SDA3

Inhoud

[1. User story 2](#_Toc150950717)

[2. State Diagram 3](#_Toc150950718)

[3. Sequence Diagrams. 4](#_Toc150950719)

[4. Class Diagram 5](#_Toc150950720)

[5. Python code 6](#_Toc150950721)

# User story

**Title: Dobot Arm**

**Priority: #1**

**Estimate: 5d**

**User story**

**As a** Dobot operator*,*

**I want to** have the Dobot arm to autonomously move objects from the conveyor belt to specified coordinates underneath a detection camera, after the objects have been placed on the specified coordinates I want a camera to detect the objects that have been moved.

**so that I can** save time and reduce manual effort*.*

**Acceptance criteria**

**Given that** a Dobot operator has configured the Dobot arm and the camera system*,*

**When** the Dobot is initiated by running the provided code

**then** the following observable outcomes should occur*:*

1. Given the Dobot is in the home position,

When the code is executed,

Then the Dobot should successfully rehome to the specified coordinates (0, -140, 60).

1. Given the Dobot has been rehomed,

When the code executes the specified movement cycles,

Then the Dobot should accurately follow the coordinates provided in the ‘coordinate\_queue’ during each cycle.

1. Given the Dobot has completed all cycles,

When the Dobot returns to the home position,

Then the Dobot should stop the conveyor belt and print a message indicating the successful completion of the task.

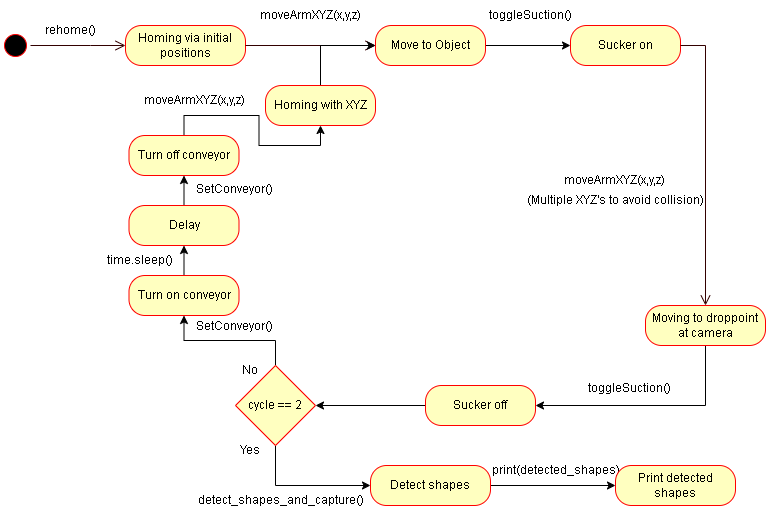
1. Given the camera system is operational,

When the Dobot detects an object of the specified color (red),

Then the Dobot should correctly identify and print out the shape of the detected object (triangle, square, or circle).

Dit is de user story voor ons project. Het is opgesteld wat de wensen zijn gezien vanaf de Dobot operator. Welke gewenste resultaten verwacht worden. En wat het doel is van de automatisering van deze handeling.

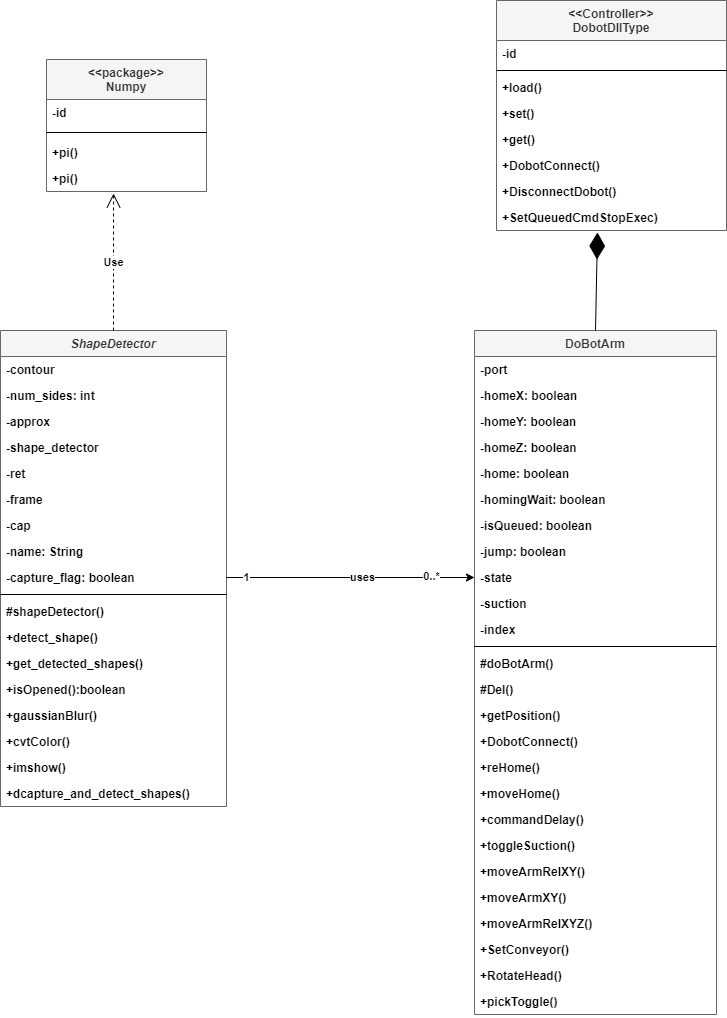
# State Diagram



Hierboven is de state diagram te zien voor onze code. Er is te zien dat hij begint door de rehome() functie aan te roepen die hem naar zijn initiële positie zal bewegen. Als dit gedaan is zal de arm naar het object bewegen en zal de sucker geactiveerd worden, hierna beweegt de arm naar de positie onder de camera en zal de sucker uit gaan. Hij zal deze cycle 3 keer uitvoeren. En zal telkens de conveyor aan en uit zetten om het volgende object naar de positie te bewegen. Als de cycles allemaal gedaan zijn zal de camera detecteren welke objecten er op het vlak liggen en zal dit hierna uitprinten.

# Sequence Diagrams.

# Class Diagram



Hierboven is ons class diagram te zien. Het is opgedeeld in het Numpy pakket, ShapeDetector, DoBotArm en de controller DobotDIIType

# Python code

1. import cv2
2. import numpy as np
3. import time
4. import DoBotArm as Dbt  # Import the DoBotArm module
5. #List of Coordinates Dobot has to follow (x, y, z, suction\_on)
6. coordinate\_queue = [(-1, -143, 14, True), (140, -170, 90, False), (175, 13, -28, True), (0, -140, 60, False)]
7. #Detection of shapes
8. class ShapeDetector:
9. def \_\_init\_\_(self):
10. pass
11. #Detect shapes using contours
12. def detect\_shape(self, contour):
13. peri = cv2.arcLength(contour, True)
14. # Increase camera sensivity.
15. approx = cv2.approxPolyDP(contour, 0.04 \* peri, True)
16. num\_sides = len(approx)
17. #Amount of contours to check what shape is detected.
18. shape = None
19. if num\_sides == 3:
20. shape = "Triangle"
21. elif num\_sides == 4:
22. shape = "Square"
23. else:
24. shape = "Circle"
25. return shape
26. #Detect if the color red is detected within limits
27. def detect\_color(self, image, cX, cY):
28. (b, g, r) = image[cY, cX]
29. if r > 200 and g < 100 and b < 100:
30. return "Red"
31. else:
32. return "Not Red"
33. def detect\_shapes\_and\_capture():
34. # Use camera index 1 for Dobot Camera
35. cap = cv2.VideoCapture(1)
36. shape\_detector = ShapeDetector()
37. capture\_flag = False
38. start\_time = None
39. #Save detected shapes in an Array to print.
40. detected\_shapes = []
41. while True:
42. ret, frame = cap.read()
43. #Change frame to grayscale for better object detection.
44. gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)
45. blurred = cv2.GaussianBlur(gray, (5, 5), 0)
46. edges = cv2.Canny(blurred, 50, 150)
47. contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)
48. #Finding the center of the detected shape
49. for contour in contours:
50. if cv2.contourArea(contour) < 100:
51. continue
52. M = cv2.moments(contour)
53. cX = int(M["m10"] / M["m00"])
54. cY = int(M["m01"] / M["m00"])
55. color = shape\_detector.detect\_color(frame, cX, cY)
56. #If color Red is detected draw a frame around the shape and put the text in the center
57. if color == "Red":
58. detected\_shape = shape\_detector.detect\_shape(contour)
59. cv2.drawContours(frame, [contour], -1, (0, 255, 0), 2)
60. cv2.putText(frame, f"{detected\_shape}", (cX, cY), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)
62. #Timer to keep the camera activated for 5 seconds to prevent any changes.
63. if start\_time is None:
64. start\_time = time.time()
65. if time.time() - start\_time >= 5:
66. detected\_shapes.append(detected\_shape)
67. capture\_flag = True
68. cv2.imshow('frame', frame)
69. #Write the captured image and print if this is successfully completed, also print the detected shapes, and close camera
70. if capture\_flag:
71. cv2.imwrite('captured\_image.png', frame)
72. print("Image captured successfully!")
73. print("Detected Shapes:", detected\_shapes)
74. break
75. #Turn off camera if 'q' is pressed.
76. if cv2.waitKey(1) & 0xFF == ord('q'):
77. break
78. cap.release()
79. cv2.destroyAllWindows()
80. return detected\_shapes
81. def main():
82. # Create an instance of the DoBotArm class
83. dobot = Dbt.DoBotArm(port='COM9', homeX=0, homeY=-140, homeZ=60, home=True, homingWait=True)
84. # Home Dobot
85. dobot.rehome(0, -140, 60)
86. print("Rehoming")
87. # Cycle to change last position in the coordinate\_queue.
88. for cycle in range(3):
89. print(f"Cycle {cycle + 1}")
90. for i, (x, y, z, suction) in enumerate(coordinate\_queue):
91. if cycle == 1 and i == 2:
92. # Change the coordinates for the second cycle
93. x, y, z = 203, -33, -28
94. elif cycle == 2 and i == 2:
95. # Change the coordinates for the third cycle
96. x, y, z = 143, -33, -28
97. elif cycle == 2 and i == 3:
98. # Return to home position
99. x, y, z = 0, -140, 60
100. # Move the Dobot to the specified coordinates
101. dobot.moveArmXYZ(x, y, z)
102. # Turn suction on and off
103. if suction:
104. dobot.toggleSuction()
105. # If the Dobot has reached its final coordinate each step, move up 20 in z-axis before going home (better object placement)
106. if (x, y, z) in [(175, 13, -28), (203, -33, -28), (143, -33, -28)]:
107. dobot.moveArmRelXYZ(0, 0, 20)
108. # Start conveyor belt 1 second after the cycle begins (for the first and second cycles)
109. if cycle < 2:
110. time.sleep(1)
111. dobot.SetConveyor(enabled=True, speed=15000)
112. # Stop conveyor belt after 1 second
113. if cycle < 2:
114. time.sleep(1)
115. dobot.SetConveyor(enabled=False)
116. # Return to the home position
117. dobot.moveArmXYZ(0, -140, 60)
118. time.sleep(2)
119. # Detect shapes using the camera
120. detected\_shapes = detect\_shapes\_and\_capture()
121. print("Shapes Detected by the Camera:", detected\_shapes)
122. if \_\_name\_\_ == "\_\_main\_\_":
123. main()