Implementing the Trashcan Project Using Subsumption Architecture

Erla Hoxha
Business Informatics, Epoka University
erlahoxha04@gmail.com

December 10, 2025

1. Introduction

In this project, I have explored the concept of Subsumption Architecture[1], a revolutionary form of reactive robotics that had a major impact on the robotics field in the 1980s and 1990s. This approach was introduced by Rodney Brooks[2] and his colleagues in 1986. Unlike classical AI's symbolic representation[3],subsumption architecture evolved with a new paradigm focusing on real-world interactions and immediate reaction mechanisms. In this way it gained a large acceptance in autonomous robotics in real-time.

In subsumption architecture, the robot isn't guided by a complex reasoning system. Instead, it operates through a series of simple processes. Each of these processes gives the robot basic capabilities, like avoiding obstacles, randomly navigating an area, or exploring its surroundings. This method follows a bottom-up approach, where sensory inputs dictate the robot's actions rather than relying on predefined symbolic representations of the world. For my project, I structured the robot's behavior into different sub behaviors[1] following Brooks' approach. I organized these behaviors hierarchically, where each level represented a specific degree of behavioral complexity. The lower-level behaviors are controlled by the higher-level ones, which allows simple behaviors to combine and achieve more complex tasks[5]. This method, known as subsumption, helps build systems where basic actions work together to complete sophisticated objectives.

This paper is structured as follows :

- ➢ Section 2: Gives a proper definition and explanation of subsumption architecture and the classical artificial intelligence approach.
- ➢ Section 3: In this section is explained the implementation of subsumption architecture on this project.
- ➢ Section 4: This section explains the algorithmic explanation of the code used for this project.
- ➢ Section 5: Presents the experimental results.
- ➢ Section 6: Conclusion of this paper.

2. Classical Artificial Intelligence Approach and Subsumption Architecture

THE CLASSICAL ARTIFICIAL INTELLIGENCE APPROACH

In the Classical Artificial Intelligence (AI)[4] approach to robot control, everything follows a step-by-step process or also called a hierarchical model. The system goes through different stages like perception, modeling, planning, task execution, and motor control (Fig. 1). Each of these subsystems is a complex program, and all have to work together perfectly for the robot to operate at all[5].This approach relies heavily on symbolic representations of the world so it needs to build an internal model of the world before it can take any action.

From what I have read[6,7], this method works well for handling complex decisions and planning for the long term. However, I've noticed that it struggles in real-time situations. Since it relies on previously made models, it can be slow and require a lot of computing power, which causes delays. Also, as the environment becomes more complex, the system needs even more resources to process and update everything, making it difficult to use in real-world robotics. This "classical" approach to robot control can be thought of as a type of "vertical" division of the control problem into functional units,where each part has a specific job, but none of them can control the robot on their own.

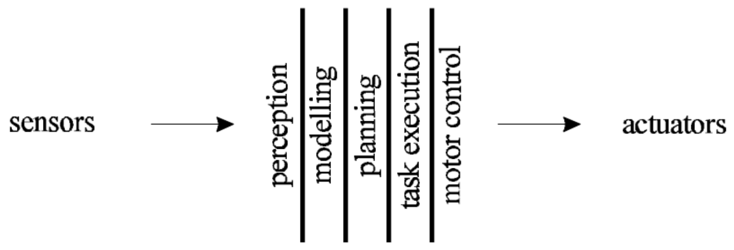


Figure 1. The Classical Artificial Intelligence Approach

SUBSUMPTION ARCHITECTURE

Subsumption architecture is quite different from the classical AI approach, it's like taking the classical method and turning it 90° to look at it from another perspective.In figure 2 I have provided a schema to explain this point of view more. Basically, it’s a reactive architecture, meaning it directly connects sensing to action.

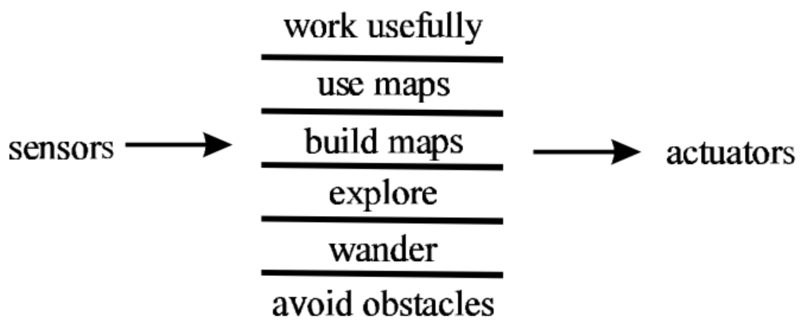


Figure 2. Subsumption Architecture

Coordination in this architecture is competitive. There are multiple vertical layers and the “rule” is that the higher-level are more powerful than the lower layers. Basically higher layers rule lower layers(Fig. 3). However in my understanding this control is not permanent but it's temporary ,because there’s a timestamp that ensures the system can change over time[6]. Rodney Brooks first started by augmenting finite state machines[6] but later extended that to a behavior language, which was an advanced version of the augmented finite state machines. This architecture is important because it has been tested on many different types of robots, including walking and flying robots.

A "classical" subsumption architecture, as described by Brooks [6] consists of a set of complete robot control systems, each of which achieves a particular level of "competence". Layers 0 through 7 are the eight levels of competency that Brooks has identified[2]. On layers 0-2 the robot is programmed to wander around ,to avoid obstacles and to not get stuck but the complex behavior starts on the layers 3 and above[6] are where the complicated behavior begins, as the robot must map the environment, make plans for it, and reason about the state of the world.

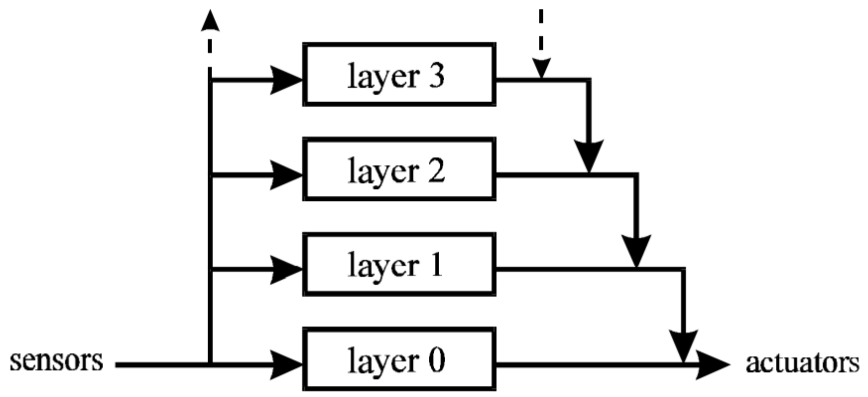


Figure 3. Layers of Subsumption Architecture

3. Subsumption Architecture Implementation

Now everything mentioned above I have put it into consideration for my project. To start off I have designed the subsumption architecture schema where I have followed the Brooks model.

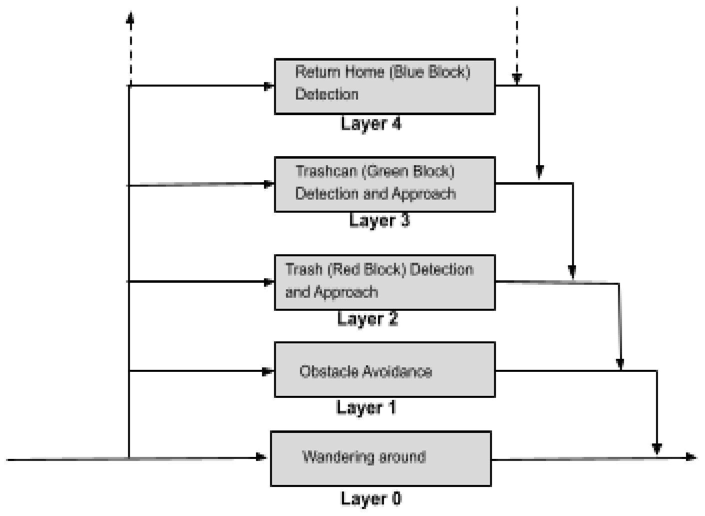

Return Home (Blue Block) Detection
Layer 4
Trashcan (Green Block) Detection and Approach
Layer 3
Trash (Red Block) Detection and Approach
Layer 2
Obstacle Avoidance
Layer 1
Wandering around
Layer 0

Figure 4. Layers of Subsumption Architecture Implementation

For each layer I have provided the proper explanation :

➢ Layer 0. Wandering and Obstacle Avoidance:

  In this layer the robot is programmed to “wander” around the world in e-puck without having a specific goal.

➢ Layer 1. Obstacle detection:

On this layer the robot is programmed to avoid getting stuck between obstacles by using his proximity sensors but also its camera.

➢ Layer 2. Trash (Red Block) Detection and Approach:

When the robot detects a red block (trash), it stops wandering and switches to a focused approach. Using color detection, it locks onto the red block and moves toward it. Once it reaches the block, it prints "The trash is found."

➢ Layer 3. Trashcan (Green Block) Detection and Approach:

So on the third layer the robot after finding the trash is looking for the trashcan (the green block). Just like in the previous layer, it overrides the wandering behavior, locks onto the trash can, and moves toward it. When it reaches the green block, it prints "The trashcan is found."

➢ Layer 4 . Return Home (Blue Block) Detection:

On the last layer the robot's task is to find its way back to home (blue block). When it reaches home, it prints "The trashcan task is completed successfully."

3. Algorithmic Explanation of the Trashcan Project

This project is implemented using a finite state machine[] following the principles of subsumption architecture. The main goal of this project is detection and navigation towards three colored blocks while following a specified rule.

ROBOT INITIALIZATION

The first step is the robot initialization which is done by enabling key devices such as the camera, proximity sensors, and motors. The camera is used to detect the color of the blocks while the sensors which are 8 are used to detect the obstacles. The left and right motors are set to infinite position mode, in order to allow continuous movement. I have also used variables like : **WANDER_SPEED, TARGET_REACHED_THRESHOLD,** and **KP** are used to control the robot's movement dynamics. These initial configurations ensure the robot can properly sense and interact with its environment

COLOR DETECTING MECHANISM

In order for the robot to recognize the blocks in front of him we have used the robot's camera in which we created an algorithmic process with RGB values to determine the most powerful color. But due to the fact that we are dealing with a robot and what seem as simple tasks to us humans for robots those tasks are not that simple so , we thought that these RGB values might be difficult for the robot to detect due to the lighting conditions so we thought to convert them to HSV (Hue Saturation Value)[8]. The saturation value helps to remove or filter out gray or weakly saturated colors.. The hue value is then used to classify blocks as follows:

- Red: $\text{Hue} < 20$ or $\text{Hue} > 340$
- Green: $70 \leq \text{Hue} \leq 150$

- Blue: $200 \leq \text{Hue} \leq 260$

TARGET DETECTION

When the robot finds a target color, it needs to figure out how far the block is from the center of the camera so it can adjust its movement. To do this, I use the **get_target_offset()** function, which scans the camera image and collects the x-coordinates of pixels that match the target color. Then, it calculates the average x-coordinate and checks how far it is from the center.If the error is positive, it means the target is on the left, and if it's negative, the target is on the right. This information is really important because it helps the robot steer correctly as it moves toward the block.

APPROACHING THE TARGET OBJECT

On this step the robot after detecting the target needs to move and go towards the target. To achieve this, I use the **approach_and_lock()** function, which applies a proportional control mechanism. The front proximity sensors help measure the distance to the block, and the robot adjusts its speed based on that data. If we have the situation that the block is far away, the robot moves at full speed, but as it gets closer, the speed gradually gets slower. The offset error from the target tracking function is multiplied by the proportional constant (KP) to adjust the motor speeds:

- The robot will turn left if the error is positive,
- The robot will turn right if the error is negative,
- If the front sensors detect that the target is very close, the robot stops approaching and moves on to the next task.

OBSTACLE AVOIDANCE

In order to make the obstacle avoidance smoother we have used the **avoid_obstacles()** function. This function allows the robot to :

1. Turn right by slowing down the left motor if an obstacle is detected on the left,
2. Turn left by slowing down the right motor if an obstacle is detected on the right,
3. Lastly if the robot doesn’t detect any obstacles he will continue to move forward at a normal speed.

FINITE STATE MACHINE (FSM) FOR TASK EXECUTION

The robot's overall behavior is managed by a state machine, which helps make sure it carries out tasks in an organized way. The state machine has different states that guide the robot through its tasks step by step. Below I have provided the explanation and illustration of each step(Fig. 5& Table 1).

State	Behaviour
SEARCH_RED	The robot wanders around until it detects a red block. Once detected, it switches to APPROACH_RED.
APPROACH_RED	The robot moves toward the red block. When the robot has reached the red block it switches to SEARCH_GREEN
SEARCH_GREEN	The robot searches for the green block (trashcan). If found it switches to APPROACH_GREEN.
APPROACH_GREEN	The robot moves toward the green block and when it's close to it it switches to SEARCH_HOME.
SEARCH_HOME	The robot looks for the blue block(home) once it is close to it it switches to APPROACH_HOME.
APPROACH_HOME	The robot moves closer to the blue block and then stops the mission.

Table 1. The states and their respective behaviour.

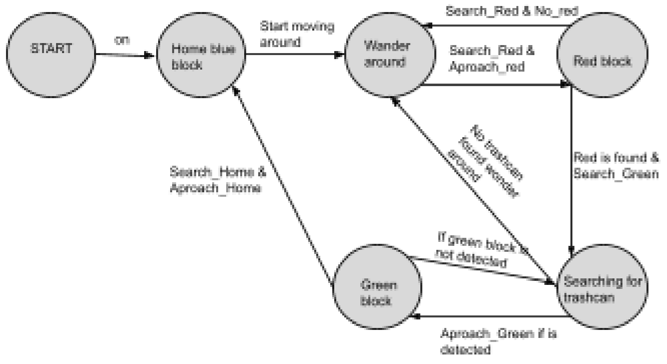
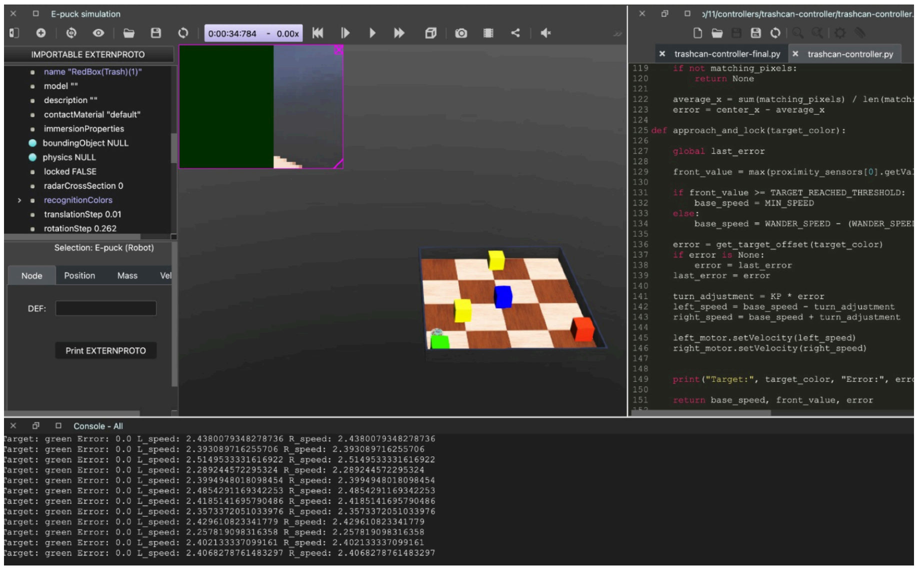

START
on
Home blue block
Start moving around
Wander around
Search_Red & No_red
Search_Red & Aproach_red
Red block
Red is found & Search_Green
No trashcan found wander around
Search_Home & Aproach_Home
If green block is not detected
Green block
Searching for trashcan
Aproach_Green if is detected

Figure 5. The schema of the robot’s tasks.

5. Experimental Results

For my project environment I have used the Webots software since we are familiar with this software. After I implemented my world and control I started to follow around the robot and I have also provided the Figure 6 and Figure 7 to show a part of my experiment. Firstly on the console part we see (Target Green) this means that the robot has found the green block and is heading towards it(Fig. 6). This is done using the robot’s camera and color recognition system, which allows it to identify blocks based on predefined colors. This represents the middle stage of the experiment, which means that the robot is currently on its way to "the trashcan" after first finding the red block that represents the trash. The error value (0.0) shows us how far the target block is from the center of the robot’s vision. Since the error is 0.0, it tells me that the green block is perfectly centered, meaning the robot does not need to make any adjustments. If the error was a different number, the robot would have to turn slightly to correct its alignment.Looking at the motor speeds (L_speed and R_speed), I notice that both values are nearly the same (for example 2.3548 for both). This makes sense because if both motor speeds are equal, the robot moves in a straight line. If there was a big difference between them, that would mean the robot is adjusting its direction to stay on track.

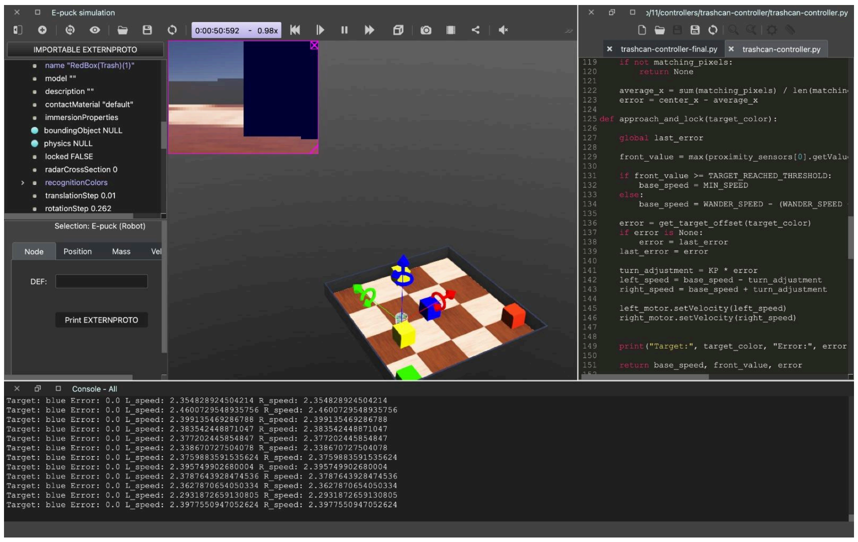


```
    if not matching_pixels:
        return None

    average_x = sum(matching_pixels) / len(matchi
    error = center_x - average_x

def approach_and_lock(target_color):

    global last_error

    front_value = max(proximity_sensors[0].getVa

    if front_value >= TARGET_REACHED_THRESHOLD:
        base_speed = MIN_SPEED
    else:
        base_speed = WANDER_SPEED - (WANDER_SPEE

    error = get_target_offset(target_color)
    if error is None:
        error = last_error
    last_error = error

    turn_adjustment = KP * error
    left_speed = base_speed - turn_adjustment
    right_speed = base_speed + turn_adjustment

    left_motor.setVelocity(left_speed)
    right_motor.setVelocity(right_speed)


    print("Target:", target_color, "Error:", erro

    return base_speed, front_value, error
```

```
Target: green Error: 0.0 L_speed: 2.4380079348278736 R_speed: 2.4380079348278736
Target: green Error: 0.0 L_speed: 2.393089716255706 R_speed: 2.393089716255706
Target: green Error: 0.0 L_speed: 2.5149533331616922 R_speed: 2.5149533331616922
Target: green Error: 0.0 L_speed: 2.289244572295324 R_speed: 2.289244572295324
Target: green Error: 0.0 L_speed: 2.3994948018098454 R_speed: 2.3994948018098454
Target: green Error: 0.0 L_speed: 2.4854291169342253 R_speed: 2.4854291169342253
Target: green Error: 0.0 L_speed: 2.4185141695790486 R_speed: 2.4185141695790486
Target: green Error: 0.0 L_speed: 2.3573372051033976 R_speed: 2.3573372051033976
Target: green Error: 0.0 L_speed: 2.429610823341779 R_speed: 2.429610823341779
Target: green Error: 0.0 L_speed: 2.257819098316358 R_speed: 2.257819098316358
Target: green Error: 0.0 L_speed: 2.402133337099161 R_speed: 2.402133337099161
Target: green Error: 0.0 L_speed: 2.4068278761483297 R_speed: 2.4068278761483297
```

Figure 6. Robot approaching near the green block.

On figure 7 I have provided the final stage of the experiment, which means that the robot is currently on its way to "home" after first finding the red block and then the green block.Firstly on the console part we see (Target Blue) this means that the robot has found the blue block and is heading towards it. The error value again is 0.0 which indicates to us that the blue object is perfectly centered. Also the motors speeds are again almost the same.



```
    if not matching_pixels:
        return None

    average_x = sum(matching_pixels) / len(matchin

    error = center_x - average_x

def approach_and_lock(target_color):

    global last_error

    front_value = max(proximity_sensors[0].getValu

    if front_value >= TARGET_REACHED_THRESHOLD:
        base_speed = MIN_SPEED
    else:
        base_speed = WANDER_SPEED - (WANDER_SPEED

    error = get_target_offset(target_color)
    if error is None:
        error = last_error
    last_error = error

    turn_adjustment = KP * error
    left_speed = base_speed - turn_adjustment
    right_speed = base_speed + turn_adjustment

    left_motor.setVelocity(left_speed)
    right_motor.setVelocity(right_speed)


    print("Target:", target_color, "Error:", error

    return base_speed, front_value, error
```

```
Target: blue Error: 0.0 L_speed: 2.354828924504214 R_speed: 2.354828924504214
Target: blue Error: 0.0 L_speed: 2.4600729548935756 R_speed: 2.4600729548935756
Target: blue Error: 0.0 L_speed: 2.399135469286788 R_speed: 2.399135469286788
Target: blue Error: 0.0 L_speed: 2.383542448871047 R_speed: 2.383542448871047
Target: blue Error: 0.0 L_speed: 2.377202445854847 R_speed: 2.377202445854847
Target: blue Error: 0.0 L_speed: 2.338670727504078 R_speed: 2.338670727504078
Target: blue Error: 0.0 L_speed: 2.3759883591535624 R_speed: 2.3759883591535624
Target: blue Error: 0.0 L_speed: 2.395749902680004 R_speed: 2.395749902680004
Target: blue Error: 0.0 L_speed: 2.3787643928474536 R_speed: 2.3787643928474536
Target: blue Error: 0.0 L_speed: 2.3627870654050334 R_speed: 2.3627870654050334
Target: blue Error: 0.0 L_speed: 2.2931872659130805 R_speed: 2.2931872659130805
Target: blue Error: 0.0 L_speed: 2.3977550947052624 R_speed: 2.3977550947052624
```

Figure 7. Robot approaching near the blue block.

6. Conclusion

In this project, I successfully implemented a reactive robotic system based on the principles of subsumption architecture, a paradigm introduced by Rodney Brooks. By structuring the robot’s behavior into hierarchical layers, I demonstrated how simple and independent behaviors such as wandering, detecting obstacles, and color-based navigation can combine to achieve complex tasks.The robot's ability to locate and move towards specific colored targets, including a red block (representing trash), a green block (representing a trashcan), and a blue block (representing home), showcases the efficiency and adaptability of subsumption-based systems. Each behavior layer operated autonomously while higher layers temporarily overrode lower ones when necessary. This approach allowed the robot to smoothly switch between tasks and avoid obstacles in real-time.

References

[1]Wikipedia Contributors, “Subsumption architecture,” *Wikipedia*[Online], Jun. 14, 2019.Available: https://en.wikipedia.org/wiki/Subsumption_architecture

[2]“Rodney Brooks – Robots, AI, and other stuff,” *Rodneybrooks.com*[Online], 2024. Available: https://rodneybrooks.com/

[3]D. Team, “What is Symbolic AI?,” *Datacamp.com*[Online], May 12, 2023. Available: https://www.datacamp.com/blog/what-is-symbolic-ai

[4]H. Kautz, “Sessions 1 & 2 Introduction to AI; Planning & Search CSE 592 Applications of Artificial Intelligence,” 2003. Accessed: Feb. 09, 2025. [Online]. Available: https://courses.cs.washington.edu/courses/csep573/03wi/lectures/slides/class1.pdf

[5]“Subsumption Architectures Support to Lecture 6.” Accessed: Feb. 09, 2025. [Online]. Available: https://users.dimi.uniud.it/~antonio.dangelo/Robotica/2013/lessons/M02mar20.pdf

[6]D. Toal, C. Flanagan, C. Jones, and B. Strunz, “Subsumption Architecture for the Control of Robots,”[Online] Jan. 01, 1995. Available: https://www.researchgate.net/publication/244321030_Subsumption_Architecture_for_the_Control_of_Robots

[7]M. Z. Kagdi, “Subsumption Architecture” CEN 352 Lecture 8,[PowerPoint slides]. Available: https://drive.google.com/file/d/1asBOUgHTBhaDxgohtDL9434O4pzLgpd-/view

[8]Wikipedia Contributors, “HSL and HSV,” *Wikipedia* [Online], Jun. 23, 2019. Available: https://en.wikipedia.org/wiki/HSL_and_HSV