```python
print("ROS is booting up...")
try:
    import boot.RKernel as kernel
except ImportError as ie:
    print("RKernel not found or failed to load.")
    print(ie)
    exit(11)

print("ROS booted up.")

import time

debug_start_time = time.time()

while True:
    frame = kernel.get_frame()
    kernel.raw_screen = frame
    kernel.set_tensor_input()
    kernel.render_tensor_and_etc()
    kernel.tick_screen()
    if kernel.cv.waitKey(1) & 0xFF == 27:
        break

    if time.time() - debug_start_time > 5:
        # kernel.notification_engine.add_notification("hard_warning.png", "5 seconds passed.")
        # kernel.notification_engine.add_notification("warning.png", "5 seconds passed1.")
        debug_start_time = time.time()

kernel.shutdown()

```

```
1 # rOS EdgeRunner
```

```
boot/__pycache__/RColor.cpython-311.pyc
boot/__pycache__/RFPS.cpython-311.pyc
boot/__pycache__/RKernel.cpython-311.pyc
boot/__pycache__/RKey.cpython-311.pyc
boot/__pycache__/RLabel.cpython-311.pyc
boot/__pycache__/RSound.cpython-311.pyc
boot/__pycache__/RTensor.cpython-311.pyc

```

1 threading 을 통하여 텐서 적용 전에도 프레임 보여주기
2 코드 정리하기
3 프로젝트 구조 정리하기
4 새로운 스플래쉬 스크린 적용
5 라즈베리파이에서 초음파센서 적용
6 라즈베리파이에서 pwm사용하기

```python
import time

tensor_last_update_time = time.time()
main_screen_last_update_time = time.time()
tensor_fps_history = []
main_screen_fps_history = []


def add_candidate_main_fps():
    global main_screen_last_update_time
    global main_screen_fps_history
    main_screen_fps_history.append(1 / (time.time() - main_screen_last_update_time))
    main_screen_last_update_time = time.time()
    if len(main_screen_fps_history) > 10:
        main_screen_fps_history.pop(0)
    return sum(main_screen_fps_history) / len(main_screen_fps_history)


def add_candidate_tensor_fps():
    global tensor_last_update_time
    global tensor_fps_history
    tensor_fps_history.append(1 / (time.time() - tensor_last_update_time))
    tensor_last_update_time = time.time()
    if len(tensor_fps_history) > 10:
        tensor_fps_history.pop(0)
    return sum(tensor_fps_history) / len(tensor_fps_history)


def get_main_screen_fps():
    if len(main_screen_fps_history) == 0:
        return -1
    return sum(main_screen_fps_history) / len(main_screen_fps_history)


def get_tensor_fps():
    if len(tensor_fps_history) == 0:
```

```
37          return -1
38      return sum(tensor_fps_history) / len(
    tensor_fps_history)
39
```

```
keys = {}
positive_signs = ["True", "true", "1", "yes", "Yes", "YES", "on", "On", "ON", "t", "T", "y", "Y"]
negative_signs = ["False", "false", "0", "no", "No", "NO", "off", "Off", "OFF", "f", "F", "n", "N"]


def read_key_line(key_line):
    global keys
    if key_line == "\n": return
    # key format: <name> name </> <type> str </> <value> hello, world! </> <comment> This is a comment. </>
    key_data = key_line.split("</>")
    key_name = key_data[0].split("<name>")[1].strip()
    key_type = key_data[1].split("<type>")[1].strip()
    key_value = key_data[2].split("<value>")[1].strip()
    key_comment = key_data[3].split("<comment>")[1].strip()
    if key_type == "int":
        key_value = int(key_value)
        key_type = type(key_value)
    elif key_type == "float":
        key_value = float(key_value)
        key_type = type(key_value)
    elif key_type == "bool" and key_value in positive_signs:
        key_value = True
        key_type = type(key_value)
    elif key_type == "bool" and key_value in negative_signs:
        key_value = False
        key_type = type(key_value)
    elif key_type == "bool":
        print("Warning!: Invalid boolean value for key: " + key_name + ".")
        key_value = None
    elif key_type == "str":
```

```
31          key_type = type(key_value)
32      else:
33          print("Warning!: Invalid key type for key
    : " + key_name + ".")
34          key_value = None
35
36      keys[key_name] = {"type": key_type, "value":
    key_value, "comment": key_comment}
37
38
39  def __load_keys__():
40      global keys
41      try:
42          r_key_file = open("boot/res/RKey.RKY", "r"
    , encoding="utf-8")
43          key_list = r_key_file.readlines()
44          r_key_file.close()
45          for one_key in key_list: read_key_line(
    one_key)
46      except FileNotFoundError:
47          print("RKey.RKY not found.")
48          return
49      except Exception as e:
50          print("Error loading keys.")
51          print(e)
52          exit(999)
53
54      if keys["BootDebugLogOn"].get("value"):
55          print("\nDebug(BootDebugLogOn): Loaded keys
     are as follows: ")
56          __debug_log_printer__()
57
58
59  def get_key(key_name):
60      global keys
61      if key_name in keys:
62          return keys[key_name]
63      else:
64          return None
65
66
```

```
 67 def set_key(key_name, key_value):
 68     global keys
 69     if key_name in keys:
 70         keys[key_name]["value"] = key_value
 71         save_keys()
 72         if keys["RKeySetDebugLogOn"].get("value"
    ): print(
 73             "\nDebug(RKeySetDebugLogOn): Key " +
    key_name + " set to " + str(key_value) + ".")
 74         return True
 75     else:
 76         return False
 77
 78
 79 def save_key_line(key_name, file):
 80     global keys
 81     key_type = keys[key_name]["type"]
 82     key_value = str(keys[key_name]["value"])
 83     key_comment = keys[key_name]["comment"]
 84     if key_type == int:
 85         key_type = "int"
 86     elif key_type == float:
 87         key_type = "float"
 88     elif key_type == bool:
 89         key_type = "bool"
 90     elif key_type == str:
 91         key_type = "str"
 92     else:
 93         print("Warning!: Invalid key type for key
    : " + key_name + ".")
 94         key_type = "ERROR_HERE!!!"
 95
 96     file.write(
 97         "<name> " + key_name + " </> <type> " +
    key_type + " </> <value> " + key_value + " </> <
    comment> " + key_comment + " </>\n")
 98
 99
100 def save_keys():
101     global keys
102     try:
```

```
103         r_key_file = open("boot/res/RKey.RKY", "w"
    , encoding="utf-8")
104         for key_name in keys: save_key_line(
    key_name, r_key_file)
105         r_key_file.close()
106     except Exception as e:
107         print("Error saving keys.")
108         print(e)
109         exit(999)
110 
111     if keys["RKeySaveDebugLogOn"].get("value"):
112         print("\nDebug(RKeySaveDebugLogOn): Saved
    keys are as follows: ")
113         __debug_log_printer__()
114     return True
115 
116 
117 def __debug_log_printer__():
118     global keys
119     name_max_len = len("Name")
120     type_max_len = len("Type")
121     value_max_len = len("Value")
122     comment_max_len = len("Comment")
123 
124     for key_name in keys:
125         name_max_len = max(name_max_len, len(
    key_name))
126         type_max_len = max(type_max_len, len(str(
    keys[key_name]["type"])))
127         value_max_len = max(value_max_len, len(str
    (keys[key_name]["value"])))
128         comment_max_len = max(comment_max_len, len
    (keys[key_name]["comment"]))
129 
130     print("+" + "-" * (name_max_len + 2) + "+" +
    "-" * (type_max_len + 2) + "+" + "-" * (
131             value_max_len + 2) + "+" + "-" * (
    comment_max_len + 2) + "+")
132     print("| Name" + " " * (name_max_len - 4) +
    " | Type" + " " * (type_max_len - 4) + " | Value"
     + " " * (
```

```
133             value_max_len - 5) + " | Comment" +
    " " * (comment_max_len - 7) + " |")
134     print("+" + "-" * (name_max_len + 2) + "+" +
    "-" * (type_max_len + 2) + "+" + "-" * (
135             value_max_len + 2) + "+" + "-" * (
    comment_max_len + 2) + "+")
136     for key_name in keys:
137         print("| " + key_name + " " * (
    name_max_len - len(key_name)) + " | " + str(
138             keys[key_name]["type"]) + " " * (
139                     type_max_len - len(str(keys[
    key_name]["type"]))) + " | " + str(
140             keys[key_name]["value"]) + " " * (
    value_max_len - len(str(keys[key_name]["value"
    ]))) + " | " +
141               keys[key_name]["comment"] + " " * (
    comment_max_len - len(keys[key_name]["comment"
    ])) + " |")
142         print("+" + "-" * (name_max_len + 2) + "+"
     + "-" * (type_max_len + 2) + "+" + "-" * (
143                 value_max_len + 2) + "+" + "-" * (
    comment_max_len + 2) + "+")
144
145
146 print("RKey engine: loading keys...")
147 __load_keys__()
148 print("RKey engine: keys loaded.")
149
```

```python
try:
    from gtts import gTTS
except ImportError:
    raise ImportError("GTTS not found or failed to load.")
try:
    import io
except ImportError:
    raise ImportError("IO not found or failed to load.")
try:
    import threading
except ImportError:
    raise ImportError("Threading not found or failed to load.")
try:
    import time
except ImportError:
    raise ImportError("Time not found or failed to load.")

sound_engine = None
generator_working = True
tts_generator_thread = None
tts_order_list = []


def tts_generator():
    global tts_order_list
    global generator_working
    while generator_working:
        time.sleep(0.01)
        if len(tts_order_list) == 0: continue
        tts_order = tts_order_list.pop(0)
        play_tts(tts_order[0], lang=tts_order[1])


def order_tts(text, lang='ko'):
    global tts_order_list
    tts_order_list.append((text, lang))

```

```
38
39 def play_tts(text, lang='ko'):
40     tts = gTTS(text=text, lang=lang)
41     fp = io.BytesIO()
42     tts.write_to_fp(fp)
43     fp.seek(0)
44     if sound_engine is not None:
45         sound_engine.play(fp)
46     return fp
47
48
49 def shutdown():
50     try:
51         tts_generator_thread.join()
52     except Exception:
53         pass
54     print("RTTS shutdown.")
55
56
57 tts_generator_thread = threading.Thread(target=
   tts_generator).start()
58
```

```
 1 try:
 2     import time
 3 except ImportError:
 4     raise ImportError("time not found. Please
   install it using 'pip install time'.")
 5 try:
 6     import threading
 7 except ImportError:
 8     raise ImportError("threading not found. Please
   install it using 'pip install threading'.")
 9
10 gpio_engine = None
11
12 echo_pin = 13
13 echo_line = None
14 trigger_pin = 21
15 trigger_line = None
16 output_distance = -1.0
17 ultra_sonic_sensor_thread = None
18 ultra_sonic_sensor_read_enabled = True
19 ultra_sonic_time_out = 0.04
20
21
22 def shutdown():
23     global ultra_sonic_sensor_read_enabled
24     ultra_sonic_sensor_read_enabled = False
25     try:
26         ultra_sonic_sensor_thread.join()
27     except Exception as e:
28         print("Failed to join
   ultra_sonic_sensor_thread.")
29         print(e)
30     pass
31
32
33 def init():
34     global gpio_engine
35     global echo_line
36     global trigger_line
37     if gpio_engine is None:
38         print("asshole, we are very fucked: there
```

```
38 is no gpio_engine in RUSS")
39         raise OSError
40     echo_line = gpio_engine.set_input(echo_pin, "
   echo_pin")
41     trigger_line = gpio_engine.set_output(
   trigger_pin, "trigger_pin", original_state=False)
42     pass
43
44
45 def ultra_sonic_sensor_tick():
46     if gpio_engine is None:
47         return
48     pulse_start = time.time()
49     pulse_end = time.time()
50     timed_out = False
51     gpio_engine.output_write(trigger_line, True)
52     time.sleep(0.00001)
53     gpio_engine.output_write(trigger_line, False)
54     start_time = time.time()
55     while not gpio_engine.input_read(echo_line) and
    pulse_start - start_time <= ultra_sonic_time_out:
56         pulse_start = time.time()
57     if pulse_start - start_time >
   ultra_sonic_time_out:
58         timed_out = True
59     while gpio_engine.input_read(echo_line) and
   pulse_end - start_time <= ultra_sonic_time_out:
60         pulse_end = time.time()
61     if pulse_end - start_time >
   ultra_sonic_time_out:
62         timed_out = True
63
64     if timed_out:
65         return -1
66     pulse_duration = pulse_end - pulse_start
67     return pulse_duration * 171.5
68
69
70 def ultra_sonic_sensor_routine():
71     global output_distance
72     time.sleep(2)
```

```
73      while ultra_sonic_sensor_read_enabled:
74          output_distance = ultra_sonic_sensor_tick
    ()
75          time.sleep(0.1)
76      pass
77
78
79  ultra_sonic_sensor_thread = threading.Thread(
    target=ultra_sonic_sensor_routine).start()
80
```

```python
try:
    import RPi.GPIO as GPIO
except ImportError:
    raise ImportError("RPi.GPIO not found. Please install it using 'sudo apt-get install python3-rpi.gpio'.")
try:
    import threading
except ImportError:
    raise ImportError("threading not found. Please install it using 'pip install threading'.")
try:
    import time
except ImportError:
    raise ImportError("time not found. Please install it using 'pip install time'.")

pwms = {}

try:
    GPIO.setmode(GPIO.BCM)
except Exception as e:
    print("Failed to set GPIO mode.")
    print(e)
    raise e


def set_output(pin):
    GPIO.setup(pin, GPIO.OUT)


def set_input(pin):
    GPIO.setup(pin, GPIO.IN)


def output_write(pin, state):
    GPIO.output(pin, state)


def input_read(pin):
    return GPIO.input(pin)
```

```
38
39
40 def set_pwm(pin, freq, name):
41     pwm = GPIO.PWM(pin, freq)
42     pwm.start(50)
43     pwms[name] = pwm
44     return pwm
45
46
47 def get_pwm(name):
48     return pwms[name]
49
50
51 def pwm_change_freq(pwm, freq):
52     pwm.ChangeFrequency(freq)
53
54
55 def shutdown():
56     for pwm in pwms:
57         pwm.stop()
58     GPIO.cleanup()
59
60 # warning: abandoned
61
```

```
colors = {}


def __read_color_save_line__(color_line):
    global colors
    if color_line == "\n": return
    # color format: label_name = #RRGGBB
    color_data = color_line.split("=")
    color_value = color_data[0].strip()
    color_data = color_data[1].strip()
    color_r = int(color_data[1:3], 16)
    color_g = int(color_data[3:5], 16)
    color_b = int(color_data[5:7], 16)
    # rgb2bgr
    colors[color_value] = (color_b, color_g, color_r)


def __load_colors__():
    try:
        r_color_file = open("boot/res/RClassColor.RCC", "r", encoding="utf-8")
        color_list = r_color_file.readlines()
        r_color_file.close()
        for one_color in color_list: __read_color_save_line__(one_color)

    except FileNotFoundError:
        print("RColor.RCL not found.")
        return
    except Exception as e:
        print("Error loading colors.")
        print(e)
        exit(999)

    label_name_max_len = len("Label Name")
    red_max_len = len("Red")
    green_max_len = len("Green")
    blue_max_len = len("Blue")
    for color_value in colors:
        label_name_max_len = max(label_name_max_len
```

```
38 , len(color_value))
39         red_max_len = max(red_max_len, len(str(
   colors[color_value][2])))
40         green_max_len = max(green_max_len, len(str(
   colors[color_value][1])))
41         blue_max_len = max(blue_max_len, len(str(
   colors[color_value][0])))
42     print("+" + "-" * (label_name_max_len + 2) +
   "+" + "-" * (red_max_len + 2) + "+" + "-" * (
43             green_max_len + 2) + "+" + "-" * (
   blue_max_len + 2) + "+")
44     print(
45         "| Label Name" + " " * (label_name_max_len
    - 10) + " | Red" + " " * (red_max_len - 3) + " |
   Green" + " " * (
46                 green_max_len - 5) + " | Blue" +
   " " * (blue_max_len - 4) + " |")
47     print("+" + "-" * (label_name_max_len + 2) +
   "+" + "-" * (red_max_len + 2) + "+" + "-" * (
48             green_max_len + 2) + "+" + "-" * (
   blue_max_len + 2) + "+")
49     for color_value in colors:
50         print("| " + color_value + " " * (
   label_name_max_len - len(color_value)) + " | " +
   str(
51             colors[color_value][2]) + " " * (
52                     red_max_len - len(str(colors[
   color_value][2]))) + " | " + str(
53             colors[color_value][1]) + " " * (
54                     green_max_len - len(str(
   colors[color_value][1]))) + " | " + str(
55             colors[color_value][0]) + " " * (
   blue_max_len - len(str(colors[color_value][0]))) +
   " |")
56     print("+" + "-" * (label_name_max_len + 2) +
   "+" + "-" * (red_max_len + 2) + "+" + "-" * (
57             green_max_len + 2) + "+" + "-" * (
   blue_max_len + 2) + "+")
58
59
60 def get_color(color_value):
```

```
61      global colors
62      if color_value in colors:
63          return colors[color_value]
64      else:
65          return colors["Else"]
66
67
68  def erase_memory():
69      global colors
70      print("Erasing color memory...")
71      colors = {}
72      print("Color memory erased.")
73
74
75  __load_colors__()
76
```

```python
try:
    import gpiod
except ImportError:
    raise ImportError("gpiod not found. Please install it using 'pip install gpiod'.")
try:
    import time
except ImportError:
    raise ImportError("time not found. Please install it using 'pip install time'.")
try:
    import threading
except ImportError:
    raise ImportError("threading not found. Please install it using 'pip install threading'.")

chip = gpiod.Chip("gpiochip4")

lines = []
pwms = {}


class PWM:
    line = None
    freq = 0
    duty_rate = 0.0
    name = ""
    pwm_run = True
    pwm_thread = None

    def __init__(self, line, freq, duty_rate, name):
        self.line = line
        self.freq = freq
        self.duty_rate = duty_rate
        self.name = name
        self.period = 1 / freq
        self.t_on = self.period * duty_rate
        self.t_off = self.period - self.t_on
        self.pwm_thread = threading.Thread(target=self.pwm_routine).start()
```

```
37
38      def pwm_routine(self):
39          while self.pwm_run: self.pwm_tick()
40
41      def pwm_tick(self):
42          self.line.set_value(1)
43          time.sleep(self.t_on)
44          self.line.set_value(0)
45          time.sleep(self.t_off)
46
47      def pwm_stop(self):
48          self.pwm_run = False
49
50      def change_duty_rate(self, duty_rate):
51          self.duty_rate = duty_rate
52          self.t_on = self.period * duty_rate
53          self.t_off = self.period - self.t_on
54
55      def change_freq(self, freq):
56          self.freq = freq
57          self.period = 1 / freq
58          self.t_on = self.period * self.duty_rate
59          self.t_off = self.period - self.t_on
60
61      def pwm_restart(self):
62          self.pwm_run = False
63          if self.pwm_thread is not None: self.
    pwm_thread.join()
64          self.pwm_thread = threading.Thread(target=
    self.pwm_routine)
65          self.pwm_thread.start()
66
67
68  def set_output(pin, name, original_state=False):
69      line = chip.get_line(pin)
70      line.request(consumer=name, type=gpiod.
    LINE_REQ_DIR_OUT, default_vals=[original_state])
71      lines.append(line)
72      return line
73
74
```

```
 75 def set_input(pin, name):
 76     line = chip.get_line(pin)
 77     line.request(consumer=name, type=gpiod.
    LINE_REQ_DIR_IN)
 78     lines.append(line)
 79     return line
 80 
 81 
 82 def output_write(line, value):
 83     line.set_value(value)
 84 
 85 
 86 def input_read(line):
 87     return line.get_value()
 88 
 89 
 90 def create_pwm(line, freq, name):
 91     pwm = PWM(line, freq, 0, name)
 92     pwms[name] = pwm
 93     return pwm
 94 
 95 
 96 def get_pwm(name):
 97     return pwms[name]
 98 
 99 
100 def shutdown():
101     for pwm in pwms.values():
102         pwm.pwm_stop()
103     for line in lines:
104         line.release()
105     chip.close()
106 
```

```
labels = {}


def read_label_line(label_line):
    global labels
    if label_line == "\n": return
    # label format: label = index % average_width
    label_data = label_line.split("=")
    label_value = label_data[0].strip()
    label_data = label_data[1].split("%")
    label_index = int(label_data[0].strip())
    if len(label_data[1].strip()) > 0:
        label_average_width = float(label_data[1].strip())
    else:
        label_average_width = -1.0
    labels[label_index] = {"value": label_value, "average_width": label_average_width}


def __load_labels__():
    global labels
    try:
        r_label_file = open("boot/res/RClassLabelEn.RCL", "r", encoding="utf-8")
        label_list = r_label_file.readlines()
        r_label_file.close()
        for one_label in label_list:
            read_label_line(one_label)
    except FileNotFoundError:
        print("RClassLabelEn.RCL not found.")
        return
    except Exception as e:
        print("Error loading labels.")
        print(e)
        exit(999)

    label_max_len = len("Label")
    index_max_len = len("Index")
    average_width_max_len = len("Average Width")
    for label_index in labels:
```

```
38          label_max_len = max(label_max_len, len(
   labels[label_index]["value"]))
39          index_max_len = max(index_max_len, len(str(
   label_index)))
40          if labels[label_index]["average_width"
   ] == -1.0: pass
41          else: average_width_max_len = max(
   average_width_max_len, len(str(labels[label_index][
   "average_width"])))
42
43      print("+" + "-" * (label_max_len + 2) + "+" +
   "-" * (index_max_len + 2) + "+" + "-" * (
44              average_width_max_len + 2) + "+")
45      print("| Label" + " " * (label_max_len - 5) +
   " | Index" + " " * (
46              index_max_len - 5) + " | Average Width"
    + " " * (average_width_max_len - 12) + " |")
47      print("+" + "-" * (label_max_len + 2) + "+" +
   "-" * (index_max_len + 2) + "+" + "-" * (
48              average_width_max_len + 2) + "+")
49      for label_index in labels:
50          average_width_specific = labels[label_index
   ]["average_width"]
51          if average_width_specific == -1.0:
   average_width_specific = ""
52          print("| " + labels[label_index]["value"
   ] + " " * (
53                  label_max_len - len(labels[
   label_index]["value"])) + " | " + str(label_index
   ) + " " * (
54                          index_max_len - len(str(
   label_index))) + " | " + str(
55              average_width_specific) + " " * (
56                          average_width_max_len - len(
   str(average_width_specific))) + " |")
57      print("+" + "-" * (label_max_len + 2) + "+" +
   "-" * (index_max_len + 2) + "+" + "-" * (
58              average_width_max_len + 2) + "+")
59
60
61 def get_label(label_index):
```

```
62      global labels
63      if label_index in labels:
64          return labels[label_index]
65      else:
66          return None
67
68
69  def erase_memory():
70      global labels
71      print("Erasing label memory...")
72      labels = {}
73      print("Label memory erased.")
74
75
76  __load_labels__()
77
```

```
 1 try:
 2     import pygame
 3 except ImportError:
 4     raise ImportError("Pygame not found. Please
   install Pygame.")
 5 try:
 6     import threading
 7 except ImportError:
 8     raise ImportError("Threading not found. Please
   install Threading.")
 9
10 channels = []
11 running_thread = None
12 overall_volume = 1.0
13
14
15 def _check_channel(channel):
16     global channels
17     if not channel.get_busy(): channels.remove(
   channel)
18
19
20 def _running():
21     while True:
22         for channel in channels: _check_channel(
   channel)
23         pygame.time.Clock().tick(10)
24         if not channels: break
25
26
27 def play(sound_path, repeat=0, volume=1.0):
28     global running_thread
29     global channels
30     channel = pygame.mixer.Channel(len(channels))
31     sound = pygame.mixer.Sound(sound_path)
32     sound.set_volume(volume * overall_volume)
33     channel.play(sound, repeat)
34     channels.append(channel)
35     if running_thread is None or not running_thread
   .is_alive():
36         running_thread = threading.Thread(target=
```

```
36 _running).start()
37     return channel
38 
39 
40 def stop(channel):
41     global channels
42     if not channel in channels: return
43     channel.stop()
44     channels.remove(channel)
45 
46 
47 def shutdown():
48     pygame.quit()
49     try:
50         running_thread.join()
51     except Exception:
52         pass
53     print("RSound shutdown.")
54 
55 
56 pygame.init()
57 pygame.mixer.init()
58 running_thread = threading.Thread(target=_running).
   start()
59 
```

```
print("RKernel is booting up...")

print("defining pre def methods...")


def make_error(error_code: str, error_message: str
):
    print("E" + error_code + ": " + error_message)
    exit(error_code)


print("methods pre def defined.")

print("Loading Third Party imports...")
try:
    import cv2 as cv
except ImportError:
    make_error("1001", "cv2 not found.")
try:
    import time
except ImportError:
    make_error("1002", "time not found.")
try:
    import picamera2
except ImportError:
    print("Picamera2 not found.")
try:
    import threading
except ImportError:
    make_error("1005", "threading not found.")
try:
    import numpy as np
except ImportError:
    make_error("1006", "numpy not found.")
try:
    import sys
except ImportError:
    make_error("1007", "sys not found.")
try:
    import os
except ImportError:
```

```
41      make_error("1008", "os not found.")
42  print("Third Party Imports loaded.")
43
44  print("Loading RKernel imports...")
45  try:
46      import boot.RKey as key_engine
47  except ImportError:
48      make_error("1101", "RKey not found.")
49  try:
50      import boot.RSound as sound_engine
51  except ImportError:
52      make_error("1102", "RSound not found.")
53  try:
54      import boot.RFPS as fps_engine
55  except ImportError:
56      make_error("1103", "RFPS not found.")
57  try:
58      import boot.RTensor as tensor_engine
59  except ImportError:
60      make_error("1104", "RTensor not found.")
61  except Exception as e:
62      make_error("1104-1", str(e))
63  try:
64      import boot.RLabel as label_engine
65  except ImportError:
66      make_error("1105", "RLabel not found.")
67  try:
68      import boot.RColor as color_engine
69  except ImportError:
70      make_error("1106", "RColor not found.")
71  try:
72      import boot.RBluetooth as bluetooth_engine
73  except ImportError as e:
74      print("RBluetooth not found.")
75  try:
76      import boot.RNotification as
    notification_engine
77  except ImportError:
78      make_error("1108", "RNotification not found.")
79  try:
80      import boot.RTTS as tts_engine
```

```
 81 except ImportError:
 82     make_error("1109", "RTTS not found.")
 83 # try:
 84 #     import boot.RGPIO as gpio_engine
 85 # except ImportError:
 86 #     print("RGPIO not found.")
 87 # try:
 88 #     if "boot.RGPIO" in sys.modules: import boot.
    RUSS as ultrasonic_engine
 89 # except ImportError:
 90 #     make_error("1111", "RUSS not found.")
 91 # try:
 92 #     if "boot.RGPIO" in sys.modules: import boot.
    RTaptic as taptic_engine
 93 # except ImportError:
 94 #     make_error("1112", "RTaptic not found.")
 95 try:
 96     import boot.RGPIOD as gpio_engine
 97 except ImportError:
 98     print("RGPIOD not found.")
 99 try:
100     if "boot.RGPIOD" in sys.modules: import boot.
    RUSS as ultrasonic_engine
101 except ImportError:
102     make_error("1111", "RUSS not found.")
103 try:
104     if "boot.RGPIOD" in sys.modules: import boot.
    RTaptic as taptic_engine
105 except ImportError:
106     make_error("1112", "RTaptic not found.")
107
108 print("RKernel imports loaded.")
109
110 print("defining variables...")
111 splash_screen = cv.imread("boot/res/apple_logo.png
    ")
112 screen = splash_screen
113 raw_screen = splash_screen
114 black_screen = cv.imread("boot/res/black_screen.
    jpg")
115 kernel_panic_screen = cv.imread("boot/res/
```

```
kernel_panic.png")
kernel_panicked = False
camera = None
find_my_keep_sounding_channel = None
find_my_sounding_one_channel = None
find_my_notification = None
boot_loading_bar = 0
hard_warning_icon = cv.imread("boot/res/hard_warning.png")
warning_icon = cv.imread("boot/res/warning.png")
print("variables defined.")

print("defining defs...")


def get_320_320_frame(raw_frame):
    if raw_frame is None:
        global kernel_panicked
        kernel_panicked = True
        raw_frame = kernel_panic_screen
    # make sure the frame is 320x320 and save it to raw_frame
    if raw_frame.shape[0] != 320 or raw_frame.shape[1] != 320:
        # make new_frame but don't flect it
        target_width = 320
        target_height = 320
        aspect_ratio = float(target_height) / raw_frame.shape[0]
        dsize = (int(raw_frame.shape[1] * aspect_ratio), target_height)
        raw_frame = cv.resize(raw_frame, dsize)

        # cut the new_frame width to 320 center
        raw_frame = raw_frame[:,
                    raw_frame.shape[1] // 2 - target_width // 2: raw_frame.shape[1] // 2 + target_width // 2]
    return raw_frame


```

```
149 def make_window():
150     cv.namedWindow("ROS", cv.WINDOW_NORMAL)
151     if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
152         ar_width = key_engine.get_key("
    ARDisplayWidth").get("value")
153         ar_height = key_engine.get_key("
    ARDisplayHeight").get("value")
154         cv.resizeWindow("ROS", ar_width, ar_height
    )
155     else:
156         cv.resizeWindow("ROS", 320, 320)
157 
158 
159 def get_camera():
160     if "picamera2" in sys.modules:
161         camera = picamera2.Picamera2()
162         camera.configure(camera.
    create_preview_configuration(main={"size": (320,
    320)}))
163         camera.start()
164     else:
165         camera = cv.VideoCapture(0)
166     return camera
167 
168 
169 def get_frame():
170     global kernel_panicked
171     if kernel_panicked:
172         return get_320_320_frame(
    kernel_panic_screen)
173     global camera
174     if "picamera2" in sys.modules:
175         frame = camera.capture_array()
176         frame = cv.cvtColor(frame, cv.
    COLOR_BGR2RGB)
177     else:
178         frame = camera.read()[1]
179 
180     if frame is None:
181         print("camera read failed")
```

```
182         kernel_panicked = True
183         return None
184 
185     # check brightness
186     frame_average_red = np.average(frame[:, :, 2])
187     frame_average_green = np.average(frame[:, :, 1
    ])
188     frame_average_blue = np.average(frame[:, :, 0
    ])
189     frame_average_brightness = (frame_average_red
     + frame_average_green + frame_average_blue) / 3
190     if frame_average_brightness < 60 and
    notification_engine.get_notification(message="low
    brightness detected") is None:
191         notification_engine.add_notification("
    hard_warning.png", "low brightness detected", "
    저조도 감지. 사용자의 즉각적인 주의가 필요합니다.")
192 
193     # make frame 320x320
194     return get_320_320_frame(frame)
195 
196 
197 def calculate_distance(class_index: int,
    box_width_pixel):
198     if "picamera2" in sys.modules:
199         distance_calculate_constant = key_engine.
    get_key("
    ROSObjectDistanceCalculateConstantRaspberryPi").
    get("value")
200     else:
201         distance_calculate_constant = key_engine.
    get_key("
    ROSObjectDistanceCalculateConstantMacBookPro").get
    ("value")
202 
203     if label_engine.get_label(class_index + 1) is
    None:
204         object_average_width = -1.0
205     else:
206         object_average_width = label_engine.
    get_label(class_index + 1).get("average_width")
```

```
207
208     if object_average_width == -1.0:
209         return str("N/A")
210
211     distance_in_meter = object_average_width / (
    box_width_pixel / distance_calculate_constant)
212     unit = key_engine.get_key("DistanceUnit").get(
    "value")
213     if unit == "SI" and distance_in_meter < 1:
214         return str(round(distance_in_meter * 100,
    2)) + " cm"
215     elif unit == "SI" and distance_in_meter <=
    1000:
216         return str(round(distance_in_meter, 2)) +
    " m"
217     elif unit == "SI" and distance_in_meter > 1000
    :
218         return str(round(distance_in_meter / 1000
    , 2)) + " km"
219     elif unit == "US" and distance_in_meter < 0.
    3048:
220         return str(round(distance_in_meter * 39.
    3701, 2)) + " in"
221     elif unit == "US" and distance_in_meter <= 0.
    9144:
222         return str(round(distance_in_meter * 3.
    28084, 2)) + " ft"
223     elif unit == "US" and distance_in_meter <=
    1609.34:
224         return str(round(distance_in_meter * 1.
    09361, 2)) + " yd"
225     elif unit == "US" and distance_in_meter > 1609
    .34:
226         return str(round(distance_in_meter / 1609.
    34, 2)) + " mi"
227     else:
228         return str("ERR")
229
230
231 def make_ar_frame(frame):
232     # make sure that input frame is 320x320
```

```
233     frame = get_320_320_frame(frame)
234     # frame input resolution: 320 320
235     ar_width = key_engine.get_key("ARDisplayWidth"
    ).get("value")
236     ar_height = key_engine.get_key("
    ARDisplayHeight").get("value")
237     ar_ppi = key_engine.get_key("ARDisplayPPI").
    get("value")
238     user_eye_distance = key_engine.get_key("
    AREyeDistance").get("value")  # meter
239 
240     ar_screen = np.zeros((ar_height, ar_width, 3
    ), np.uint8)
241     ar_screen = cv.cvtColor(ar_screen, cv.
    COLOR_BGR2RGB)
242 
243     eye_distance_to_inch = user_eye_distance * 39.
    3701
244     eye_distance_to_pixel = int(
    eye_distance_to_inch * ar_ppi)
245 
246     left_eye_center_x = (ar_width // 2) - (
    eye_distance_to_pixel // 2)
247     right_eye_center_x = (ar_width // 2) + (
    eye_distance_to_pixel // 2)
248     eye_center_y = ar_height // 2
249 
250     left_eye_start_x = left_eye_center_x - (320
     // 2)
251     right_eye_start_x = right_eye_center_x - (320
     // 2)
252     eye_start_y = eye_center_y - (320 // 2)
253 
254     preferred_eye = key_engine.get_key("
    ARPreferredEye").get("value")
255 
256     if key_engine.get_key("ARMode").get("value"
    ) == "both eye":
257         # new version
258         left_eye_screen = frame
259         right_eye_screen = frame
```

```
260
261         center_cross_bar_width = 0.005  # meter
262         center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
263
264         left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
265         right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
266
267         left_eye_screen_width = left_eye_max_x -
    left_eye_start_x
268         right_eye_screen_width = right_eye_start_x
     + 320 - right_eye_min_x
269
270         left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
271         right_eye_screen = right_eye_screen[:, 320
     - right_eye_screen_width:]
272
273         ar_screen[eye_start_y:eye_start_y + 320,
274         left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
275         ar_screen[eye_start_y:eye_start_y + 320,
276         right_eye_start_x + 320 -
    right_eye_screen_width:right_eye_start_x + 320] =
    right_eye_screen
277
278     elif key_engine.get_key("ARMode").get("value"
    ) == "one eye" and preferred_eye == "left":
279         # ar_screen[eye_start_y:eye_start_y + 320
    , left_eye_start_x:left_eye_start_x + 320] = frame
280         left_eye_screen = frame
281
282         center_cross_bar_width = 0.01  # meter
283         center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
284
285         left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
286         left_eye_screen_width = left_eye_max_x -
```

```
286 left_eye_start_x
287         left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
288 
289         ar_screen[eye_start_y:eye_start_y + 320,
290         left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
291         pass
292 
293     elif key_engine.get_key("ARMode").get("value"
    ) == "one eye" and preferred_eye == "right":  #
    copy right eye
294         # ar_screen[eye_start_y:eye_start_y + 320
    , right_eye_start_x:right_eye_start_x + 320] =
    frame
295         right_eye_screen = frame
296 
297         center_cross_bar_width = 0.01  # meter
298         center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
299 
300         right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
301         right_eye_screen_width = right_eye_start_x
     + 320 - right_eye_min_x
302         right_eye_screen = right_eye_screen[:, 320
     - right_eye_screen_width:]
303 
304         ar_screen[eye_start_y:eye_start_y + 320,
305         right_eye_start_x + 320 -
    right_eye_screen_width:right_eye_start_x + 320] =
    right_eye_screen
306         pass
307 
308     return ar_screen
309 
310 
311 def render_tensor(screen, boxes, classes, scores,
    distance):
312     if scores < 0.5:
313         return screen
```

```
314     box = boxes * [320, 320, 320, 320]
315     class_name = label_engine.get_label(int(
    classes + 1)).get("value")
316     class_color = color_engine.get_color(
    class_name)
317     inverted_color = (255 - class_color[0], 255 -
    class_color[1], 255 - class_color[2])
318     text_x, text_y = 0, 0  # for text position
319     if key_engine.get_key("ROSMindDisplayWay").get
    ("value") == "filled box":
320         # outer line
321         cv.rectangle(screen, (int(box[1]), int(box
    [0]), int(box[3]), int(box[2])), inverted_color, 2
    )
322         # inner box
323         cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, -1
    )
324         # put text center of the box
325         text_size = cv.getTextSize(class_name, cv.
    FONT_HERSHEY_SIMPLEX, 0.5, 2)[0]
326         text_x = int((box[1] + box[3]) / 2 -
    text_size[0] / 2)
327         text_y = int((box[0] + box[2]) / 2 +
    text_size[1] / 2)
328         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
329                    inverted_color, 2)
330     elif key_engine.get_key("ROSMindDisplayWay").
    get("value") == "outlined box":
331         cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, 2)
332         text_x = int(box[1])
333         text_y = int(box[0])
334         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
335                    class_color, 2)
336
337     if key_engine.get_key("DistanceDisplayEnabled"
    ).get("value"):
338         cv.putText(screen, str(distance), (text_x
```

```
338 , text_y + 20), cv.FONT_HERSHEY_SIMPLEX, 0.5,
339                     inverted_color, 1, cv.LINE_AA)
340 
341     return screen
342 
343 
344 def render_tensors(tensor_output):
345     global raw_screen
346     boxes, classes, scores, distance =
    tensor_output
347     in_print_screen = raw_screen
348     for i in range(len(scores)):
349         in_print_screen = render_tensor(
    in_print_screen, boxes[i], classes[i], scores[i],
    distance[i])
350     return in_print_screen
351 
352 
353 def render_notifications(frame, notifications):
354     one_notification_max_height = 40
355     one_notification_width = 260
356     notification_start_y = 10
357     printed_y_pixel = 0
358 
359     global black_screen
360     global hard_warning_icon
361 
362     for i in range(len(notifications)):
363         # draw notification
364         this_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
365         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
366         this_notification_start_x = (320 -
    this_notification_width) // 2
367         this_notification_radius = int(
    this_notification_height // 2)
368         cv.circle(frame, (
    this_notification_start_x +
```

```
368 this_notification_radius,
369                             notification_start_y +
    this_notification_radius + printed_y_pixel),
370                     this_notification_radius, (255,
    255, 255), -1)
371         cv.circle(frame, (
    this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
372                             notification_start_y +
    this_notification_radius + printed_y_pixel),
373                     this_notification_radius, (255,
    255, 255), -1)
374         cv.rectangle(frame,
375                         (this_notification_start_x +
    this_notification_radius, notification_start_y +
    printed_y_pixel),
376                         (this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
377                          notification_start_y +
    printed_y_pixel + this_notification_height), (255
    , 255, 255), -1)
378         printed_y_pixel +=
    this_notification_height
379
380     printed_y_pixel = 0
381     for i in range(len(notifications) - 1):
382         upper_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
383         lower_notification_height = int(
    one_notification_max_height * notifications[i + 1
    ].display_visibility)
384         upper_notification_radius = int(
    upper_notification_height // 2)
385         lower_notification_radius = int(
    lower_notification_height // 2)
386         upper_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
```

```
387         lower_notification_width = int(
    one_notification_width * notifications[i + 1].
    display_visibility)
388         upper_notification_start_x = (320 -
    upper_notification_width) // 2
389         lower_notification_start_x = (320 -
    lower_notification_width) // 2
390         this_notification_width = min(
    upper_notification_width, lower_notification_width
    )
391         this_notification_start_x = max(
    upper_notification_start_x,
    lower_notification_start_x)
392
393         max_circle_radius = max(
    upper_notification_radius,
    lower_notification_radius)
394         cv.rectangle(frame,
395                      (this_notification_start_x,
    notification_start_y + printed_y_pixel +
    upper_notification_radius),
396                      (this_notification_start_x +
    max_circle_radius,
397                       notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
398                      (255, 255, 255), -1)
399         cv.rectangle(frame, (
    this_notification_start_x +
    this_notification_width - max_circle_radius,
400                              notification_start_y
     + printed_y_pixel + upper_notification_radius),
401                      (this_notification_start_x +
    this_notification_width,
402                       notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
403                      (255, 255, 255), -1)
404         divide_bar_height = 2
405         cv.rectangle(frame, (
    this_notification_start_x,
```

```
406                                 notification_start_y
     + printed_y_pixel + upper_notification_height -
    divide_bar_height // 2),
407                         (this_notification_start_x +
    this_notification_width,
408                          notification_start_y +
    printed_y_pixel + upper_notification_height +
    divide_bar_height // 2),
409                         (200, 200, 200), -1)
410         printed_y_pixel +=
    upper_notification_height
411
412     printed_y_pixel = 0
413     printed_y_pixel_after = 0
414     # now draw notification
415     for i in range(len(notifications)):
416         printed_y_pixel += printed_y_pixel_after
417         this_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
418         printed_y_pixel_after =
    this_notification_height
419         if this_notification_height <= 10:
    continue
420         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
421         this_notification_start_x = (320 -
    this_notification_width) // 2
422         this_notification_radius = int(
    this_notification_height // 2)
423         this_notification_image_size =
    this_notification_height - 10
424         if this_notification_image_size <= 0:
    continue
425         icon = cv.resize(black_screen, (
    this_notification_image_size,
    this_notification_image_size))
426         if notifications[i].icon == "hard_warning.
    png":
427             icon = cv.resize(hard_warning_icon, (
```

```
427 this_notification_image_size,
    this_notification_image_size))
428         elif notifications[i].icon == "warning.png
    ":
429             icon = cv.resize(warning_icon, (
    this_notification_image_size,
    this_notification_image_size))
430         frame[
431         notification_start_y + 5 + printed_y_pixel
    :notification_start_y + 5 + printed_y_pixel +
    this_notification_image_size,
432         this_notification_start_x + 5:
    this_notification_start_x + 5 +
    this_notification_image_size] = icon
433         string_able_pixel_width =
    this_notification_width -
    this_notification_image_size - 10
434         string_able_pixel_height =
    this_notification_height - 10
435         string_max_length =
    string_able_pixel_width // 10
436         string = notifications[i].message
437         if len(string) > string_max_length: string
     = string[:string_max_length] + "..."
438         cv.putText(frame, string, (
    this_notification_start_x +
    this_notification_image_size + 10,
439
    notification_start_y + 5 + printed_y_pixel +
    this_notification_height // 2 + 5),
440                     cv.FONT_HERSHEY_SIMPLEX, 0.5,
441                     (64, 64, 64), 1, cv.LINE_AA)
442
443     return frame
444
445
446 def render_tensor_and_etc():
447     global raw_screen
448     new_frame = raw_screen
449     # render tensor
450     tensor_output = tensor_engine.tensor_output
```

```
451     if tensor_output is not None:
452         new_frame = render_tensors(tensor_output)
453     # render etc
454     # render notifications
455     notifications_count = len(notification_engine.
    notifications)
456     try:
457         if key_engine.get_key("Notification
    Display Overlap").get("value"): new_frame =
    render_notifications(new_frame,
458 
                         notification_engine.notifications)
459     except Exception as e:
460         print("Error while rendering notifications
    .")
461         print(e)
462     # render fps
463     if key_engine.get_key("ROSDisplayFPSEnable").
    get("value"):
464         main_screen_fps = round(fps_engine.
    get_main_screen_fps(), 2)
465         tensor_fps = round(fps_engine.
    get_tensor_fps(), 2)
466         cv.putText(new_frame, "MS FPS: " + str(
    main_screen_fps), (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5,
467                    (255, 255, 255), 1, cv.LINE_AA)
468         cv.putText(new_frame, " T FPS: " + str(
    tensor_fps), (10, 40), cv.FONT_HERSHEY_SIMPLEX, 0.
    5, (255, 255, 255),
469                    1, cv.LINE_AA)
470     global screen
471     screen = new_frame
472     return new_frame
473 
474 
475 def tick_screen():
476     global kernel_panicked
477     global screen
478     if kernel_panicked:
```

```
479         screen = kernel_panic_screen
480     make_window()
481     if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
482         cv.imshow("ROS", make_ar_frame(screen))
483     else:
484         cv.imshow("ROS", screen)
485 
486     fps_engine.add_candidate_main_fps()
487     if fps_engine.get_main_screen_fps() < 20 and
    notification_engine.get_notification(
488             message="Low System FPS detected.") is
     None:
489         notification_engine.add_notification("
    warning.png", "Low System FPS detected.", "시스템
    FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
490     if fps_engine.get_tensor_fps() < 15 and
    notification_engine.get_notification(
491             message="Low Tensor FPS detected.") is
     None:
492         notification_engine.add_notification("
    warning.png", "Low Tensor FPS detected.", "텐서
    FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
493 
494 
495 def set_tensor_input():
496     global raw_screen
497     raw_screen = get_320_320_frame(raw_screen)
498     raw_data = cv.cvtColor(raw_screen, cv.
    COLOR_BGR2RGB)
499     raw_data = np.expand_dims(raw_data, axis=0)
500     tensor_engine.raw_data = raw_data
501 
502 
503 def shutdown():
504     print("Shutting down...")
505     global camera
506     notification_engine.add_notification("
    hard_warning.png", "Shutting down...", "종료 중...")
507     # shutdown thread later
508     tensor_engine.stop_process_frame()
```

```
509     label_engine.erase_memory()
510     color_engine.erase_memory()
511     if "boot.RBluetooth" in sys.modules:
512         bluetooth_engine.close()
513     key_engine.save_keys()
514     cv.destroyAllWindows()
515     if "picamera2" in sys.modules:
516         camera.stop()
517     else:
518         camera.release()
519     notification_engine.close()
520     tts_engine.shutdown()
521     if "boot.RTaptic" in sys.modules:
522         taptic_engine.shutdown()
523     if "boot.RUSS" in sys.modules:
524         ultrasonic_engine.shutdown()
525     if "boot.RGPIO" in sys.modules:
526         gpio_engine.shutdown()
527     current_threads = threading.enumerate()
528     for thread in current_threads:
529         if thread.name == "MainThread": continue
530         print("Thread " + thread.name + " is in
    running.")
531     if len(current_threads) > 1:
532         print("There are still threads running.")
533         print("Force shutdown.")
534         os._exit(0)
535     print("Shutdown complete.")
536     exit(0)
537
538
539 def bluetooth_connected_callback():
540     sound_engine.play("boot/res/alert.mp3")
541     print("Bluetooth connected.")
542     notification_engine.add_notification("warning.
    png", "Bluetooth connected.", "블루투스 연결되었습니다.")
543
544
545 def bluetooth_signal_callback(data):
546     global find_my_sounding_one_channel
547     global find_my_keep_sounding_channel
```

```
548     global find_my_notification
549     if data == b"a":
550         sound_engine.stop(
    find_my_sounding_one_channel)
551         find_my_sounding_one_channel =
    sound_engine.play("boot/res/FindMy.mp3")
552         notification_engine.add_notification("
    warning.png", "Find My is activated.", "나의 찾기가
    활성화 되었습니다.")
553     elif data == b"b":
554         find_my_keep_sounding_channel =
    sound_engine.play("boot/res/FindMy.mp3", -1)
555         find_my_notification = notification_engine
    .add_notification("warning.png", "Find My is
    activated.", "나의 찾기가 활성화 되었습니다.")
556         find_my_notification.display_duration =
    1000000
557         pass
558     elif data == b"c":
559         sound_engine.stop(
    find_my_keep_sounding_channel)
560         find_my_notification.display_duration = 0
561         pass
562
563
564 def boot_logo(started_ticks: float, target_ticks:
    float = 8.0):
565     global screen
566     screen = black_screen
567     global splash_screen
568     resized_splash_screen = cv.resize(
    splash_screen, (80, 80))
569     screen[120:200, 120:200] =
    resized_splash_screen
570
571     # cv.putText(screen, "ROS", (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv.
    LINE_AA)
572     boot_progress = 0
573     if started_ticks < target_ticks * 0.35:
574         # boot_progress = started_ticks * 10.0
```

```
575             boot_progress = started_ticks / (
    target_ticks * 0.35) * 30.0
576             pass
577         elif started_ticks < target_ticks * 0.6:
578             # boot_progress = 30 + (started_ticks - 3
    ) * 30.0
579             # rage is 30 to 90
580             boot_progress = 30 + (started_ticks -
    target_ticks * 0.35) / (target_ticks * 0.25) * 60.
    0
581             pass
582         elif started_ticks < target_ticks * 0.9:
583             # boot_progress = 90 + 10 / 1.5 * (
    started_ticks - 5)
584             # range is 90 to 100
585             boot_progress = 90 + (started_ticks -
    target_ticks * 0.6) / (target_ticks * 0.3) * 10.0
586             pass
587         else:
588             boot_progress = 100
589             pass
590 
591         progress_bar_width_margin = 40
592         progress_bar_height_margin = 10
593         progress_bar_height = 5
594         progress_bar_width = 320 -
    progress_bar_width_margin * 2
595         progress_bar_corner_radius =
    progress_bar_height // 2
596         # progress bar background
597         cv.circle(screen, (progress_bar_width_margin
     + progress_bar_corner_radius,
598                            320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
599                   (64, 64, 64), -1)
600         cv.circle(screen, (320 -
    progress_bar_width_margin -
    progress_bar_corner_radius,
601                            320 -
```

```
601 progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
602                 (64, 64, 64), -1)
603     cv.rectangle(screen, (
604         progress_bar_width_margin +
    progress_bar_corner_radius, 320 -
    progress_bar_height_margin - progress_bar_height),
605                  (320 - progress_bar_width_margin
     - progress_bar_corner_radius, 320 -
    progress_bar_height_margin),
606                  (64, 64, 64), -1)
607     # progress bar
608     cv.circle(screen, (progress_bar_width_margin
     + progress_bar_corner_radius,
609                        320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
610                 (255, 255, 255), -1)
611     cv.rectangle(screen, (
612         progress_bar_width_margin +
    progress_bar_corner_radius, 320 -
    progress_bar_height_margin - progress_bar_height),
613                  (
614                      int(progress_bar_width_margin
     + progress_bar_corner_radius + (
615                                  320 -
    progress_bar_width_margin -
    progress_bar_corner_radius -
    progress_bar_width_margin -
    progress_bar_corner_radius) * (
616                                      boot_progress /
    100)), 320 - progress_bar_height_margin), (255,
    255, 255), -1)
617     cv.circle(screen, (int(
    progress_bar_width_margin +
    progress_bar_corner_radius + (
618             320 - progress_bar_width_margin -
    progress_bar_corner_radius -
    progress_bar_width_margin -
```

```
618 progress_bar_corner_radius) * (
619                                     boot_progress
     / 100)),
620                          320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
621                (255, 255, 255), -1)
622
623
624 print("defs defined.")
625
626 print("preparing RKernel...")
627 tensor_engine.fps_engine = fps_engine
628 tensor_engine.calculate_distance_function =
    calculate_distance
629 notification_engine.tts_engine = tts_engine
630 notification_engine.tts_enabled = key_engine.
    get_key("Notification TTS Enabled").get("value")
631 if "boot.RBluetooth" in sys.modules:
632     print("callback set.")
633     bluetooth_engine.connected_callback =
    bluetooth_connected_callback
634     bluetooth_engine.recv_callback =
    bluetooth_signal_callback
635 camera = get_camera()
636 sound_engine.overall_volume = key_engine.get_key("
    SoundVolume").get("value")
637 if "boot.RGPIOD" in sys.modules:
638     if "boot.RTaptic" in sys.modules:
    taptic_engine.gpio_engine = gpio_engine
639     if "boot.RUSS" in sys.modules:
    ultrasonic_engine.gpio_engine = gpio_engine
640
641 if "boot.RTaptic" in sys.modules:
642     taptic_engine.init()
643
644 if "boot.RUSS" in sys.modules:
645     ultrasonic_engine.init()
646
647 print("RKernel prepared.")
```

```
648
649 # set_tensor_input()
650 if key_engine.get_key("ROSBootChimeEnabled").get("
    value"):
651     sound_engine.play("boot/res/StartUp.mp3")
652
653 started_time = time.time()
654 splash_display_time = key_engine.get_key("
    ROSSplashScreenTime").get("value")
655 while time.time() - started_time <
    splash_display_time:
656     boot_logo(time.time() - started_time,
    splash_display_time)
657     tick_screen()
658     # debug
659     if hard_warning_icon is None:
660         kernel_panicked = True
661     if cv.waitKey(1) & 0xFF == 27:
662         shutdown()
663
664 tts_engine.sound_engine = sound_engine
665 notification_engine.sound_engine = sound_engine
666
```

```
 1 gpio_engine = None
 2
 3 line_left_p = None
 4 line_left_n = None
 5 line_right_p = None
 6 line_right_n = None
 7
 8 pwm_left_p = None
 9 pwm_left_n = None
10 pwm_right_p = None
11 pwm_right_n = None
12
13 pin_left_p = 16
14 pin_left_n = 19
15 pin_right_p = 20
16 pin_right_n = 26
17
18 target_left_freq = 0
19 target_right_freq = 0
20
21 target_left_amp = 0
22 target_right_amp = 0
23
24
25 def shutdown():
26     pass
27
28
29 def init():
30     global gpio_engine
31     global line_left_p
32     global line_left_n
33     global line_right_p
34     global line_right_n
35     global pwm_left_p
36     global pwm_left_n
37     global pwm_right_p
38     global pwm_right_n
39     if gpio_engine is None:
40         print("very fucked: there is no gpio_engine
    lol")
```

```
41          raise OSError
42      line_left_p = gpio_engine.set_output(pin_left_p
    , "taptic_left_p")
43      line_left_n = gpio_engine.set_output(pin_left_n
    , "taptic_left_n")
44      line_right_p = gpio_engine.set_output(
    pin_right_p, "taptic_right_p")
45      line_right_n = gpio_engine.set_output(
    pin_right_n, "taptic_right_n")
46
47      pwm_left_p = gpio_engine.create_pwm(line_left_p
    , 100, "taptic_left_p")
48      pwm_left_n = gpio_engine.create_pwm(line_left_n
    , 100, "taptic_left_n")
49      pwm_right_p = gpio_engine.create_pwm(
    line_right_p, 100, "taptic_right_p")
50      pwm_right_n = gpio_engine.create_pwm(
    line_right_n, 100, "taptic_right_n")
51
52      # debug
53      # pwm_left_p.change_duty_rate(0.5)
54      # pwm_left_n.change_duty_rate(0.2)
55      # pwm_right_p.change_duty_rate(0.8)
56      # pwm_right_n.change_duty_rate(0.5)
57      # pwm_right_n.change_freq(1)
58
59      pass
60
```

```python
try:
    import tensorflow as tf
except ImportError:
    raise ImportError("Tensorflow not found or failed to load.")

try:
    model = tf.lite.Interpreter(model_path="boot/res/model.tflite")
except FileNotFoundError:
    raise FileNotFoundError("Model not found or failed to load.")
except ValueError:
    raise ValueError("Model is invalid.")

try:
    model.allocate_tensors()
except ValueError:
    raise ValueError("Model is invalid.")
try:
    model_input_details = model.get_input_details()
    model_output_details = model.get_output_details()
except ValueError:
    raise ValueError("Model is invalid.")
except IndexError:
    raise IndexError("Model is invalid.")
except TypeError:
    raise TypeError("Model is invalid.")

try:
    import threading
except ImportError:
    raise ImportError("Threading not found or failed to load.")

tensor_output = None
raw_data = None
fps_engine = None
tensor_running = True
calculate_distance_function = None
```

```
37
38
39 def process_frame():
40     global raw_data
41     if raw_data is None:
42         return
43
44     global model
45     global model_input_details
46     global model_output_details
47
48     # set input tensor
49     model.set_tensor(model_input_details[0]['index'
   ], raw_data)
50
51     # invoke model
52     model.invoke()
53
54     # get output tensor
55     boxes_idx, classes_idx, scores_idx = 0, 1, 2
56     boxes = model.get_tensor(model_output_details[
   boxes_idx]['index'])[0]
57     classes = model.get_tensor(model_output_details
   [classes_idx]['index'])[0]
58     scores = model.get_tensor(model_output_details[
   scores_idx]['index'])[0]
59     distances = [None] * len(boxes)
60
61     for i in range(len(boxes)):
62         if scores[i] < 0.5: continue
63         # get box width
64         box_width = (boxes[i][3] - boxes[i][1]) *
   320
65         if calculate_distance_function is not None
   : distances[i] = calculate_distance_function(
   classes[i], box_width)
66         else: distances[i] = None
67
68     global tensor_output
69     tensor_output = (boxes, classes, scores,
   distances)
```

```
70
71     global fps_engine
72     if fps_engine is not None:
73         fps_engine.add_candidate_tensor_fps()
74
75
76 def process_frame_logic():
77     while tensor_running:
78         process_frame()
79
80
81 def stop_process_frame():
82     print("Stopping process frame...")
83     global process_frame_thread
84     global tensor_running
85     tensor_running = False
86     if process_frame_thread is not None:
87         process_frame_thread.join()
88         process_frame_thread = None
89     print("Process frame stopped.")
90
91
92 process_frame_thread = threading.Thread(target=
   process_frame_logic)
93 process_frame_thread.start()
94
```

```python
try:
    import bluetooth
except ImportError:
    raise ImportError("Bluetooth not found or failed to load.")
try:
    import threading
except ImportError:
    raise ImportError("Threading not found or failed to load.")
try:
    import time
except ImportError:
    raise ImportError("Time not found or failed to load.")

client_info = None
client_sock = None
bluetooth_rx_thread = None
bluetooth_connected = False
server_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
server_sock.bind(("", bluetooth.PORT_ANY))
server_sock.listen(1)
connected_callback = None
recv_callback = None


port = server_sock.getsockname()[1]
uuid = "00001101-0000-1000-8000-00805F9B34FB"
service_name = "FindMy"

bluetooth_connect_try_enabled = True


def try_routine():
    global bluetooth_connected
    if bluetooth_connected:
        time.sleep(1)
        return

```

```
38      global client_info
39      global client_sock
40      global bluetooth_rx_thread
41
42      try:
43          client_sock, client_info = server_sock.
    accept()
44          print("Accepted connection from",
    client_info)
45          bluetooth_connected = True
46          bluetooth_rx_thread = threading.Thread(
    target=bluetooth_rx_interrupt).start()
47          if connected_callback is not None:
    connected_callback()
48      except Exception as e:
49          bluetooth_connected = False
50          if client_sock is not None: client_sock.
    close()
51          if bluetooth_rx_thread is not None:
    bluetooth_rx_thread.join()
52          bluetooth_rx_thread = None
53          print("RBluetooth: Error: Error in
    bluetooth connection.")
54          print(e)
55          print("RBluetooth: retrying...")
56          pass
57      pass
58
59
60  def bluetooth_connect_try():
61      global bluetooth_connected
62      global client_sock
63      global client_info
64      global bluetooth_rx_thread
65
66      while bluetooth_connect_try_enabled:
67          try_routine()
68
69
70  def recv():
71      while bluetooth_connected:
```

```
 72         data = client_sock.recv(1024)
 73         if not data: break
 74         print("Received: " + str(data))
 75         if recv_callback is None: continue
 76         recv_callback(data)
 77
 78
 79 def bluetooth_rx_interrupt():
 80     global bluetooth_connected
 81     global client_sock
 82     global recv_callback
 83     try:
 84         recv()
 85     except Exception as e:
 86         print("Error in rx_interrupt.")
 87         print(e)
 88         bluetooth_connected = False
 89         pass
 90
 91
 92 def close():
 93     global bluetooth_rx_thread
 94     global try_thread
 95     global client_sock
 96     global server_sock
 97     global recv_callback
 98
 99     global bluetooth_connect_try_enabled
100     bluetooth_connect_try_enabled = False
101     global bluetooth_connected
102     bluetooth_connected = False
103
104     recv_callback = None
105     print("Bluetooth closed.")
106
107
108 try:
109     bluetooth.advertise_service(server_sock,
    service_name,
110                                 service_id=uuid,
111                                 service_classes=[
```

```
111 uuid, bluetooth.SERIAL_PORT_CLASS],
112                                     profiles=[
    bluetooth.SERIAL_PORT_PROFILE])
113 except Exception as e:
114     print("Error in bluetooth advertise_service.")
115
116 try_thread = threading.Thread(target=
    bluetooth_connect_try).start()
117 print("now you can connect to the bluetooth.")
118
```

```
notifications = []

notifications_management_enabled = True
notifications_management_thread = None
sound_engine = None
tts_engine = None
tts_enabled = False

try:
    import threading
except ImportError:
    raise ImportError("Threading not found or failed to load.")
try:
    import time
except ImportError:
    raise ImportError("Time not found or failed to load.")
try:
    import math
except ImportError:
    raise ImportError("Math not found or failed to load.")


class Notification:
    display_visibility = 0.0
    display_duration = 10.0
    notification_finished = False
    function_x = 0

    def __init__(self, icon, message):
        self.icon = icon
        self.message = message
        self.notification_start_time = time.time()
        self.notification_thread = threading.Thread(target=self.notification_thread_routine).start()

    def notification_thread_routine(self):
        while self.function_x <2.0: self.increase_notification_size()
```

```
37          self.display_visibility = 1.0
38          while time.time() - self.
   notification_start_time < Notification.
   display_duration: time.sleep(0.1)
39          self.function_x = 0
40          while self.function_x < 1.0: self.
   decrease_notification_size()
41          self.display_visibility = 0.0
42          self.notification_finished = True
43          pass
44
45      def increase_notification_size(self):
46          if self.function_x<=1: self.
   display_visibility = math.sqrt(self.function_x)
47          elif self.function_x<=2: self.
   display_visibility = 1 + 0.3535533906*(2 - self.
   function_x) * math.sin(self.function_x-1)
48          self.function_x += 0.01
49          time.sleep(0.005)
50
51      def decrease_notification_size(self):
52          self.display_visibility = math.pow(self.
   function_x-1, 2)
53          self.function_x += 0.01
54          time.sleep(0.005)
55
56
57  def add_notification(icon, notification, saying):
58      global notifications
59      global sound_engine
60      global tts_engine
61      notifications.append(Notification(icon,
   notification))
62      if sound_engine is not None: sound_engine.play(
   "boot/res/alert.mp3", volume=2.0)
63      if tts_engine is not None and tts_enabled:
   tts_engine.order_tts(saying)
64      return notifications[-1]
65
66
67  def get_notification(icon=None, message=None):
```

```
 68     global notifications
 69     if icon is None and message is None: return
    notifications
 70 
 71     candidates = []
 72     for notification in notifications:
 73         if icon is not None and notification.icon
     != icon: continue
 74         if message is not None and notification.
    message != message: continue
 75         candidates.append(notification)
 76 
 77     if len(candidates) == 0: return None
 78     elif len(candidates) == 1: return candidates[0
    ]
 79     else: return candidates
 80 
 81 
 82 def notification_management_routine(notification):
 83     global notifications
 84     if notification.notification_finished:
 85         notifications.remove(notification)
 86         return
 87     pass
 88 
 89 
 90 def notifications_management_routine():
 91     global notifications
 92     global notifications_management_enabled
 93     while notifications_management_enabled:
 94         for notification in notifications:
    notification_management_routine(notification)
 95         time.sleep(0.01)
 96 
 97 
 98 def close():
 99     global notifications_management_enabled
100     notifications_management_enabled = False
101     global notifications_management_thread
102     if notifications_management_thread is not None
    : notifications_management_thread.join
```

```
103     pass
104 
105 
106 notifications_management_thread = threading.Thread
    (target=notifications_management_routine).start()
107 
```

```
 1 <name> ROSVersion </> <type> str </> <value>
   EdgeRunner.4.beta3 </> <comment> ROSVersion </>
 2 <name> ROSBootChimeEnabled </> <type> bool </> <
   value> True </> <comment>
   ROSBootChimeEnabledNoticeROS boot chime is enabled
   or not </>
 3 <name> ROSSplashScreenTime </> <type> float </> <
   value> 6.0 </> <comment>
   ROSSplashScreenTimeNoticeROS splash screen time </>
 4 <name> ROSIsOn </> <type> bool </> <value> False
    </> <comment> ROSIsOnNoticeROS is shutdown
   properly </>
 5 <name> BootDebugLogOn </> <type> bool </> <value>
   False </> <comment> BootDebugLogOnNoticeBoot debug
   log is on or off </>
 6 <name> RKeySetDebugLogOn </> <type> bool </> <value
   > False </> <comment>
   RKeySetDebugLogOnNoticeRKeySet debug log is on or
   off </>
 7 <name> RKeySaveDebugLogOn </> <type> bool </> <
   value> False </> <comment>
   RKeySaveDebugLogOnNoticeRKeySave debug log is on or
    off </>
 8 <name> CameraDevice </> <type> str </> <value>
   macbook pro </> <comment> CameraDeviceName </>
 9 <name> ROSRunningDevice </> <type> str </> <value>
   macbook pro </> <comment> ROSRunningDeviceNoticeROS
    running device </>
10 <name> ROSModelActive </> <type> bool </> <value>
   True </> <comment> ROSModelActiveNoticeROS model is
    active or not </>
11 <name> ROSMindDisplayWay </> <type> str </> <value
   > filled box </> <comment>
   ROSMindDisplayWayNoticeROS mind display way </>
12 <name> ROSDisplayFPSEnable </> <type> bool </> <
   value> True </> <comment>
   ROSDisplayFPSEnableNoticeROS display FPS is enable
   or not </>
13 <name> SLDDeviceIP </> <type> str </> <value> 0.0.0
   .0 </> <comment> SLDDeviceIPNoticeSLD device IP </>
14 <name> SLDDevicePort </> <type> int </> <value>
```

```
14 5001 </> <comment> SLDDevicePortNoticeSLD device
   port </>
15 <name> SLDConnectTimeout </> <type> int </> <value
   > 5 </> <comment> SLDConnectTimeoutNoticeSLD
   connect timeout </>
16 <name> ROSObjectDistanceCalculateConstantDefault
    </> <type> float </> <value> 500.0 </> <comment>
   ROSObjectDistanceCalculateConstantNoticeROS object
   distance calculate constant </>
17 <name> DistanceUnit </> <type> str </> <value> SI
    </> <comment> DistanceUnitNoticeDistance unit </>
18 <name> DistanceDisplayEnabled </> <type> bool </> <
   value> True </> <comment>
   DistanceDisplayEnabledNoticeDistance display is
   enabled or not </>
19 <name> DistanceDisplayWay </> <type> str </> <value
   > invertedColorInBox </> <comment>
   DistanceDisplayWayNoticeDistance display way </>
20 <name> ROSARDisplayEnabled </> <type> bool </> <
   value> True </> <comment>
   ROSARDisplayEnabledNoticeROS AR display is enabled
   or not </>
21 <name> AREyeDistance </> <type> float </> <value> 0
   .058 </> <comment> AREyeDistanceNoticeAR eye
   distance in meter </>
22 <name> ARDisplayPPI </> <type> int </> <value> 131
    </> <comment> ARDisplayPPINoticeAR display PPI </>
23 <name> ARDisplayWidth </> <type> int </> <value>
   800 </> <comment> ARDisplayWidthNoticeAR display
   width </>
24 <name> ARDisplayHeight </> <type> int </> <value>
   480 </> <comment> ARDisplayHeightNoticeAR display
   height </>
25 <name> ARPreferredEye </> <type> str </> <value>
   right </> <comment> ARPreferredEyeNoticeAR
   preferred eye </>
26 <name> ARMode </> <type> str </> <value> both eye
    </> <comment> ARModeFor auto or Both Eye </>
27 <name>
   ROSObjectDistanceCalculateConstantRaspberryPi </> <
   type> float </> <value> 500.0 </> <comment>
```

```
27
   ROSObjectDistanceCalculateConstantRaspberryPiNotice
   ROS object distance calculate constant for
   Raspberry Pi </>
28 <name> ROSObjectDistanceCalculateConstantMacBookPro
    </> <type> float </> <value> 500.0 </> <comment>
   ROSObjectDistanceCalculateConstantMacBookProNoticeR
   OS object distance calculate constant for MacBook
   Pro </>
29 <name> ROSObjectDistanceCalculateConstantIphone
    </> <type> float </> <value> 500.0 </> <comment>
   ROSObjectDistanceCalculateConstantIphoneNoticeROS
   object distance calculate constant for iPhone </>
30 <name> SoundVolume </> <type> float </> <value> 0.3
    </> <comment> SoundVolumeNoticeSound volume </>
31 <name> Notification Display Overlap </> <type> bool
    </> <value> True </> <comment> Notification
   Display OverlapNoticeNotification display overlap
    </>
32 <name> Notification TTS Enabled </> <type> bool
    </> <value> True </> <comment> Notification TTS
   EnabledNoticeNotification TTS is enabled or not </>
33
```

```
1 Person = #0000ff
2 Car = #ff0000
3 Knife = #ff0000
4 Bicycle = #ffa500
5 Else = #000000
6
```

```
 1 Person = 1 % 0.55
 2 Bicycle = 2 %
 3 Car = 3 %
 4 motorcycle = 4 %
 5 Airplane = 5 %
 6 Bus = 6 %
 7 Train = 7 %
 8 Truck = 8 %
 9 Boat = 9 %
10 Traffic light = 10 %
11 Fire hydrant = 11 %
12 ??? = 12 %
13 Stop sign = 13 %
14 Parking meter = 14 %
15 Bench = 15 %
16 Bird = 16 %
17 Cat = 17 %
18 Dog = 18 %
19 Horse = 19 %
20 Sheep = 20 %
21 Cow = 21 %
22 Elephant = 22 %
23 Bear = 23 %
24 Zebra = 24 %
25 Giraffe = 25 %
26 ??? = 26 %
27 Backpack = 27 %
28 Umbrella = 28 %
29 ??? = 29 %
30 ??? = 30 %
31 Handbag = 31 %
32 Tie = 32 %
33 Suitcase = 33 %
34 Frisbee = 34 %
35 Skis = 35 %
36 Snowboard = 36 %
37 Sports ball = 37 %
38 Kite = 38 %
39 Baseball bat = 39 %
40 Baseball glove = 40 %
41 Skateboard = 41 %
```

```
42 Surfboard = 42 %
43 Tennis racket = 43 %
44 Bottle = 44 % 0.06
45 ??? = 45 %
46 Wine glass = 46 %
47 Cup = 47 %
48 Fork = 48 %
49 Knife = 49 %
50 Spoon = 50 %
51 Bowl = 51 %
52 Banana = 52 %
53 Apple = 53 %
54 Sandwich = 54 %
55 Orange = 55 %
56 Broccoli = 56 %
57 Carrot = 57 %
58 Hot dog = 58 %
59 Pizza = 59 %
60 Donut = 60 %
61 Cake = 61 %
62 Chair = 62 %
63 Couch = 63 %
64 Potted plant = 64 %
65 Bed = 65 %
66 ??? = 66 %
67 Dining table = 67 %
68 ??? = 68 %
69 ??? = 69 %
70 Toilet = 70 %
71 ??? = 71 %
72 TV = 72 %
73 Laptop = 73 % 0.37
74 Mouse = 74 %
75 Remote = 75 %
76 Keyboard = 76 % 0.45
77 Cell phone = 77 %
78 Microwave = 78 %
79 Oven = 79 %
80 Toaster = 80 %
81 Sink = 81 %
82 Refrigerator = 82 %
```

```
83 ??? = 83 %
84 Book = 84 %
85 Clock = 85 % 0.38
86 Vase = 86 %
87 Scissors = 87 %
88 Teddy bear = 88 %
89 Hair drier = 89 %
90 Toothbrush = 90 %
91
```