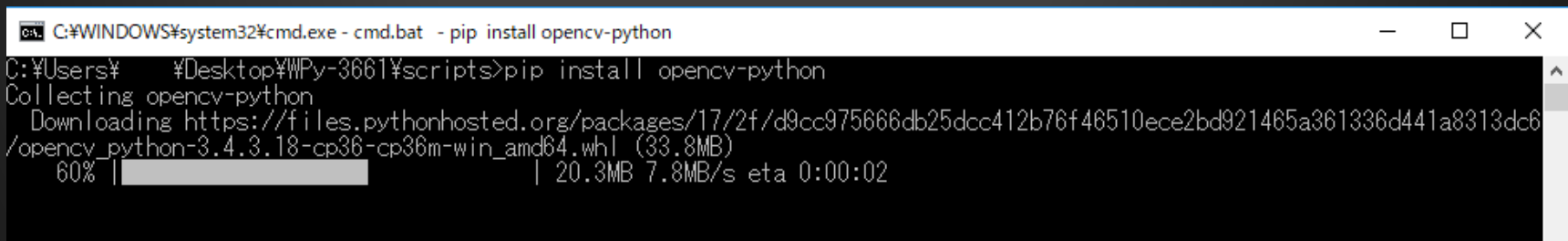
- WinPythonへのopencv-pythonのインストール手順

- (1) WinPythonを展開したディレクトリ内にある「WinPython Command Prompt.exe」を起動する
- (2) 「**pip install opencv-python**」 と入力しEnterキーを押下する



```
C:\WINDOWS\system32\cmd.exe - cmd.bat
C:\Users\%username%\Desktop\WPY-3661\scripts>pip install opencv-python_
```

- (3) ダウンロードとインストールが始まるので暫く待つ



```
C:\WINDOWS\system32\cmd.exe - cmd.bat - pip install opencv-python
C:\Users\%username%\Desktop\WPY-3661\scripts>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/17/2f/d9cc975666db25dcc412b76f46510ece2bd921465a361336d441a8313dc6/opencv_python-3.4.3.18-cp36-cp36m-win_amd64.whl (33.8MB)
    60% |██████████| 20.3MB 7.8MB/s eta 0:00:02
```

- (4) 「Successfully installed opencv-python-・・・」が表示されればインストール完了（表示されない場合は実習当日対応します）

opencv-pythonのセットアップ

- OpenCV:画像処理ライブラリ
 - Intelが開発したオープンソースのライブラリでC/C++により記述
 - 様々なプラットフォーム, 開発言語への移植がなされている
- 画像処理 (Image Processing)
 - 勾配、エッジ、コーナー (Gradients, Edges and Corners)
 - サンプリング、補間、幾何変換 (Sampling, Interpolation and Geometrical Transforms)
 - モルフォロジー (英語版) 演算 (Morphological Operations)
 - フィルタと色変換 (Filters and Color Conversion)
 - ピラミッドとその応用 (Pyramids and the Applications)
 - 画像分割、領域結合、輪郭検出 (Image Segmentation, Connected Components and Contour Retrieval)
 - 画像と形状のモーメント (Image and Contour Moments)
 - 特殊な画像変換 (Special Image Transforms)
 - ヒストグラム (Histograms)
 - マッチング (Matching)
 - ラベリング (Labeling) : OpenCV 3.0以降
- 構造解析 (Structural Analysis)
 - 輪郭処理 (Contour Processing)
 - 計算幾何 (Computational Geometry)
 - 平面再分割 (Planar Subdivisions)
- モーション解析と物体追跡 (Motion Analysis and Object Tracking)
 - 背景統計量の累積 (Accumulation of Background Statistics)
 - モーションテンプレート (Motion Templates)
 - 物体追跡 (Object Tracking)
 - オプティカルフロー (Optical Flow)
 - 推定器 (Estimators)
- パターン認識 (Pattern Recognition)
 - 物体検出 (Object Detection)
- カメラキャリブレーションと3次元再構成 (Camera Calibration and 3D Reconstruction)
 - カメラキャリブレーション (Camera Calibration)
 - 姿勢推定 (Pose Estimation)
 - エピポーラ幾何 (Epipolar Geometry)
- 機械学習
 - 単純ベイズ分類器 (Naive Bayes Classifier)
 - k近傍法 (K Nearest Neighbors)
 - サポートベクターマシン (SVM)
 - 決定木 (Decision Trees)
 - ブースティング (Boosting)
 - Random forest (Random forest)
 - EMアルゴリズム (Expectation-Maximization)
 - ニューラルネットワーク (Neural Networks)
- ユーザインタフェース
 - シンプルGUI (Simple GUI)
 - 画像の読み込みと保存 (Loading and Saving Images)
 - ビデオ入出力 (Video I/O)
 - OpenGL/Direct3D相互運用

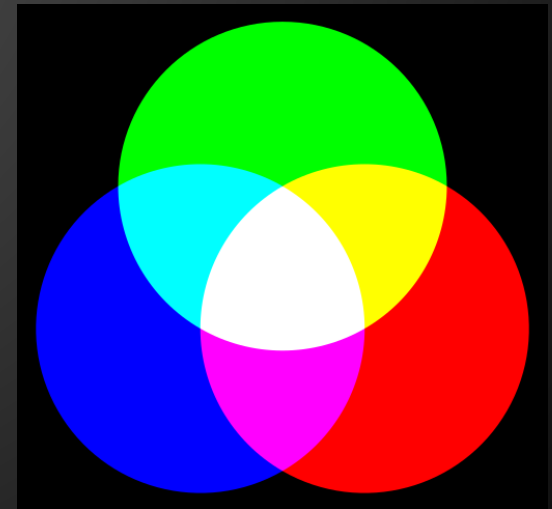
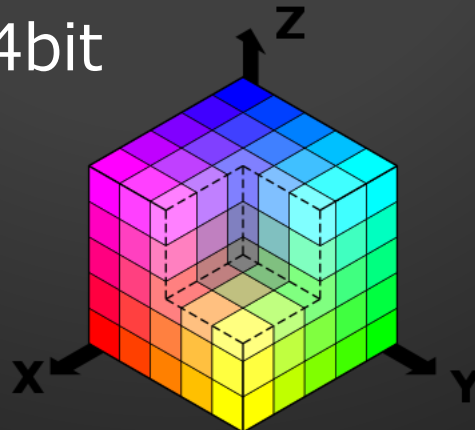
画像処理で対象物を抽出する

- 色に基づいて対象物を抽出する
 - 色空間を用いたセグメンテーション
 - 今回扱う方法
- 形状に基づいて対象物を抽出する
 - 形状マッチング（楕円・円・矩形・凸包）
 - 機械学習

色空間の話

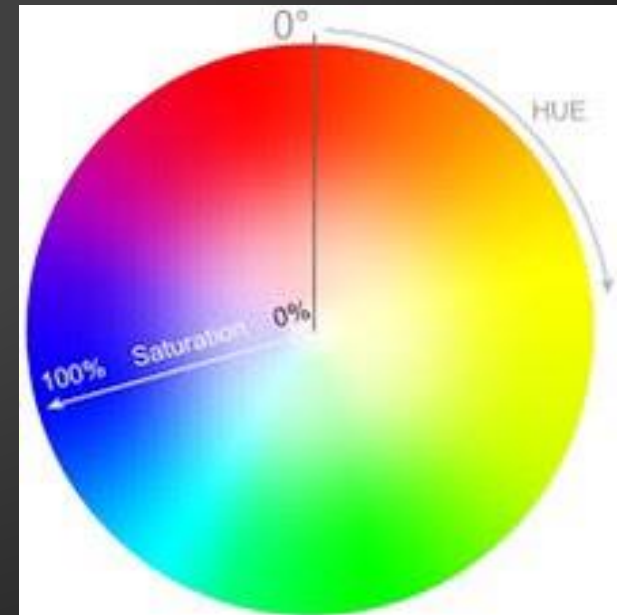
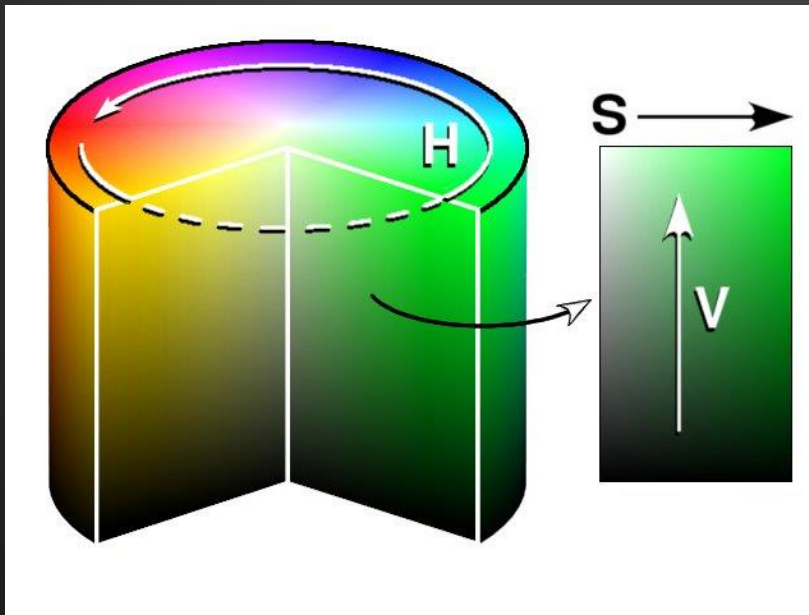
- RGB色空間

- 1画素に定義される色をRGBの3要素により表現する
- 色は全部足すと白になる（黒は光がないことを意味する）
- 1ch/8bitの場合は0～255(0x00～0xFF) の数値で表現
 - RGBの3chで24bit



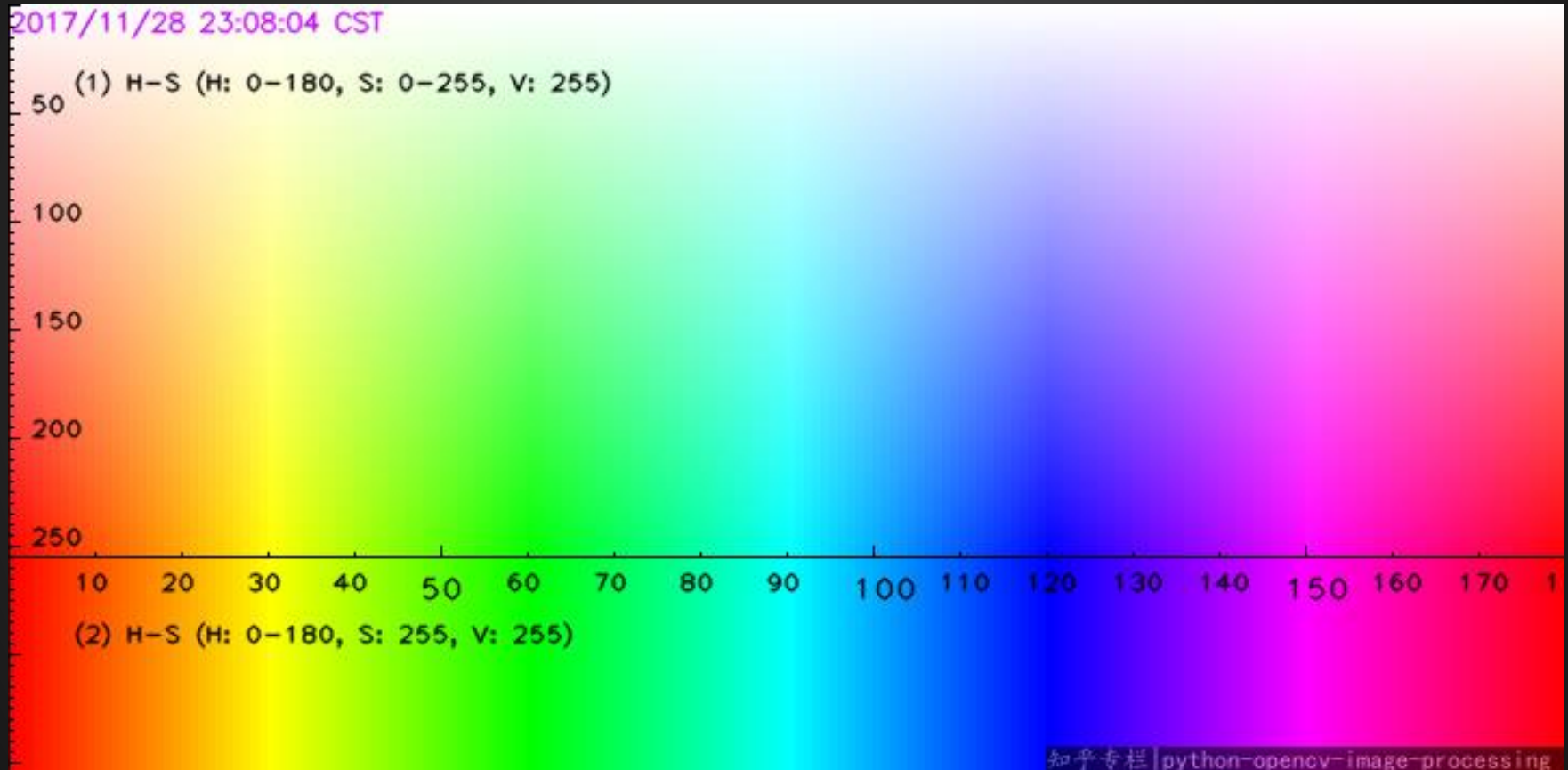
色空間の話

- HSV色空間
 - 色相(Hue)、彩度(Saturation)、明度(Value · Lightness · Brightness)の3要素で色を表現する方法
 - 円筒または円錐状の座標系に定義される



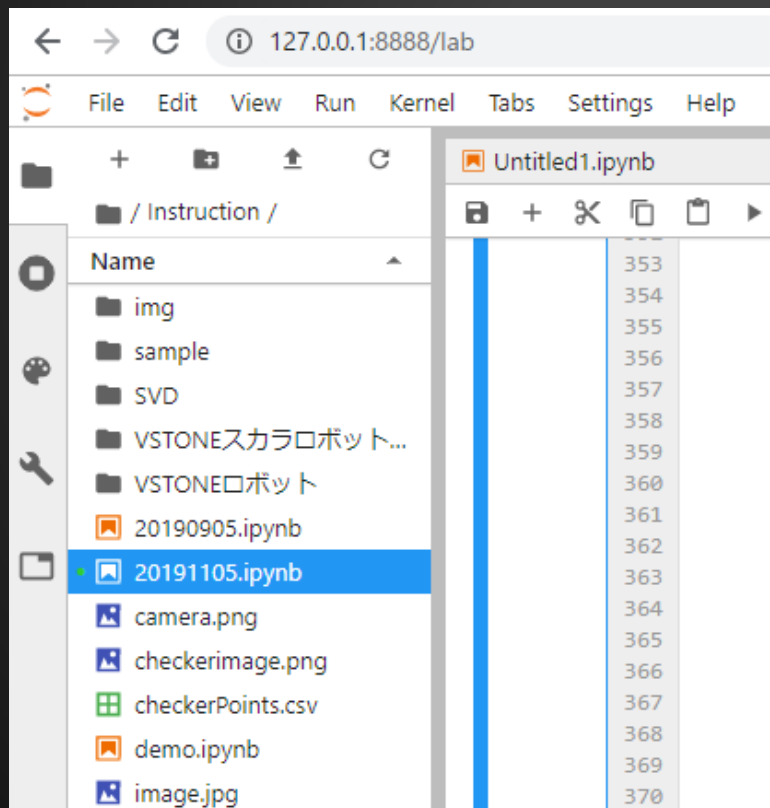
opencv-pythonにおけるHSV色空間マップ

- 横軸H, 縦軸S

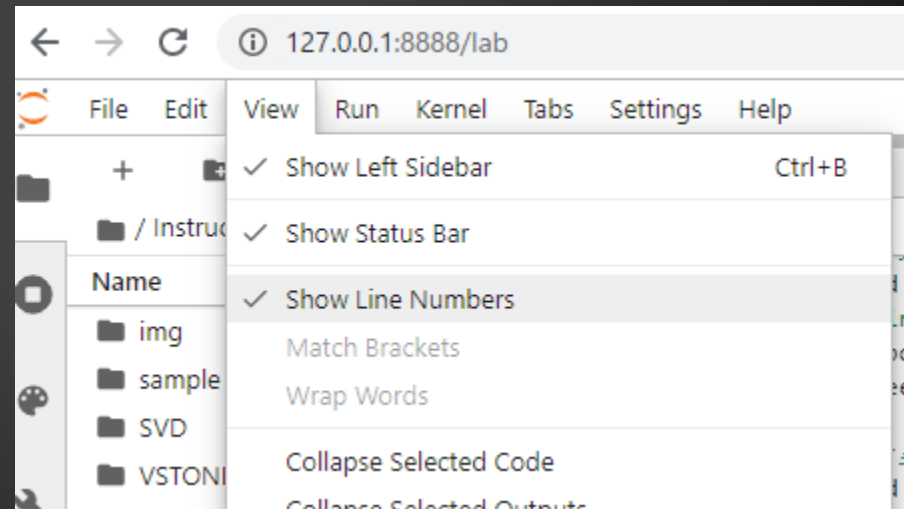


実習ソース等のセットアップ

- 実習サイトからダウンロード済みのファイルをWinPythonのインストールディレクトリ直下の「notebook」ディレクトリに移動する



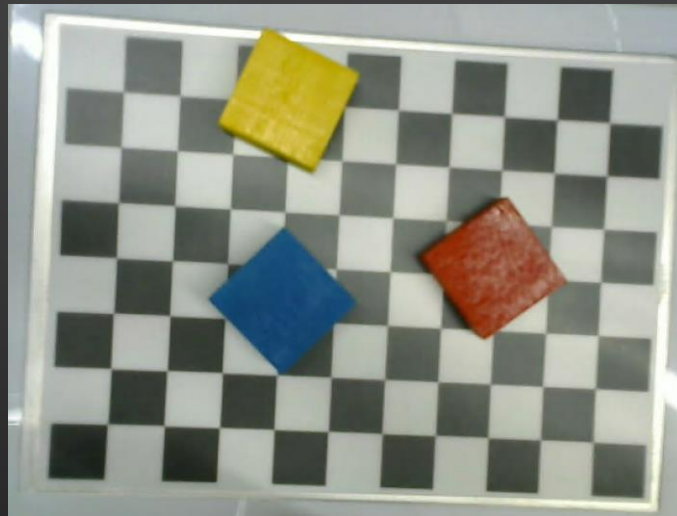
- ダブルクリックすると編集画面が開く
- [View]→[Show Line Number]にチェックを入れて、エディタに行番号が表示されるようにする



課題（１）

1.カメラから画像を取得する

- L.18 `capture_camera()`関数
- 一番目のソースコードを実行するとカメラウィンドウが表示される
- 引数の「camid」は本体に接続されたカメラのインデックス
 - フロントカメラがあるPCの場合、camid=0としてフロントカメラが割り当てられることが多いため、USB接続のカメラがcamid=1以降となる。
 - この場合、「`capture_camera(camid=1, mirror=True, size=None)`」などと修正する
- 下記画像のように、チェッカーボードがすべておさまるようにカメラの姿勢やスタンド位置を修正する



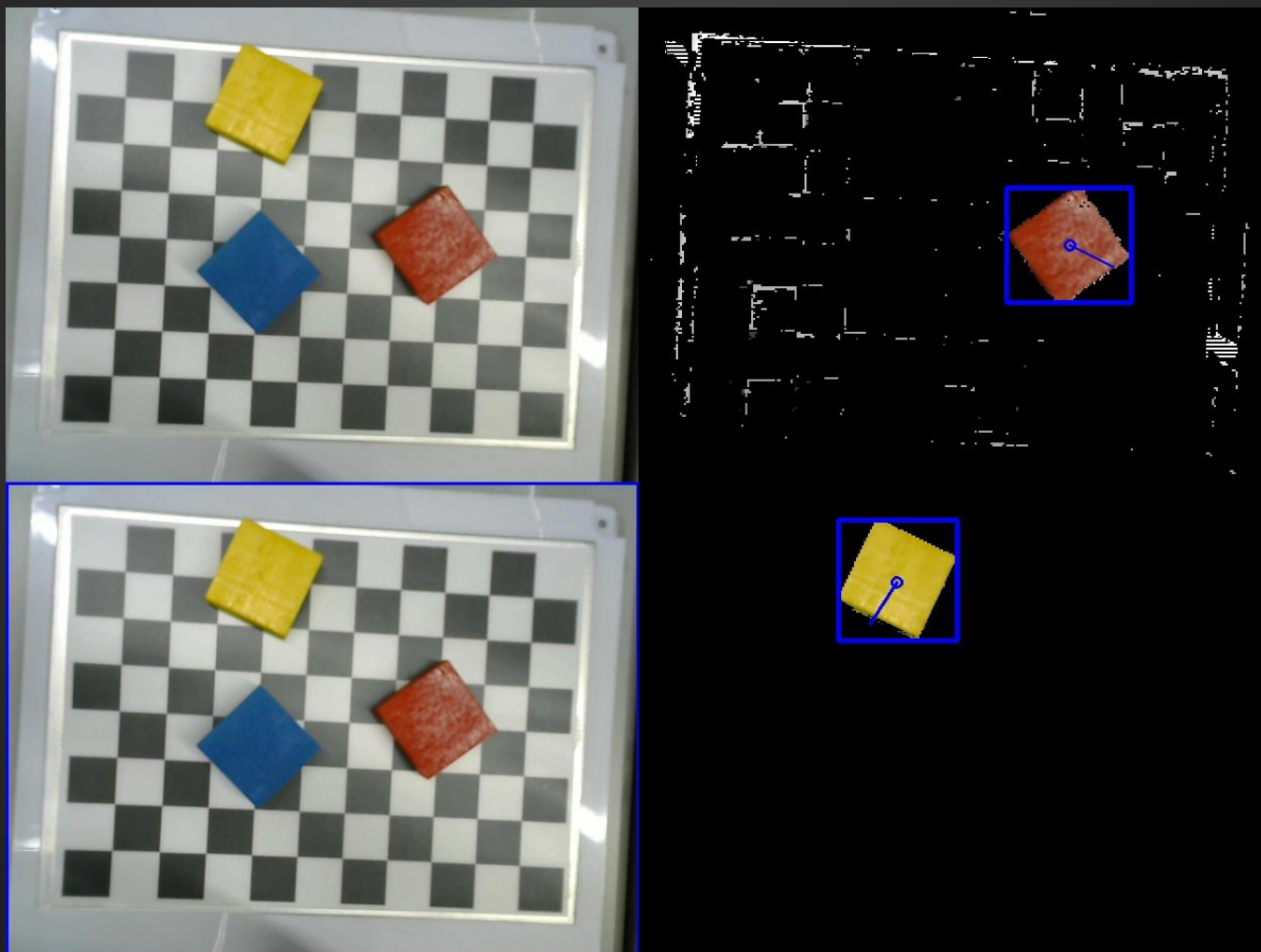
課題（２）

2.HSV色空間を用いて「赤・黄・青」三色の抽出を行う

- L.70 extractcolorHSV()関数の内の設定値を探索・変更する
- カメラから画像を取得しながら設定値を変更してみる
- それぞれの色に対応するHSVの値を得る
 - それぞれ別の人を担当してみる
 - <https://www.peko-step.com/tool/hsvrgb.html>
- 実際にカメラと画像を用いて抽出を行う

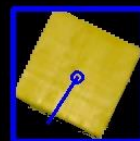
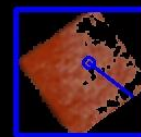
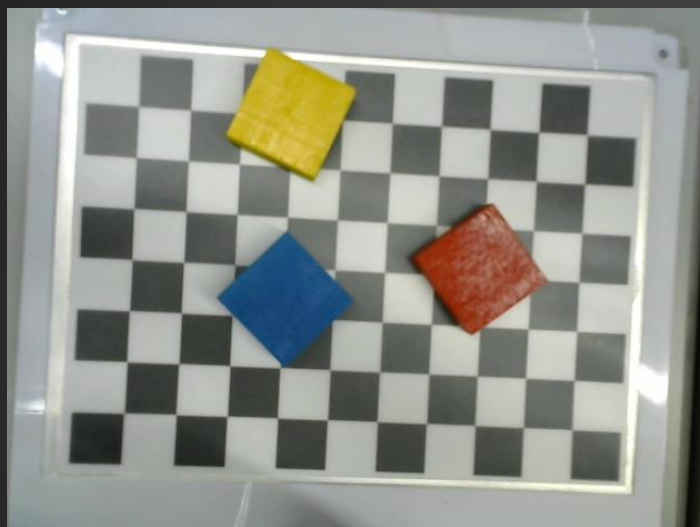
色抽出の結果

- 色を抽出するとその重心と向きを算出した画像が得られる



色抽出の結果

- うまく抽出できると下記の画像のようになる



カメラとロボットの協調

- ロボットの原点とカメラの原点が異なる
 - ロボットは土台中央, 画像は左上
 - 場合によっては軸のとり方も逆
- カメラの視野は鉛直下向きと限らない
 - ロボット座標の軸とカメラ座標の軸の関係を記述したいが厳密に合わせるのが難しい
 - カメラを真下に向けたつもりでも, 視線ベクトルが画像の中心を貫いているとは限らない
 - そもそも合っているかどうかをどう評価するか
 - ロボットの座標系をカメラ座標系に変換する行列を決定する
 - 同次変換行列による回転・並進行列を用いてカメラ座標系をロボット座標系に変換

ワークが置かれる平面の推定

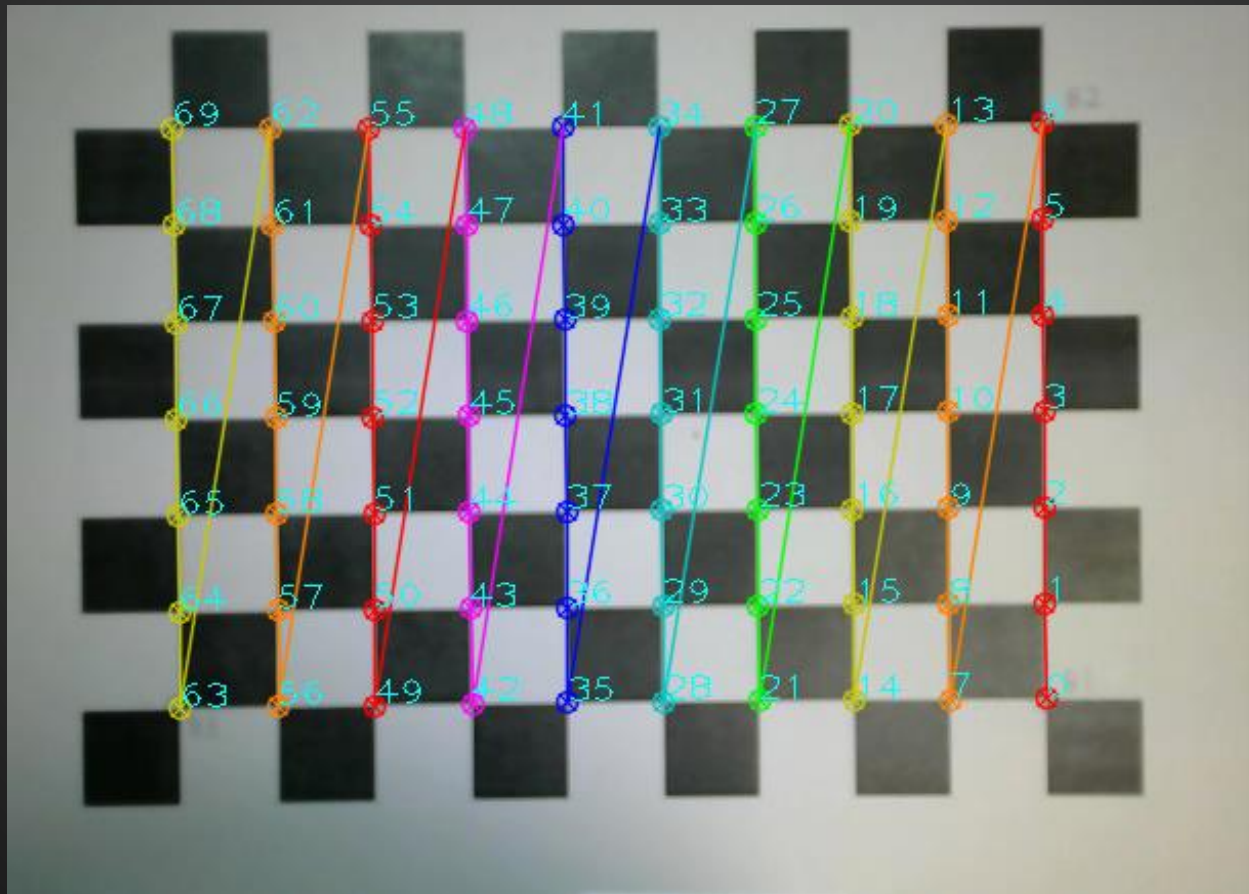
- 平面上3点で決定できる
 - 平面上に存在する3点決めるとはじめて一意に定義できる
- ワークを置く平面と画像平面の傾きを補正する行列を得ることで変換可能となる

手順

1. カメラとロボットを固定する
 - 位置関係が変わらないようにテープなどで固定する
2. ロボットの作業領域にキャリブレーションプレート（チェッカーボード）を固定する
3. カメラでキャリブレーションプレートを撮影する
 - 「(2) カメラからチェッカーボードを検出するプログラム」を参照
4. ロボットの手先でキャリブレーションプレート状の3点にティーチングを行い、その座標の値を記録する
 - 「(4) 画像処理（色抽出）およびロボットの制御を行うプログラム」を参照

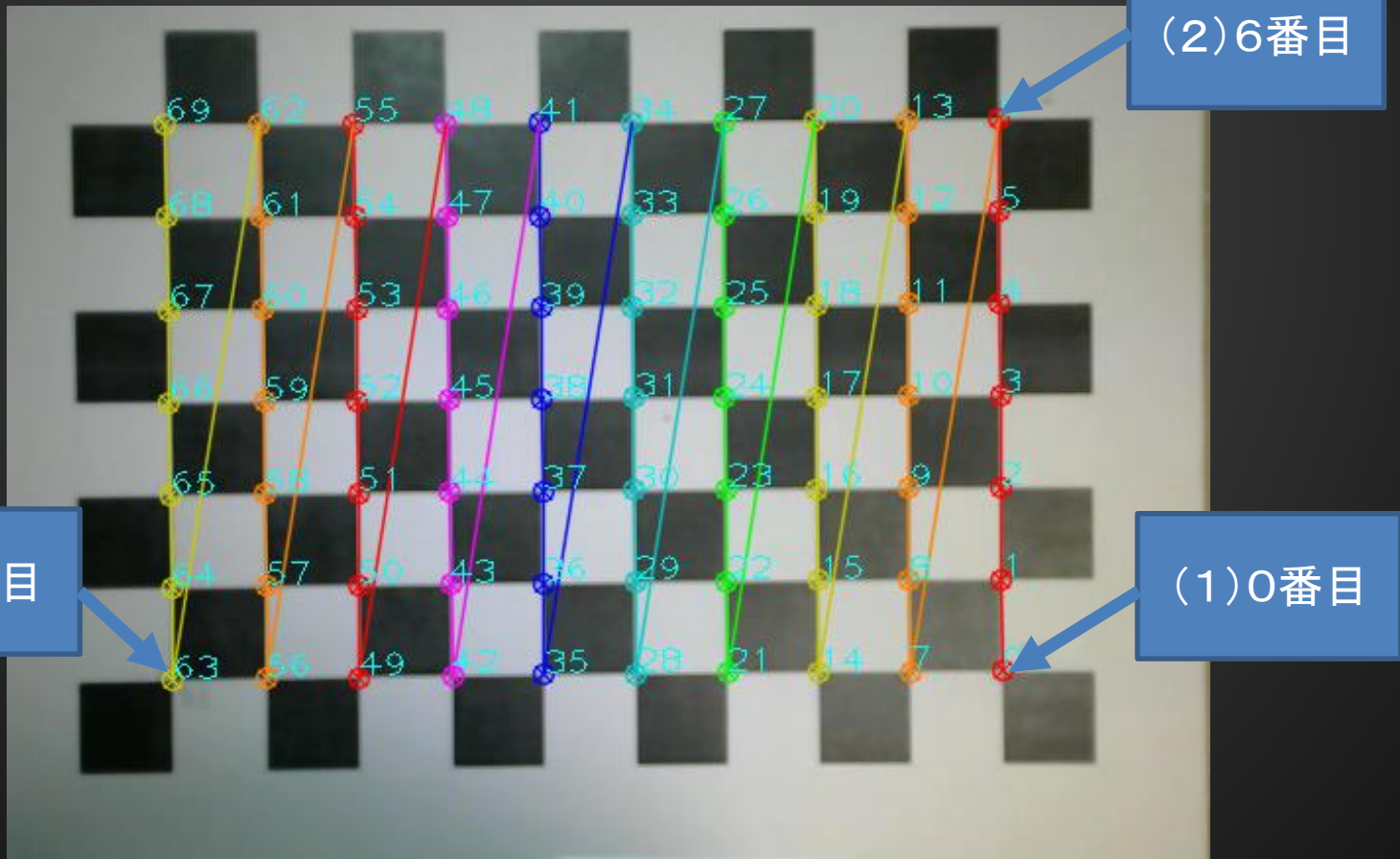
撮像した画像のチェッカー交点抽出

- 右下の点から左上にインデックスのついた点群が得られる



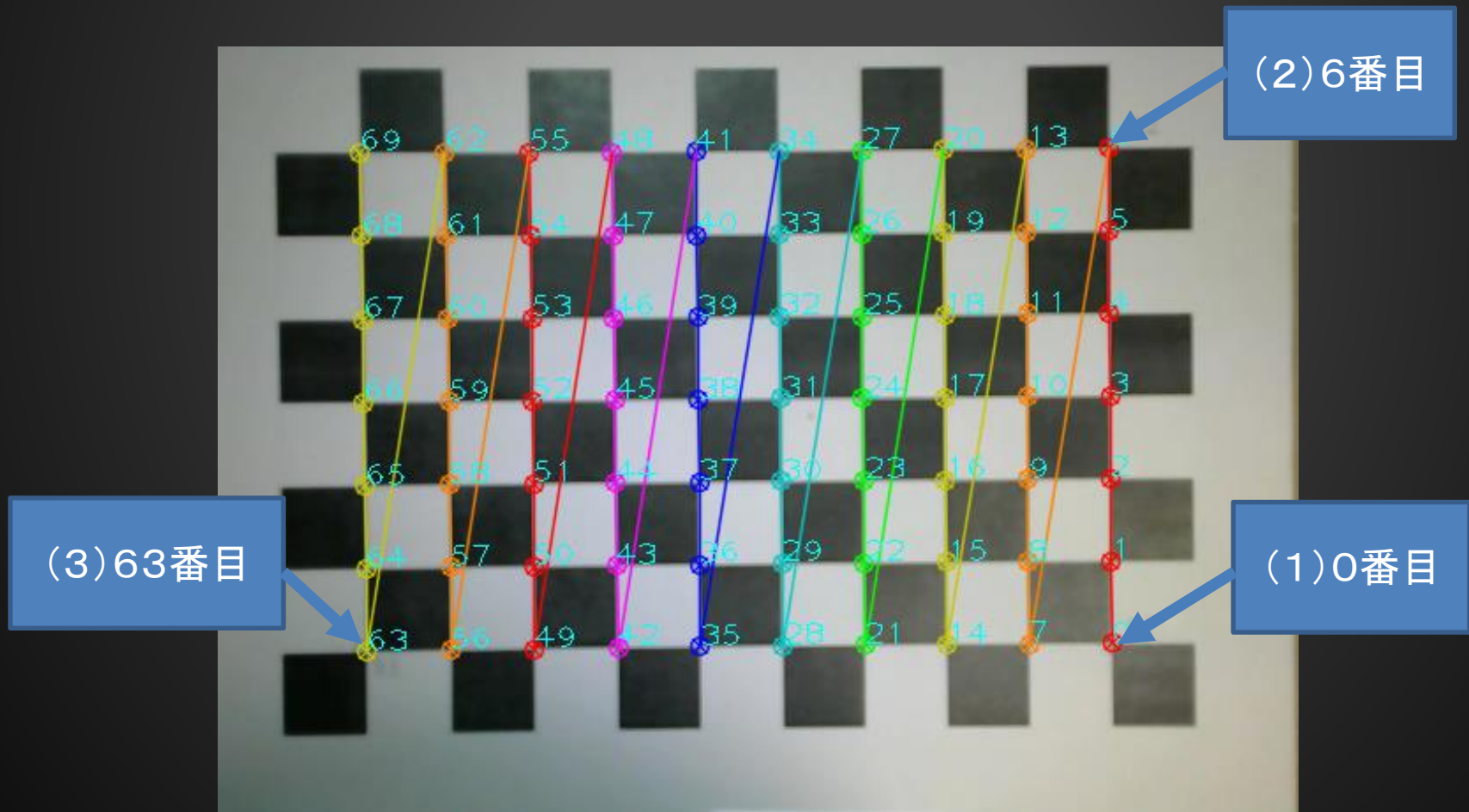
ロボットのティーチングによる 手先座標の取得

- 下記の3点に対しロボットをティーチングして手先を移動し, ロボットの手先座標 (XYZ) を記録する
 - teachPoints.csvにカンマ区切りで3点を記入する



ロボットのティーチングによる 手先座標の取得

- 次のスライドで述べるTCP/IP制御プログラム，または前回の実習で使ったシミュレータを用いる



ロボットのティーチングによる 手先座標の取得

- カメラ座標上の撮影した**3点**のチェッカーの位置にロボットのハンドを実際に動かすことでロボット座標との対応をとる
 - 2点だと1軸しか一意に決まらない。
 - 3点だと2軸が一意に決まる。2軸で外積とると1軸が決まる。
- 対応をとったカメラとロボットの座標を特異値分解することにより、カメラ座標上の点とロボット座標上の点で以下の関係が成り立つようにする

$$\begin{bmatrix} X_r \\ Y_r \\ Z_r \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

ロボット座標系 回転行列 カメラ座標系 並進行列

→ 特異値分解により回転行列と並進行列を決める

TCP/IP通信によるロボットの制御

- 添付のアプリケーションを用いてTCP/IP通信によるロボットの制御が可能であるので併せて確認する
 - ロボットをUSB接続してからVS-ASR_Console.exeを起動する
 - アラートが表示されたら許可する
 - コマンド体系については添付資料を参照
 - 下記のソースコードにより検証可能

```
# -*- coding:utf-8 -*-
```

```
import socket
```

```
host = "127.0.0.1"
```

```
port = 5000
```

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
client.connect((host, port)) #これでサーバーに接続します
```

```
client.send(b"J,900,0,0,0,0,300¥n") #コマンドを送信する
```

```
response = client.recv(4096) #適当な2の累乗にします(大きすぎるとダメ)
```

```
print(response)
```

課題（３）

- TCP/IPを用いたロボット制御アプリケーションの動作確認を行う
 - コマンド体系については同梱の資料を参照
- カメラで撮影した色抽出により把持対象物を画像から抽出する
- 把持対象物を把持して所定の場所に移動する
 - 色によるワークの選り分け

課題（３）の手順

- TCP制御サーバの起動
 - コンソールが表示され，下記のメッセージが表示されれば制御サーバが正常起動済

```
# -*- coding:utf-8 -*-
import socket

host = "127.0.0.1"
port = 5000
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((host, port)) #これでサーバーに接続します
client.send(b"J,900,0,0,0,0,300¥n")
response = client.recv(4096) #適当な2の累乗にします(大きすぎるとダメ)

print(response)
```

課題（３）の手順

- TCP制御サーバの起動
 - コンソールが表示され，下記のメッセージが表示されれば制御サーバが正常起動済

```
1個のデバイスが見つかりました
1番目のデバイスに接続中
1      -898      252      126
2      0         0       0
3      -13       255     243
4      2         0       2
5      -6        255     250
-----
server start
Listenを開始しました(0.0.0.0:5000)。
```

課題（３）の手順

- カメラから画像を取得する
 - Jupiter labでカメラから画像を取得するコードを入力する

```
%matplotlib inline
import cv2 # opencvのインポート
import matplotlib.pyplot as plt # matplotlib(描画用)
import numpy as np
from IPython.display import display, Image

def display_cv_image(image, format='.png'):
    decoded_bytes = cv2.imencode(format, image)[1].tobytes()
    display(Image(data=decoded_bytes))

def capture_camera(camid=0, mirror=True, size=None):
    """Capture video from camera"""
    # カメラをキャプチャする
    cap = cv2.VideoCapture(camid) # 0はカメラのデバイス番号
    frameldx = 0
    while True:
        # retは画像を取得成功フラグ
        ret, frame = cap.read()

        # 鏡のように映るか否か
        if mirror is True:
            frame = frame[:,::-1]

        # フレームをリサイズ
        # sizeは例えば(800, 600)
        if size is not None and len(size) == 2:
            frame = cv2.resize(frame, size)
```

課題（３）の手順

- カメラから画像を取得する
 - カメラ画像から色抽出を行うコードを入力する

```
# -*- coding:utf-8 -*-
import cv2
from time import sleep
import numpy as np
import math
import socket
from IPython.display import display, Image

def display_cv_image(image, format='.png'):
    decoded_bytes = cv2.imencode(format, image)[1].tobytes()
    display(Image(data=decoded_bytes))
def capture_camera(camid=0, mirror=True, size=None):
    """Capture video from camera"""
    # カメラをキャプチャする
    cap = cv2.VideoCapture(camid) # 0はカメラのデバイス番号
    frameldx = 0
    while True:
        # retは画像を取得成功フラグ
        ret, frame = cap.read()

        # 鏡のように映るか否か
        if mirror is True:
            frame = frame[:,::-1]

        # フレームをリサイズ
        # sizeは例えば(800, 600)
        if size is not None and len(size) == 2:
            frame = cv2.resize(frame, size)

        # フレームを表示する
```

課題（３）の手順

- カメラから画像を取得する
 - カメラーロボット座標の変換行列同定用のチェッカーボード座標を抽出する

```
%matplotlib inline
import cv2 # opencvのインポート
import matplotlib.pyplot as plt # matplotlib(描画用)
import numpy as np
from IPython.display import display, Image
import math

def display_cv_image(image, format='.png'):
    decoded_bytes = cv2.imencode(format, image)[1].tobytes()
    display(Image(data=decoded_bytes))

def detectChecker(fname):
    # termination criteria
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....., (6,5,0)
    objp = np.zeros((6*7,3), np.float32)
    objp[:, :2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

    # Arrays to store object points and image points from all the images.
    objpoints = [] # 3d point in real world space
    imgpoints = [] # 2d points in image plane.
    square_size = 1.0 # 正方形のサイズ
    pattern_size = (7, 10) # 模様のサイズ
    img = cv2.imread(fname)
    img_w_checker = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

ロボット実習のまとめ

- ティーチング作業を実習した
 - ロボット単体が行える作業はこの程度
 - 付帯装置がないと有用な作業はできない
- 画像処理
 - 現場でも役に立つ頑強な色抽出方法について実習した
 - RGBよりもHSV表色系のほうが色抽出は容易
 - ただし、照明環境により色の見えかたが異なるため、照明環境の整備は画像処理の適用においては大きな問題となる
- カメラを視覚としたロボットの活用
 - ロボットとセンサ類を組み合わせることではじめて自動機として役に立つ
 - SIerとしてロボットシステムを構築する場合…
 - ごく上流では実装の必要はないが、実装の中でも特にアルゴリズムに関する知識が必要。
 - どのように何を利用して実現するか。誰が／どこができるのか
 - 結局実装を知らないと工程引けない→値付けもできない
 - 一気通貫で立ち上げ、操業を行う場合には幅広い知識が必要
 - 自社で得意な分野はきちんとこなす。できないことは外にお願いするの大事
 - できる人を捕まえる能力やネットワークも大事