SDA3 End Assignment

**Group:** A1\_WinnenVanB2IsBelangrijkerDanMeedoen

**Class:** M3A

**Team members:** Irven van Daal

Siem van Cranenbroek

Niels Konings

Teun Jacobs

**Teacher:** Henk van de Schoot

**GitHub Repository:** <https://github.com/WinnenVanB2IsBelangrijkerDanMeedoen/EndAssigmentMain>

Contents

[**User Story** 2](#_Toc149726588)

[**Class diagram** 2](#_Toc149726589)

[**Sequence Diagram** 3](#_Toc149726590)

[**State Machine Diagram** 4](#_Toc149726591)

[**Demonstratie video** 5](#_Toc149726592)

[**Python code** 5](#_Toc149726593)

[RoboticMain.py 5](#_Toc149726594)

[RobotArm.py 6](#_Toc149726595)

[LoadingRobot.py 7](#_Toc149726596)

[ResizeFrame 9](#_Toc149726597)

[ObjectDetection.py 10](#_Toc149726598)

[ColorRecognition.py 12](#_Toc149726599)

[CoördinateCalculation 12](#_Toc149726600)

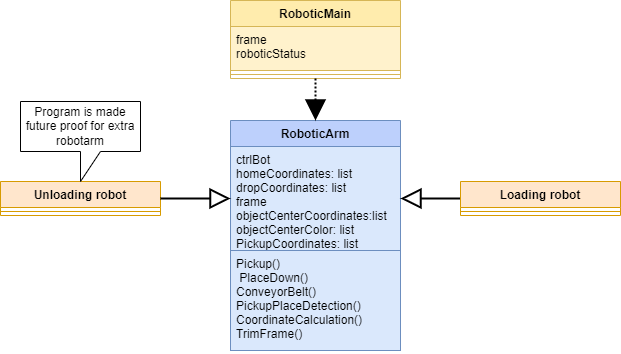
[UserColorSelector.py 13](#_Toc149726601)



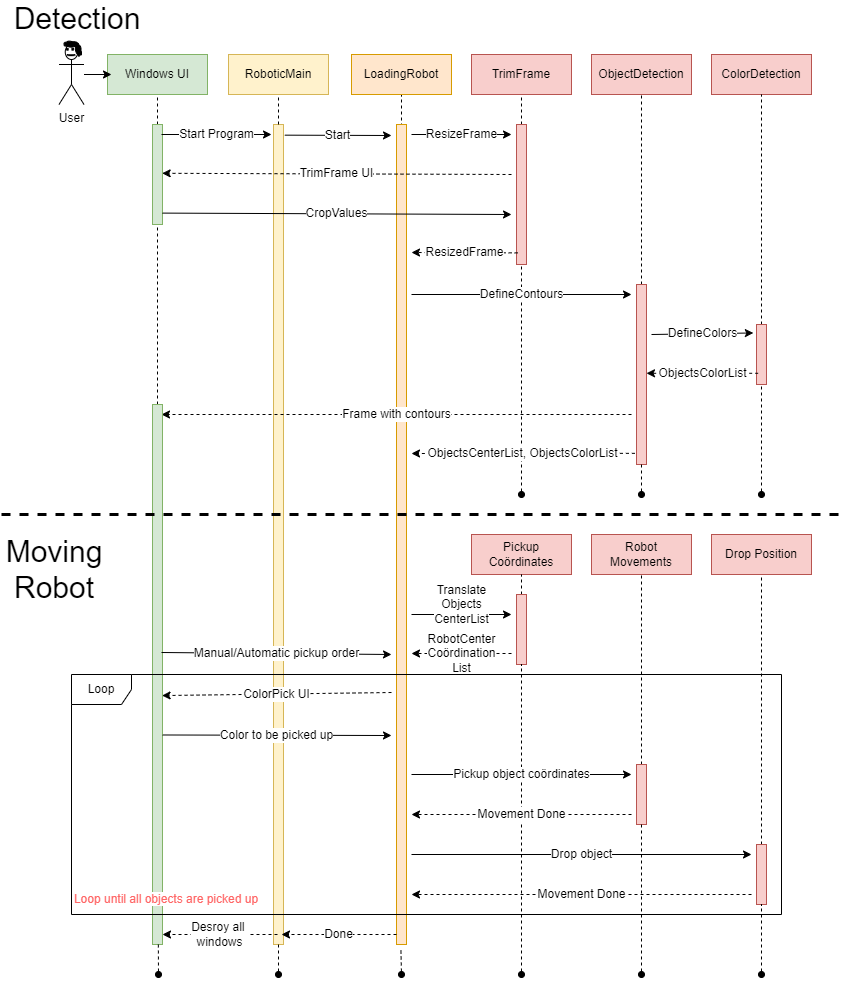
# **User Story**

|  |  |  |
| --- | --- | --- |
| **Title:** End Assignment SDA3 | **Priority:** 1 | **Estimate:** 3 weeks |
| **As a:** operator  **I want to:** automate loading and unloading objects from a conveyor belt  **So that I can:** automatically sort objects by colour | | |
| **Acceptance criteria:**   1. Only objects with a red colour get picked up when the red button is pressed on the UI 2. Only objects with a blue colour get picked up when the blue button is pressed on the UI 3. Only objects with a yellow colour get picked up when the yellow button is pressed on the UI 4. Only objects with a green colour get picked up when the green button is pressed on the UI 5. Objects with other colours than mentioned above will not be picked up | | |

# **Class diagram**



# **Sequence Diagram**

Below are the two sequence diagrams for the program are shown. The first for detecting the objects and the second for controlling the robot. Because the second one starts directly after the first one, some functions from the first diagram are extended to the second diagram because they carry required data and/or instructions.

# **State Machine Diagram**

A group of ovals with text

Description automatically generated

# **Demonstratie video**

<https://youtu.be/CpRquNOuQ80>

(File was to big for GitHub)

# **Python code**

## RoboticMain.py

from LoadingRobot import \*

import time

import os

import keyboard

import threading

from DoBotArm import DoBotArm as Dbt

from serial.tools import list\_ports

from abc import ABC, abstractmethod

from ObjectDetection import \*

import tkinter as tk

def main():

    os.chdir(path= 'SDA3\_final\_assignment\_sol-main/code/DoBotArm')

    roboticState = 'initialize'

    userCoordinates = (145,215)

    while True:

        match roboticState:

            case 'initialize':

                port = portSelection()

                homeCoordinatesLoadingRobot = (5, 210, 60)

                ctrlBot = Dbt.DoBotArm(port, homeCoordinatesLoadingRobot[0], homeCoordinatesLoadingRobot[1], homeCoordinatesLoadingRobot[2], home = True) #Create DoBot Class Object with home position x,y,z

                loadingBot = LoadingRobot(ctrlBot)

                ctrlBot.moveArmRelXY(0, 0, wait=True) #deze slaat hij voor een of andere manier over

                loadingBot.TrimFrame()

                loadingBot.PickupPlaceDetection()

                loadingBot.CoordinateCalculation()

                roboticState = 'userColorChoice'

            case 'userColorChoice':

                userCoordinates = loadingBot.UserChoice()

                roboticState = 'pickUp'

            case 'pickUp':

                loadingBot.PickUp(userCoordinates)

                roboticState = 'placeDown'

            case 'placeDown':

                loadingBot.PlaceDown()

                roboticState = 'retakePhoto'

            case 'retakePhoto':

                loadingBot.RetakePhoto()

                loadingBot.PickupPlaceDetection()

                loadingBot.CoordinateCalculation()

                roboticState = 'userColorChoice'

        if keyboard.is\_pressed("esc"):

            ctrlBot.SetConveyor(enabled=False)

            break

if \_\_name\_\_ == "\_\_main\_\_":

    main()

## RobotArm.py

import threading

from DoBotArm import DoBotArm as Dbt

from serial.tools import list\_ports

from abc import ABC, abstractmethod

from ObjectDetection import \*

def portSelection():

    #Choosing port

    availablePorts = list\_ports.comports()

    print('Available COM-ports0:')

    for i, port in enumerate(availablePorts):

        print(f"  {i}: {port.description}")

    choice = 0 #int(input('Choose port by typing a number followed by [Enter]: ')) use this if you want to manualy select the port

    return availablePorts[choice].device

def homingPrompt():

    while (True):

        response = input("Do you wanna home? (y/n)")

        if(response == "y") :

            return True

        elif (response == "n"):

            return False

        else:

            print("Unrecognised response")

class RoboticArm(ABC):

    def \_\_init\_\_(self):

        pass

    @abstractmethod

    def PickUp():

        raise NotImplementedError

    @abstractmethod

    def PlaceDown():

        raise NotImplementedError

    @abstractmethod

    def ConveyorBelt():

        raise NotImplementedError

    def PickUpPlaceDetection():

        raise NotImplementedError

    @abstractmethod

    def TrimFrame():

        raise NotImplementedError

    @abstractmethod

    def CoordinateCalculation():

        raise NotImplementedError

## LoadingRobot.py

from RoboticArm import \*

from ObjectDetection import\*

from ResizeFrame import \*

from CoordinateCalculation import CoordinateCalculation

from userColorSelector import SelectColor

import cv2

class LoadingRobot(RoboticArm):

    def \_\_init\_\_(self, ctrlBot):

        self.ctrlBot = ctrlBot

        print("innit")

    def Initialize(self):

        print("initialize")

        return self.ctrlBot

    def TrimFrame(self):

        self.resizedFrame, self.resizeValues = ResizeFrame()

    def PickUp(self, userCoordinates, defaultPickUpLocation = (145,215,60)):

        self.ctrlBot.moveArmXYZ(defaultPickUpLocation[0],defaultPickUpLocation[1],defaultPickUpLocation[2], jump = True, wait = True)

        self.ctrlBot.moveArmXY(userCoordinates[0], userCoordinates[1], wait = True)

        self.ctrlBot.SetConveyor(enabled=True)

        self.ctrlBot.pickToggle()

        self.ctrlBot.SetConveyor(enabled=False)

        self.ctrlBot.moveArmXYZ(None, None, 60, wait = False)

    def PlaceDown(self):

        self.ctrlBot.moveHome()

        self.ctrlBot.pickToggle()

    def ConveyorBelt():

        pass

    def PickupPlaceDetection(self):

        self.centerList, self.colorList, frame = ObjectDetection(self.resizedFrame)

    def RetakePhoto(self):

        vidCapture = cv2.VideoCapture(1, cv2.CAP\_DSHOW)

        \_, frame = vidCapture.read()

        self.resizedFrame = frame[self.resizeValues[0]:self.resizeValues[1], self.resizeValues[2]:self.resizeValues[3]]

    def UserChoice(self):

        userChoice = SelectColor(self.colorList)

        objectNumber = self.colorList.index(userChoice)

        return (self.RobotCoordinateCenterList[objectNumber])

    def CoordinateCalculation(self):

        self.RobotCoordinateCenterList = CoordinateCalculation(self.centerList)

## ResizeFrame

import cv2

def ResizeFrame():

    vidCapture = cv2.VideoCapture(1, cv2.CAP\_DSHOW)

    frameWidth = int(vidCapture.get(3))

    frameHeight = int(vidCapture.get(4))

    def TrackBars():

        global show

        show = True

        pass

    cv2.namedWindow("SliderWindow",cv2.WINDOW\_AUTOSIZE)

    cv2.createTrackbar("xMax", "SliderWindow", frameWidth, frameWidth, TrackBars)

    cv2.createTrackbar("xMin", "SliderWindow", 0, frameWidth, TrackBars)

    cv2.createTrackbar("yMax", "SliderWindow", frameWidth, frameHeight, TrackBars)

    cv2.createTrackbar("yMin", "SliderWindow", 0, frameHeight, TrackBars)

    while(vidCapture.isOpened()):

        ret, frame = vidCapture.read()

        if ret == True:

            xMax = cv2.getTrackbarPos("xMax", 'SliderWindow')

            xMin = cv2.getTrackbarPos("xMin", 'SliderWindow')

            yMax = cv2.getTrackbarPos("yMax", 'SliderWindow')

            yMin = cv2.getTrackbarPos("yMin", 'SliderWindow')

            if xMax <= xMin:

                xMax = xMin + 50

            if yMax <= yMin:

                yMax = yMin + 50

            frame = frame[xMin:xMax, yMin:yMax]

            cv2.imshow("ResizeFrame", frame)

            key = cv2.waitKey(20)

            if key == ord("q"):

                print(xMin,":",xMax, '\t', yMin,":",yMax)

                cv2. destroyWindow("ResizeFrame")

                cv2. destroyWindow("SliderWindow")

                break

        else:

            break

    return frame, (xMin,xMax, yMin, yMax)

## ObjectDetection.py

import cv2

import numpy as np

from ColorRecognition import \*

def ObjectDetection(frame):

    frameinfo = frame.shape

    \_ , frameWidth, \_ = frameinfo

    colorList = [

        ["Yellow",  [18, 160, 0], [39, 255, 255]],

        ["Red",     [120, 100, 0], [225, 255, 255]],

        ["Green",   [43, 79, 0], [84, 255, 255]],

        ["Blue",    [100, 90, 0], [162, 255, 255]],

    ]

    #yellow

    minHSVYellow = np.array([18, 160, 0])

    maxHSVYellow = np.array([39, 255, 255])

    #Red

    minHSVRed = np.array([0, 0, 0])

    maxHSVRed = np.array([225, 255, 255])

    #green

    minHSVGreen = np.array([43, 79, 0])

    maxHSVGreen = np.array([84, 255, 255])

    #blue

    minHSVBlue = np.array([100, 90, 0])

    maxHSVBlue = np.array([162, 255, 255])

    def MergeImage(image1, image2):

        alpha\_background = image1[:,:,2] / 255.0

        alpha\_foreground = image2[:,:,2] / 255.0

        for color in range(0, 3):

            image1[:,:,color] = alpha\_foreground \* image2[:,:,color] + \

            alpha\_background \* image1[:,:,color] \* (1 - alpha\_foreground)

        image1[:,:,2]=(1-(1-alpha\_foreground)\*(1-alpha\_background))\*255

        return image1

    #detect the collors with HSV so that the light intesenty has less of an inpact on the collor value

    imageHSV = cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV)

    #Red detection

    maskHSVRed = cv2.inRange(imageHSV, minHSVRed, maxHSVRed)

    resultHSVRed = cv2.bitwise\_and(frame, frame, mask = maskHSVRed)

    #yellow detection

    maskHSVYellow = cv2.inRange(imageHSV, minHSVYellow, maxHSVYellow)

    resultHSVYellow = cv2.bitwise\_and(frame, frame, mask = maskHSVYellow)

    #green detection

    maskHSVGreen = cv2.inRange(imageHSV, minHSVGreen, maxHSVGreen)

    resultHSVGreen = cv2.bitwise\_and(frame, frame, mask = maskHSVGreen)

    #blue detection

    maskHSVBlue = cv2.inRange(imageHSV, minHSVBlue, maxHSVBlue)

    resultHSVBlue = cv2.bitwise\_and(frame, frame, mask = maskHSVBlue)

    #merging images

    resultBlueGreen = MergeImage(resultHSVBlue, resultHSVGreen)

    resultBlueGreenRed = MergeImage(resultBlueGreen, resultHSVRed)

    resultBlueGreenRedYellow = MergeImage(resultBlueGreenRed, resultHSVYellow)

    edged = cv2.Canny(resultBlueGreenRedYellow, 30, 100) #create edges on image

    kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))

    dilate = cv2.dilate(edged, kernel, iterations=1) # zorgt ervoor dat er verbindgen zijn tussen de randen

    contoursresult, \_ = cv2.findContours(image = dilate, mode=cv2.RETR\_EXTERNAL, method=cv2.CHAIN\_APPROX\_SIMPLE)

    contourImage = frame.copy()

    #Calculate contour center

    borderMargin = 10

    centerObjectList=[]

    colorObjectList =[]

    for contour in contoursresult:

        area = cv2.contourArea(contour)

        if 5000 > area > 400: #5000 is max area for an object and 400 is minimal area for object

            if len(contour) > 0:

                x, \_ = contour[0][0]

                if borderMargin <= x < frameWidth - borderMargin: #removes the frame edge as an contour

                    moment = cv2.moments(contour)

                    #calculating centroid

                    if moment["m00"] != 0:

                        cX = int(moment["m10"]/moment["m00"]) #center X coordinate

                        cY = int(moment["m01"]/moment["m00"]) #center Y coordinate

                    else:

                        cX, cY = 0, 0

                colorName = PixelHsvColor(frame, cY, cX, colorList)

            cv2.circle(contourImage, (cX, cY), 5, (0, 255, 255), -1)

            cv2.putText(contourImage, colorName, (cX -25, cY -35), cv2.FONT\_HERSHEY\_DUPLEX, 0.4, (0, 255, 0), 1)

            centerObjectList.append((cX,cY))

            colorObjectList.append(colorName)

            cv2.drawContours(image=contourImage, contours=[contour], contourIdx=-1, color=(0, 255, 0), thickness=2, lineType=cv2.LINE\_AA)

    return centerObjectList, colorObjectList, contourImage

## ColorRecognition.py

import cv2

import numpy as np

def PixelHsvColor(frame, cY, cX, colorlist):

    hsvFrame = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

    hsvPixelValue = hsvFrame[cY,cX]

    for colorName, minimum, maximum in colorlist:

            lowerBound = np.array(minimum)

            upperBound = np.array(maximum)

            if(lowerBound <= hsvPixelValue).all() and (hsvPixelValue <= upperBound).all():

                return colorName

    return "unknown"

## CoördinateCalculation

def CoordinateCalculation(centerCoordinatesList):

    XAdjustment = -3 #correction X postion of the nose

    YAdjjustment = 12 #correctioin Y postion of the nose

    scaleFactor = 0.725 #Robot lengt of picup area divided by the camara lengt of picup area

    MaxX = 300 #maximum X postion of the pickup area

    maxY = 225 #maximum Y postion of the picup are

    centerCoordinateRoboticarm = []

    print(centerCoordinatesList)

    for centerCoordinate in centerCoordinatesList:

        X, Y = centerCoordinate

        #transfrom the camare coordinate to the robot coordinates

        X \*= scaleFactor

        Y \*= scaleFactor

        #adjust coordinates to the actual position

        X = MaxX - X + XAdjustment

        Y = maxY - Y + YAdjjustment

        centerCoordinate = (Y, X)

        centerCoordinateRoboticarm.append(centerCoordinate)

    return centerCoordinateRoboticarm

## UserColorSelector.py

ChatGPT inspired because there was not enough time to make PyGame interface

import tkinter as tk

def SelectColor(colorList):

    # Define a dictionary to map color names to their corresponding color codes

    colorMapping = {

        "Red": "Red",

        "Blue": "Blue",

        "Yellow": "Yellow",

        "Green": "Green"

    }

    # Function to handle button click

    def onButtonClick(selectedColor):

        selectedColorVar.set(selectedColor)

        root.destroy()

        root.quit()  # Close the app

    # Create the main application window

    root = tk.Tk()

    root.title("Color Selector")

    # Create a StringVar to store the selected color

    selectedColorVar = tk.StringVar()

    selectedColorVar.set("")

    # Create buttons for each color in the colorList list

    for selectedColor in colorList:

        if selectedColor in colorMapping:

            selectedColor = colorMapping[selectedColor]

            button = tk.Button(root, text=selectedColor, background=selectedColor, command=lambda c=selectedColor: onButtonClick(c))

            button.pack()

    # Label to display the selected color

    colorLabel = tk.Label(root, text="Selected Color: ", background="white")

    colorLabel.pack()

    # Label to display the selected color's name

    selectedColorLabel = tk.Label(root, textvariable=str(selectedColor), background="white")

    selectedColorLabel.pack()

    # Start the GUI event loop

    root.mainloop()

    # When the GUI event loop ends, return the selected color

    return selectedColorVar.get()

if \_\_name\_\_ == "\_\_main\_\_":

    while True:

        colorList = ["Blue","Yellow", "Red", "Green"]

        selectedColor = SelectColor(colorList)

        print("Selected color:", selectedColor)