**Содержание**

[ВВЕДЕНИЕ 2](#_Toc134824222)

[Разработка программы для управления роботом 3](#_Toc134824223)

[Разработка программы для определения объекта на изображении 4](#_Toc134824224)

[Создание конфигурации для оптимизации определения объекта 5](#_Toc134824225)

[Определение координат объекта относительно робота 6](#_Toc134824226)

[Разработка интерфейса для передачи координат роботу 7](#_Toc134824227)

[Приложение 1 8](#_Toc134824228)

[Приложение 2 11](#_Toc134824229)

[Приложение 3 13](#_Toc134824230)

# ВВЕДЕНИЕ

**Цель:** разработка системы для автоматизации захвата объекта с использованием компьютерного зрения и манипулятора ABB120.

**Задачи:**

1. Разработка программы для управления роботом
2. Разработка программы для определения объекта на изображении
3. Создание конфигурации для оптимизации определения объекта
4. Определение координат объекта относительно робота
5. Разработка интерфейса для передачи координат роботу

# Разработка программы для управления роботом

Был проведен анализ готовых решений, и было выяснено жизнеспособность одного из проектов.

Документация к приложению была написано на польском, но сам код был легко читаем, поэтому разобраться как он устроен не составило труда. Приложение было не доделано и выкидывало критические ошибки, поэтому около 2 недель, мы (дебажели) отлаживали программу, в итоге она заработала и теперь ее можно использовать для управления роботом.

Основные элементы интерфейса:

|  |
| --- |
|  |
| Рисунок 1 - Табло информирующее о манипуляторах |

|  |
| --- |
|  |
| Рисунок 2 - Окно консоли |

|  |
| --- |
|  |
| Рисунок 3 - Параметры для передвижения относительно начальной точки отсчета |

|  |
| --- |
|  |
| Рисунок 4 - Параметры для передвижения относительно текущей точки отсчета |

Алгоритм использования программы:

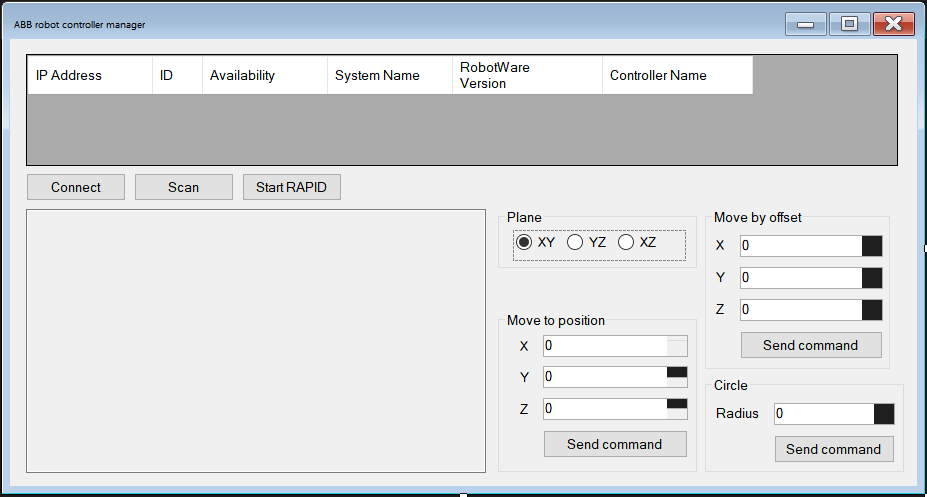
1. Нажать на кнопку Scan
2. Выбрать нужный манипулятор
3. Нажать на кнопку Connect

\*Не забудьте включить автоматический режим и включить двигатели\*

4) Нажать кнопку Start Rapid

5) Можно вводить координаты и перемешаться

ПРИЛОЖЕНИЕ 1



# Разработка программы для определения объекта на изображении

Для определения объекта была разработана программа, которая может находить контуры используя hsv фильтр и определять центры объектов. В данной программе использовались библиотеки numpy, cv2, matplotlib и т.д.

В программу загружается одно или несколько фото, у которых изменяется цветовая модель и в последствии применяется определенный фильтр. Далее определяются всевозможные контуры на обработанном изображении. После этого из этих контуров строятся возможные прямоугольники, которые находятся в рабочем поле и соответствуют определенному размеру, площади и т.д.

Следующим шагом определяются координаты центров оставшихся контуров и выводятся на графики при ручном режиме или записываются в файл для автоматизации процесса и дальнейшего использования данных в программе для робота.

Готовая программа показана в приложении 3. А итоговые результаты обработки изображения показаны на рис. 3.

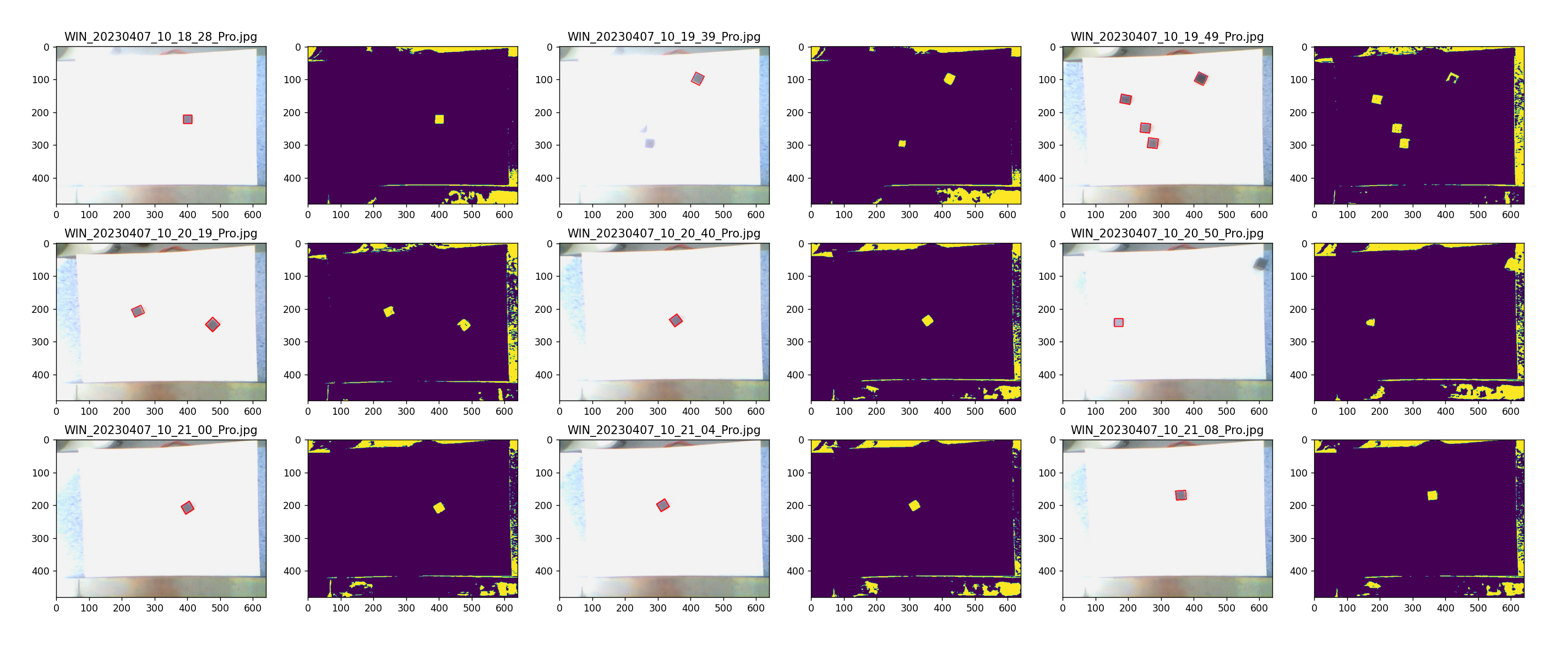


Рис. 3 – Вывод обработанных изображений

# Создание конфигурации для оптимизации определения объекта

Из-за большого количества входных данным и возможных отличий изображений друг от друга становится тяжело вручную подобрать подходящие коэффициенты для hsv фильтра. Поэтому для оптимизации определения объекта было решено создать отдельную программу в который можно будет определить самые подходящие значения коэффициентов hsv фильтра при разных внешних воздействиях (освещение, фон объекта и т.д.). Для реализации данной задачи был выбран метод бинаризации изображения, с дальнейшим ручным подбором параметров по визуальному результату.

Метод бинаризации – это перевод цветного (или в градациях серого) изображения в двухцветное черно-белое. Главным параметром такого преобразования является порог t – значение, с которым сравнивается яркость каждого пикселя. По результатам сравнения, пикселю присваивается значение 0 или 1.

В итоге мы имеем программу (приложение 2), которая создает два окна на рабочем столе компьютера. В одном окне можно регулировать коэффициенты hsv фильтра, а в другом сразу фидель результат изменений. Данные действия показаны на рисунке 1.

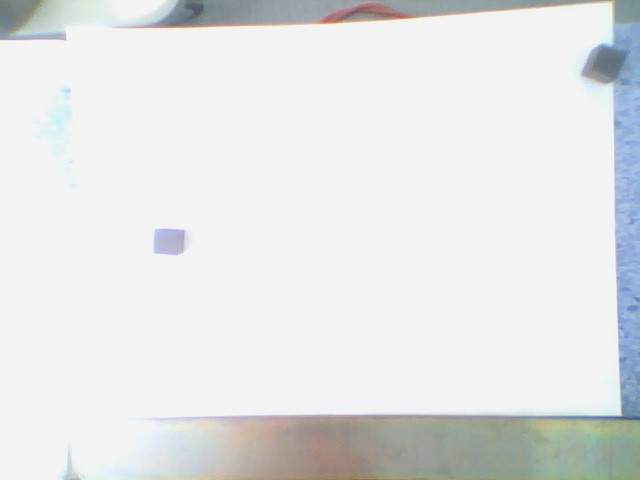
 

Рис. 1 – Подбор коэффициентов по изображению

# Определение координат объекта относительно робота

Следующей задачей было сопоставить полученные координаты объекта в пикселях с реальными координатами по которым перемещается робот. Для этого была создана позиция, в которую перемещается робот для создания фотографии. Данная позиция представлена на листинге ниже.

!! Позиция для создания изображений

CONST robtarget Alt230:=[[432.29,19.76,512.94],[0.00142583,-0.00444574,-0.999692,0.0243684],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

Зная данную позицию и используя полученные изображения, удалось найти соотношение которые могло сопоставить пиксели и миллиметры. Полученные значения прибавлялись к уже известной точке нахождения схвата, и робот перемещался к ней. Зная заранее координаты схвата, оставалось только изменять высоту, чтобы захватить объект.

Исходная система, с которой велась данная работа показана на рис. 2.

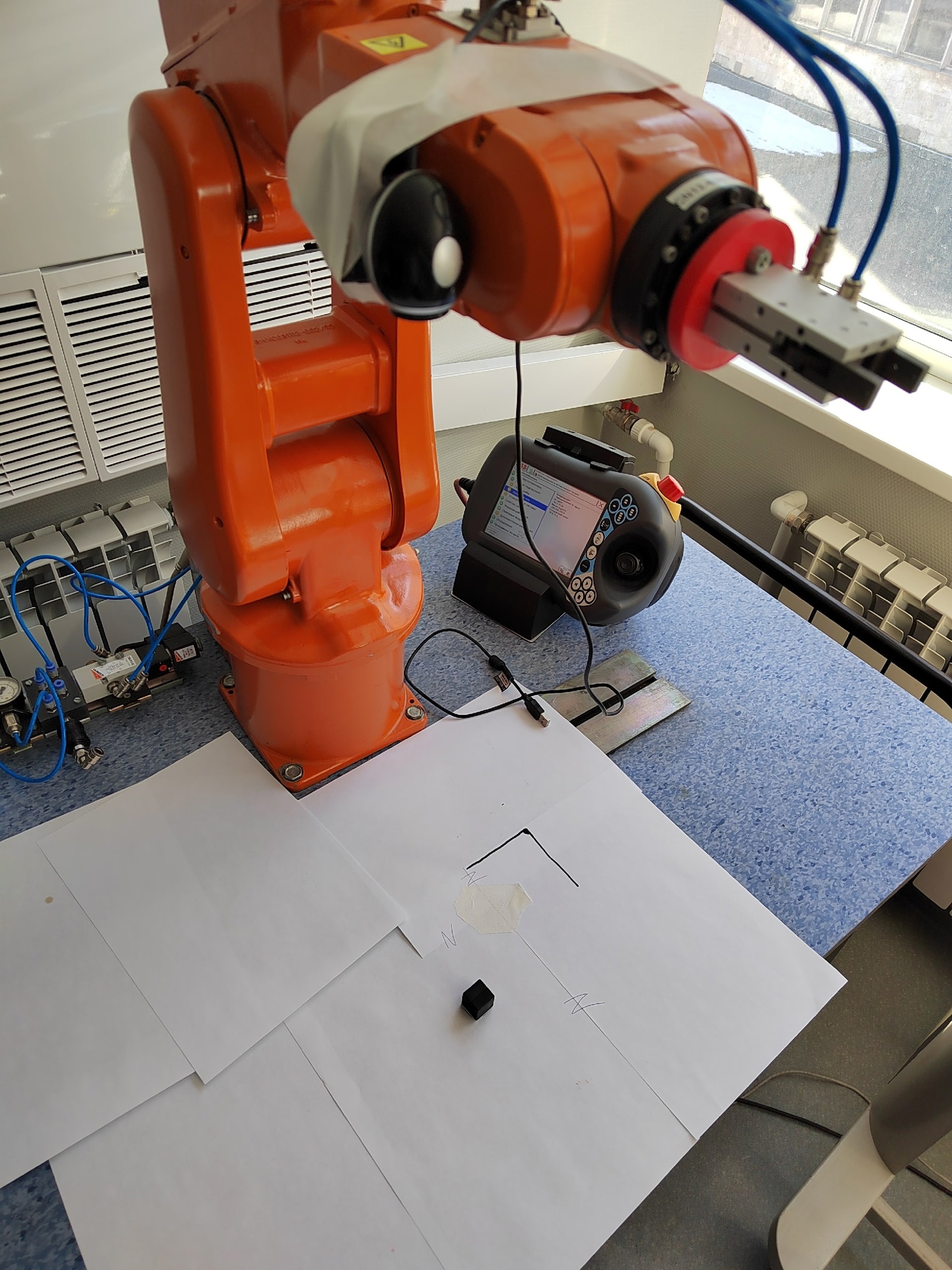


Рис. 2 – Положение робота и объекта на стенде

# Разработка интерфейса для передачи координат роботу

Для того чтобы связать код написанный на питоне и польскую программу, мы решили использовать пользовательский интерфейс для взаимодействия. То есть, мы подготовили скрипт который симулирует поведение пользователя и вводит данные в нужные ячейки и нажимает кнопки. Такой уровень взаимодействия в обычном случае не приветствуется, но с точки зрения эксперимента сойдет.

Передача состоит в том, чтобы обработать картинку с камеры, определить где находится объект на изображении и сопоставив с осью координат манипулятора отправить координаты в польскую программу, чтобы дать команду на передвижение робота в это место и произвести захват объекта.

# Заключение

Хочется отметить, что результат полученной работы является выше среднего, так как изначально проект служил целью развития в человеке способностей к взаимодействию с другими членами команды, изучение компьютерного зрения и работой с системой RobotStudio.

Не все задачи были выполнены, так как внутренняя технология робота ABB120, который расположен на кафедре в аудитории А307, несовершенна. В нем отсутствует способ сохранения последней координаты, где остановился робот в прошлый сеанс работы. Способ сохранения координат на жесткий диск робота не рассматривался из-за отсутствия безопасности эксплуатации программного кода без сотрудника кафедры.

# Приложение 1

private void Button\_Connection\_Click(object sender, EventArgs e)

{

if ((bool)(Button\_Connection.Tag ?? false))

{

\_controlUnit?.Dispose();

UpdateButtons(false);

}

else

{

InitControlUnit();

UpdateButtons(\_controlUnit.TryConnect());

}

}

private void Button\_Scan\_Click(object sender, EventArgs e) => ReloadControllers();

private void Button\_Rapid\_Click(object sender, EventArgs e)

{

if ((bool)(Button\_Rapid.Tag ?? false))

{

\_controlUnit?.TryStopRapidProgram();

}

else

{

\_controlUnit?.TryStartRapidProgram();

}

}

private void UpdateRapidButton(bool isRunning)

{

Invoke((MethodInvoker)delegate

{

Button\_Rapid.Tag = isRunning;

if (isRunning)

{

Button\_Rapid.Text = "Stop RAPID";

}

else

{

Button\_Rapid.Text = "Start RAPID";

}

});

}

private void Button\_MoveByOffset\_Click(object sender, EventArgs e)

{

\_controlUnit?.OffsetMove((double)NumericBox\_OffsetX.Value, (double)NumericBox\_OffsetY.Value, (double)NumericBox\_OffsetZ.Value);

}

private void Button\_MoveToPosition\_Click(object sender, EventArgs e)

{

\_controlUnit.OnMessageCall(((double)NumericBox\_PositionX.Value).ToString());

\_controlUnit?.Move((double)NumericBox\_PositionX.Value, (double)NumericBox\_PositionY.Value, (double)NumericBox\_PositionZ.Value);

}

public bool Move(double x, double y, double z, bool doLog = true)

{

return MoveDetailed(x, y, z, doLog).result;

}

private (bool result, double x, double y, double z) MoveDetailed(double x, double y, double z, bool doLog = true)

{

OnMessageCall("HERE");

if (!AwaitCompletion())

{

return (false, 0, 0, 0);

}

var task = \_controller.Rapid.GetTask(RapidNames.TaskName);

var homeFlag = task.GetRapidData(RapidNames.ModuleName, RapidNariables.HomeFlag); ;

using (var m = Mastership.Request(\_controller.Rapid))

{

homeFlag.Value = new Bool(true);

}

var executeFlag = task.GetRapidData(RapidNames.ModuleName, RapidNariables.ExecuteFlag);

var xData = task.GetRapidData(RapidNames.ModuleName, RapidNariables.XCoord);

OnMessageCall(xData.ToString());

homeFlag = task.GetRapidData(RapidNames.ModuleName, RapidNariables.HomeFlag);

var yData = task.GetRapidData(RapidNames.ModuleName, RapidNariables.YCoord);

var zData = task.GetRapidData(RapidNames.ModuleName, RapidNariables.ZCoord);

try

{

if (\_controller.OperatingMode == ControllerOperatingMode.Auto)

{

using (var m = Mastership.Request(\_controller.Rapid))

{

xData.Value = new Num(x);

yData.Value = new Num(y); zData.Value = new Num(z);

executeFlag.Value = new Bool(true);

homeFlag.Value = new Bool(true);

}

if (doLog)

{

OnMessageCall(

$"Moving to position: " +

$"{x.GetFixedString(3, 2)}; " +

$"{y.GetFixedString(3, 2)}; " +

$"{z.GetFixedString(3, 2)}");

}

var position = GetPosition();

return (AwaitCompletion(), position.x, position.y, position.z);

}

else

{

OnMessageCall("Failed to move to target position: Automatic mode red to start execution from a remote client.");

}

}

catch (InvalidOperationException)

{

OnMessageCall("Failed to move to target position: Mastership is held er client.");

}

catch (Exception ex)

{

OnMessageCall("Failed to move to target position: " + ex.Message);

}

return (false, 0, 0, 0);

}

private bool AwaitCompletion()

{

RapidTask task;

RapidData executeFlag;

do

{

if (HasTimedOut())

{

StopTimeout();

OnMessageCall($"Operation has timed out.");

return false;

}

task = \_controller.Rapid.GetTask(RapidNames.TaskName);

executeFlag = task.GetRapidData(RapidNames.ModuleName, es.Variables.ExecuteFlag);

using (var m = Mastership.Request(\_controller.Rapid))

{

executeFlag.Value = new Bool(true);

}

OnMessageCall(executeFlag.GetBool().ToString());

Thread.Sleep(1);

} while (!executeFlag.GetBool());

StopTimeout();

return true;

}

# Приложение 2

import cv2

import numpy as np

if \_\_name\_\_ == '\_\_main\_\_':

    def nothing(\*arg):

        pass

cv2.namedWindow("result", cv2.WINDOW\_NORMAL) # создаем главное окно

cv2.namedWindow( "settings" ) # создаем окно настроек

# создаем 6 бегунков для настройки начального и конечного цвета фильтра

cv2.createTrackbar('h1', 'settings', 0, 255, nothing)

cv2.createTrackbar('s1', 'settings', 0, 255, nothing)

cv2.createTrackbar('v1', 'settings', 0, 255, nothing)

cv2.createTrackbar('h2', 'settings', 255, 255, nothing)

cv2.createTrackbar('s2', 'settings', 255, 255, nothing)

cv2.createTrackbar('v2', 'settings', 255, 255, nothing)

crange = [0,0,0, 0,0,0]

while True:

    fn = 'WIN\_20230407\_10\_20\_50\_Pro.jpg' # имя файла, который будем анализировать

    img = cv2.imread(fn)

    hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV )

    # считываем значения бегунков

    h1 = cv2.getTrackbarPos('h1', 'settings')

    s1 = cv2.getTrackbarPos('s1', 'settings')

    v1 = cv2.getTrackbarPos('v1', 'settings')

    h2 = cv2.getTrackbarPos('h2', 'settings')

    s2 = cv2.getTrackbarPos('s2', 'settings')

    v2 = cv2.getTrackbarPos('v2', 'settings')

    # формируем начальный и конечный цвет фильтра

    h\_min = np.array((h1, s1, v1), np.uint8)

    h\_max = np.array((h2, s2, v2), np.uint8)

    # накладываем фильтр на кадр в модели HSV

    thresh = cv2.inRange(hsv, h\_min, h\_max)

    cv2.imshow('result', thresh)

    ch = cv2.waitKey(5)

    if ch == 27:

        break

cv2.destroyAllWindows()

# Приложение 3

import sys

import glob

import numpy as np

import cv2 as cv

import matplotlib.pyplot as plt

# матрица для вывода нескольких рисунков

fig, axs = plt.subplots(3, 6, figsize=(9, 3))

strok = 0

stolb = 0

# Импорт одного определенного файла

# file = 'WIN\_20230407\_10\_19\_28\_Pro.jpg' # имя файла, который будем анализировать

# img = cv.imread(file)

for file in glob.glob("D:/University/Maks/ABB120PythonScript/\*.jpg"):

    img = cv.imread(file)

    # hsv - Hue, Saturation, Value — тон, насыщенность, значение

    # меняем цветовую модель с BGR на HSV

    hsv = cv.cvtColor(img, cv.COLOR\_BGR2HSV)        # меняем цветовую модель с BGR на HSV

    # Создание маски на изображение, чтобы исключить ложные объекты

    hsv\_min = np.array((55, 5, 100), np.uint8)        # нижняя граница

    hsv\_max = np.array((167, 90, 220), np.uint8)    # верхняя граница

    mask = cv.inRange(hsv, hsv\_min, hsv\_max)        # применяем цветовой фильтр

    # меняем цветовую модель с BGR на RGB для привычного отображения

    img = cv.cvtColor(img, cv.COLOR\_BGR2RGB)

    # ищем все контуры

    contours, hierarchy = cv.findContours(mask.copy(), cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE)

    # перебираем все найденные контуры в цикле и ищем прямоугольники

    for cnt in contours:

        # нахождение прямоугольника минимальной площади

        # на выход - ( center (x,y), (width, height), angle of rotation )

        rect = cv.minAreaRect(cnt)

        # проверка на нахождение прямоугольника в рабочем поле

        if rect[0][0] < 620 and rect[0][0] > 70 and rect[0][1] < 420 and rect[0][0] > 20: # можно положить бумагу на все поле видимости камеры чтобы исключить фон

            # проверка на длины сторон (чтобы был только квадрат)

            if 0.8\*rect[1][1] < rect[1][0] < 1.2\*rect[1][1]:

                box = cv.boxPoints(rect)            # поиск четырех вершин прямоугольника

                box = np.int0(box)                  # округление координат вершин

                area = int(rect[1][0]\*rect[1][1])   # вычисление площади

                # отфильтровываем прямоугольники по площади

                if 1000 > area > 500:

                    cv.drawContours(img,[box],0,(255,0,0),2)

                    # Определение координат центра кубика

                    X = int(rect[0][0])

                    Y = int(rect[0][1])

                    print(box)                      # вывод массива координат углов

                    print("Square:", area)          # вывод площади

                    print("Center of Cube:", X, Y)  # вывод центра

                    print("Angle:", int(rect[2]))   # вывод угла поворота кубика

                    print("")

    # вывод графиков матрицей

    if stolb == 6:

        strok += 1

        stolb = 0

    elif stolb < 6:

        axs[strok,stolb].imshow(img)

        axs[strok,stolb].set\_title(str(file[38:]))

        stolb += 1

        axs[strok,stolb].imshow(mask)

        stolb += 1

# запись координат центра кубика в файл

# with open('example.txt', 'w') as f:

#     f.write("X\_center: " + str(X) + " " + "Y\_center: " + str(Y))

plt.show()