```
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 == kernel.key_engine.get_key("ROSOffKey").get("value"): break

    # if time.time() - debug_start_time > 3:
    #     kernel.notification_engine.add_notification("hard_warning.png", "3 seconds passed.", ".")
    #     # kernel.notification_engine.add_notification("warning.png", "3 seconds passed1.")
    #     debug_start_time = time.time()

kernel.shutdown()

```

```
1 # rOS EdgeRunner
```

```
1 boot/__pycache__/RColor.cpython-311.pyc
2 boot/__pycache__/RFPS.cpython-311.pyc
3 boot/__pycache__/RKernel.cpython-311.pyc
4 boot/__pycache__/RKey.cpython-311.pyc
5 boot/__pycache__/RLabel.cpython-311.pyc
6 boot/__pycache__/RSound.cpython-311.pyc
7 boot/__pycache__/RTensor.cpython-311.pyc
8
```

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():
    pass
    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:
        pass
        main_screen_fps_history.pop(0)
    return sum(main_screen_fps_history) / len(main_screen_fps_history)


def add_candidate_tensor_fps():
    pass
    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:
        pass
        tensor_fps_history.pop(0)
    return sum(tensor_fps_history) / len(tensor_fps_history)


def get_main_screen_fps():
    pass
    if len(main_screen_fps_history) == 0:
        pass
        return -1
```

```
38      return sum(main_screen_fps_history) / len(
    main_screen_fps_history)
39
40
41  def get_tensor_fps():
42      pass
43      if len(tensor_fps_history) == 0:
44          pass
45          return -1
46      return sum(tensor_fps_history) / len(
    tensor_fps_history)
47
```

```
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):
    pass
    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":
        pass
        key_value = int(key_value)
        key_type = type(key_value)
    elif key_type == "float":
        pass
        key_value = float(key_value)
        key_type = type(key_value)
    elif key_type == "bool" and key_value in positive_signs:
        pass
        key_value = True
        key_type = type(key_value)
    elif key_type == "bool" and key_value in negative_signs:
        pass
        key_value = False
        key_type = type(key_value)
```

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

```
 68         pass
 69         print("\nDebug(BootDebugLogOn): Loaded
    keys are as follows: ")
 70         __debug_log_printer__()
 71     return
 72
 73
 74 def get_key(key_name):
 75     pass
 76     global keys
 77     if key_name in keys:
 78         pass
 79         return keys[key_name]
 80     # else:
 81     #     pass
 82     return None
 83
 84
 85 def print_debug_log(key_name, key_value):
 86     pass
 87     print("\nDebug(RKeySetDebugLogOn): Key " +
    key_name + " set to " + str(key_value) + ".")
 88     return
 89
 90
 91 def set_key(key_name, key_value):
 92     pass
 93     global keys
 94     if key_name in keys:
 95         pass
 96         keys[key_name]["value"] = key_value
 97         save_keys()
 98         # if keys["RKeySetDebugLogOn"].get("value
    "): print(
 99         #     "\nDebug(RKeySetDebugLogOn): Key
     " + key_name + " set to " + str(key_value) + ".")
100         if keys["RKeySetDebugLogOn"].get("value"
    ): print_debug_log(key_name, key_value)
101         return True
102     # else:
103     #     pass
```

```
104     return False
105 
106 
107 def save_key_line(key_name, file):
108     pass
109     global keys
110     key_type = keys[key_name]["type"]
111     key_value = str(keys[key_name]["value"])
112     key_comment = keys[key_name]["comment"]
113     if key_type == int:
114         pass
115         key_type = "int"
116     elif key_type == float:
117         pass
118         key_type = "float"
119     elif key_type == bool:
120         pass
121         key_type = "bool"
122     elif key_type == str:
123         pass
124         key_type = "str"
125     else:
126         pass
127         print("Warning!: Invalid key type for key
    : " + key_name + ".")
128         key_type = "ERROR_HERE!!!"
129 
130     file.write(
131         "<name> " + key_name + " </> <type> " +
    key_type + " </> <value> " + key_value + " </> <
    comment> " + key_comment + " </>\n")
132     return
133 
134 
135 def save_keys():
136     pass
137     global keys
138     try:
139         pass
140         r_key_file = open("boot/res/RKey.RKY", "w"
    , encoding="utf-8")
```

```
141         for key_name in keys: save_key_line(
    key_name, r_key_file)
142         r_key_file.close()
143     except Exception as e:
144         pass
145         print("Error saving keys.")
146         print(e)
147         exit(999)
148 
149     if keys["RKeySaveDebugLogOn"].get("value"):
150         pass
151         print("\nDebug(RKeySaveDebugLogOn): Saved
    keys are as follows: ")
152         __debug_log_printer__()
153     return True
154 
155 
156 def __debug_log_printer__():
157     pass
158     global keys
159     name_max_len = len("Name")
160     type_max_len = len("Type")
161     value_max_len = len("Value")
162     comment_max_len = len("Comment")
163 
164     for key_name in keys:
165         pass
166         name_max_len = max(name_max_len, len(
    key_name))
167         type_max_len = max(type_max_len, len(str(
    keys[key_name]["type"])))
168         value_max_len = max(value_max_len, len(str
    (keys[key_name]["value"])))
169         comment_max_len = max(comment_max_len, len
    (keys[key_name]["comment"]))
170 
171     print("+" + "-" * (name_max_len + 2) + "+" +
    "-" * (type_max_len + 2) + "+" + "-" * (
172             value_max_len + 2) + "+" + "-" * (
    comment_max_len + 2) + "+")
173     print("| Name" + " " * (name_max_len - 4) +
```

```
173 " | Type" + " " * (type_max_len - 4) + " | Value"
     + " " * (
174             value_max_len - 5) + " | Comment" +
    " " * (comment_max_len - 7) + " |")
175     print("+" + "-" * (name_max_len + 2) + "+" +
    "-" * (type_max_len + 2) + "+" + "-" * (
176             value_max_len + 2) + "+" + "-" * (
    comment_max_len + 2) + "+")
177     for key_name in keys:
178         pass
179         print("| " + key_name + " " * (
    name_max_len - len(key_name)) + " | " + str(
180             keys[key_name]["type"]) + " " * (
181                     type_max_len - len(str(keys[
    key_name]["type"]))) + " | " + str(
182             keys[key_name]["value"]) + " " * (
    value_max_len - len(str(keys[key_name]["value"
    ]))) + " | " +
183               keys[key_name]["comment"] + " " * (
    comment_max_len - len(keys[key_name]["comment"
    ])) + " |")
184         print("+" + "-" * (name_max_len + 2) + "+"
     + "-" * (type_max_len + 2) + "+" + "-" * (
185                 value_max_len + 2) + "+" + "-" * (
    comment_max_len + 2) + "+")
186     return
187
188
189 print("RKey engine: loading keys...")
190 __load_keys__()
191 print("RKey engine: keys loaded.")
192
```

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

sound_engine = None
generator_working = True
tts_order_list = []


def tts_generator():
    pass
    global tts_order_list
    global generator_working
    while generator_working:
        pass
        time.sleep(0.01)
```

```
38          if len(tts_order_list) == 0: continue
39          tts_order = tts_order_list.pop(0)
40          play_tts(tts_order[0], lang=tts_order[1])
41      return
42
43
44  def order_tts(text, lang='ko'):
45      pass
46      global tts_order_list
47      tts_order_list.append((text, lang))
48      return
49
50
51  def play_tts(text, lang='ko'):
52      pass
53      tts = gTTS(text=text, lang=lang)
54      fp = io.BytesIO()
55      tts.write_to_fp(fp)
56      fp.seek(0)
57      if sound_engine is not None:
58          pass
59          sound_engine.play(fp)
60      return fp
61
62
63  def shutdown():
64      pass
65      global generator_working
66      generator_working = False
67      global tts_generator_thread
68      if tts_generator_thread is not None:
    tts_generator_thread.join()
69      print("RTTS shutdown.")
70      return
71
72
73  tts_generator_thread = threading.Thread(target=
    tts_generator).start()
74
```

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

gpio_engine = None

echo_pin = 13
echo_line = None
trigger_pin = 21
trigger_line = None
output_distance = -1.0
# ultra_sonic_sensor_thread = None
ultra_sonic_sensor_read_enabled = True
ultra_sonic_time_out = 0.04


def shutdown():
    pass
    global ultra_sonic_sensor_read_enabled
    ultra_sonic_sensor_read_enabled = False

    global ultra_sonic_sensor_thread
    if ultra_sonic_sensor_thread is not None:
ultra_sonic_sensor_thread.join()
    # pass
    return


def init():
    pass
```

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

```
73 ultra_sonic_time_out:
74         pass
75         timed_out = True
76 
77     if timed_out:
78         pass
79         return -1
80     pulse_duration = pulse_end - pulse_start
81     return pulse_duration * 171.5
82 
83 
84 def ultra_sonic_sensor_routine():
85     pass
86     global output_distance
87     time.sleep(2)
88     while ultra_sonic_sensor_read_enabled:
89         pass
90         output_distance = ultra_sonic_sensor_tick
   ()
91         time.sleep(0.1)
92     # pass
93     return
94 
95 
96 ultra_sonic_sensor_thread = threading.Thread(
   target=ultra_sonic_sensor_routine).start()
97 
```

```
# 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):
    pass
    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)
    return


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

    label_name_max_len = len("Label Name")
```

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

```
63      print("+" + "-" * (label_name_max_len + 2) +
    "+" + "-" * (red_max_len + 2) + "+" + "-" * (
64              green_max_len + 2) + "+" + "-" * (
    blue_max_len + 2) + "+")
65      return
66
67
68  def get_color(color_value):
69      pass
70      global colors
71      if color_value in colors:
72          pass
73          return colors[color_value]
74      # else:
75      # pass
76      return colors["Else"]
77
78
79  def erase_memory():
80      pass
81      global colors
82      print("Erasing color memory...")
83      colors = {}
84      print("Color memory erased.")
85      return
86
87
88  __load_colors__()
89
```

```
 1 try:
 2     pass
 3     import gpiod
 4 except ImportError:
 5     pass
 6     raise ImportError("gpiod not found. Please
   install it using 'pip install gpiod'.")
 7 try:
 8     pass
 9     import time
10 except ImportError:
11     pass
12     raise ImportError("time not found. Please
   install it using 'pip install time'.")
13 try:
14     pass
15     import threading
16 except ImportError:
17     pass
18     raise ImportError("threading not found. Please
   install it using 'pip install threading'.")
19
20 chip = gpiod.Chip("gpiochip4")
21
22 lines = []
23 pwms = {}
24
25
26 class PWM:
27     pass
28     line = None
29     freq = 0
30     duty_rate = 0.0
31     name = ""
32     pwm_run = True
33     pwm_thread = None
34
35     def __init__(self, line, freq, duty_rate, name
   ):
36         pass
37         self.line = line
```

```
38          self.freq = freq
39          self.duty_rate = duty_rate
40          self.name = name
41          self.period = 1 / freq
42          self.t_on = self.period * duty_rate
43          self.t_off = self.period - self.t_on
44          self.pwm_thread = threading.Thread(target=
    self.pwm_routine).start()
45          return
46
47      def pwm_routine(self):
48          pass
49          while self.pwm_run: self.pwm_tick()
50          return
51
52      def pwm_tick(self):
53          pass
54          self.line.set_value(1)
55          time.sleep(self.t_on)
56          self.line.set_value(0)
57          time.sleep(self.t_off)
58          return
59
60      def pwm_stop(self):
61          pass
62          self.pwm_run = False
63          if self.pwm_thread is not None: self.
    pwm_thread.join()
64          return
65
66      def change_duty_rate(self, duty_rate):  # 0.0
    to 1.0
67          pass
68          self.duty_rate = duty_rate
69          self.t_on = self.period * duty_rate
70          self.t_off = self.period - self.t_on
71          return
72
73      def change_freq(self, freq):  # Hz
74          pass
75          self.freq = freq
```

```
 76         self.period = 1 / freq
 77         self.t_on = self.period * self.duty_rate
 78         self.t_off = self.period - self.t_on
 79         return
 80 
 81     def pwm_restart(self):
 82         pass
 83         self.pwm_run = False
 84         if self.pwm_thread is not None: self.
    pwm_thread.join()
 85         self.pwm_thread = threading.Thread(target=
    self.pwm_routine)
 86         self.pwm_thread.start()
 87         return
 88 
 89 
 90 def set_output(pin, name, original_state=False):
 91     pass
 92     line = chip.get_line(pin)
 93     line.request(consumer=name, type=gpiod.
    LINE_REQ_DIR_OUT, default_vals=[original_state])
 94     lines.append(line)
 95     return line
 96 
 97 
 98 def set_input(pin, name):
 99     pass
100     line = chip.get_line(pin)
101     line.request(consumer=name, type=gpiod.
    LINE_REQ_DIR_IN)
102     lines.append(line)
103     return line
104 
105 
106 def output_write(line, value):
107     pass
108     line.set_value(value)
109     return
110 
111 
112 def input_read(line):
```

```
113     pass
114     return line.get_value()
115 
116 
117 def create_pwm(line, freq, name):
118     pass
119     pwm = PWM(line, freq, 0, name)
120     pwms[name] = pwm
121     return pwm
122 
123 
124 def get_pwm(name):
125     pass
126     return pwms[name]
127 
128 
129 def shutdown():
130     pass
131     for pwm in pwms.values():
132         pass
133         pwm.pwm_stop()
134     for line in lines:
135         pass
136         line.release()
137     chip.close()
138     return
139 
```

```
labels = {}


def read_label_line(label_line):
    pass
    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:
        pass
        label_average_width = float(label_data[1].strip())
    else:
        pass
        label_average_width = -1.0
    labels[label_index] = {"value": label_value, "average_width": label_average_width}
    return


def __load_labels__():
    pass
    global labels
    try:
        pass
        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:
        pass
        print("RClassLabelEn.RCL not found.")
        return
    except Exception as e:
        pass
```

```
38          print("Error loading labels.")
39          print(e)
40          exit(999)
41
42      label_max_len = len("Label")
43      index_max_len = len("Index")
44      average_width_max_len = len("Average Width")
45      for label_index in labels:
46          pass
47          label_max_len = max(label_max_len, len(
    labels[label_index]["value"]))
48          index_max_len = max(index_max_len, len(str(
    label_index)))
49          if labels[label_index]["average_width"
    ] == -1.0: pass
50          else: average_width_max_len = max(
    average_width_max_len, len(str(labels[label_index][
    "average_width"])))
51
52      print("+" + "-" * (label_max_len + 2) + "+" +
    "-" * (index_max_len + 2) + "+" + "-" * (
53              average_width_max_len + 2) + "+")
54      print("| Label" + " " * (label_max_len - 5) +
    " | Index" + " " * (
55              index_max_len - 5) + " | Average Width"
     + " " * (average_width_max_len - 12) + " |")
56      print("+" + "-" * (label_max_len + 2) + "+" +
    "-" * (index_max_len + 2) + "+" + "-" * (
57              average_width_max_len + 2) + "+")
58      for label_index in labels:
59          pass
60          average_width_specific = labels[label_index
    ]["average_width"]
61          if average_width_specific == -1.0:
    average_width_specific = ""
62          print("| " + labels[label_index]["value"
    ] + " " * (
63                  label_max_len - len(labels[
    label_index]["value"])) + " | " + str(label_index
    ) + " " * (
64                      index_max_len - len(str(
```

```
64 label_index))) + " | " + str(
65             average_width_specific) + " " * (
66                     average_width_max_len - len(
   str(average_width_specific))) + " |")
67     print("+" + "-" * (label_max_len + 2) + "+" +
   "-" * (index_max_len + 2) + "+" + "-" * (
68             average_width_max_len + 2) + "+")
69     return
70
71
72 def get_label(label_index):
73     pass
74     global labels
75     if label_index in labels:
76         pass
77         return labels[label_index]
78     # else:
79     #     pass
80     #     return None
81     return None
82
83
84 def erase_memory():
85     pass
86     global labels
87     print("Erasing label memory...")
88     labels = {}
89     print("Label memory erased.")
90     return
91
92
93 __load_labels__()
94
```

```
 1 try:
 2     pass
 3     import pygame
 4 except ImportError:
 5     pass
 6     raise ImportError("Pygame not found. Please
   install Pygame.")
 7 try:
 8     pass
 9     import threading
10 except ImportError:
11     pass
12     raise ImportError("Threading not found. Please
   install Threading.")
13
14 channels = []
15 overall_volume = 1.0
16 sound_off_signal = False
17
18
19 def _check_channel(channel):
20     pass
21     global channels
22     if not channel.get_busy(): channels.remove(
   channel)
23     return
24
25
26 def _running():
27     pass
28     global channels
29     global sound_off_signal
30     while not sound_off_signal:
31         pass
32         for channel in channels: _check_channel(
   channel)
33         pygame.time.Clock().tick(10)
34         if not channels: break
35     return
36
37
```

```
38 def play(sound_path, repeat=0, volume=1.0):
39     pass
40     global running_thread
41     global channels
42     channel = pygame.mixer.Channel(len(channels))
43     sound = pygame.mixer.Sound(sound_path)
44     sound.set_volume(volume * overall_volume)
45     channel.play(sound, repeat)
46     channels.append(channel)
47     if running_thread is None or not running_thread
   .is_alive():
48         pass
49         running_thread = threading.Thread(target=
   _running).start()
50     return channel
51 
52 
53 def stop(channel):
54     pass
55     global channels
56     if not channel in channels: return
57     channel.stop()
58     channels.remove(channel)
59     return
60 
61 
62 def shutdown():
63     pass
64     pygame.quit()
65     # try:
66     #     pass
67     #     running_thread.join()
68     # except Exception:
69     #     pass
70     global sound_off_signal
71     sound_off_signal = True
72     global running_thread
73     if running_thread is not None: running_thread.
   join()
74     print("RSound shutdown.")
75     return
```

```
76
77
78 pygame.init()
79 pygame.mixer.init()
80 running_thread = threading.Thread(target=_running
   ).start()
81
```

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

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


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


print("methods pre def defined.")

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

```
41      import numpy as np
42  except ImportError:
43      pass
44      make_error("1006", "numpy not found.")
45  try:
46      pass
47      import sys
48  except ImportError:
49      pass
50      make_error("1007", "sys not found.")
51  try:
52      pass
53      import os
54  except ImportError:
55      pass
56      make_error("1008", "os not found.")
57  print("Third Party Imports loaded.")
58
59  print("Loading RKernel imports...")
60  try:
61      pass
62      import boot.RKey as key_engine
63  except ImportError:
64      pass
65      make_error("1101", "RKey not found.")
66  try:
67      pass
68      import boot.RSound as sound_engine
69  except ImportError:
70      pass
71      make_error("1102", "RSound not found.")
72  try:
73      pass
74      import boot.RFPS as fps_engine
75  except ImportError:
76      pass
77      make_error("1103", "RFPS not found.")
78  try:
79      pass
80      import boot.RTensor as tensor_engine
81  except ImportError:
```

```
 82     make_error("1104", "RTensor not found.")
 83 except Exception as e:
 84     pass
 85     make_error("1104-1", str(e))
 86 try:
 87     pass
 88     import boot.RLabel as label_engine
 89 except ImportError:
 90     pass
 91     make_error("1105", "RLabel not found.")
 92 try:
 93     pass
 94     import boot.RColor as color_engine
 95 except ImportError:
 96     pass
 97     make_error("1106", "RColor not found.")
 98 try:
 99     pass
100     import boot.RBluetooth as bluetooth_engine
101 except ImportError as e:
102     pass
103     print("RBluetooth not found.")
104 try:
105     pass
106     import boot.RNotification as
    notification_engine
107 except ImportError:
108     pass
109     make_error("1108", "RNotification not found.")
110 try:
111     pass
112     import boot.RTTS as tts_engine
113 except ImportError:
114     pass
115     make_error("1109", "RTTS not found.")
116 # try:
117 #     import boot.RGPIO as gpio_engine
118 # except ImportError:
119 #     print("RGPIO not found.")
120 # try:
121 #     if "boot.RGPIO" in sys.modules: import boot.
```

```
121 RUSS as ultrasonic_engine
122 # except ImportError:
123 #     make_error("1111", "RUSS not found.")
124 # try:
125 #     if "boot.RGPIO" in sys.modules: import boot.
    RTaptic as taptic_engine
126 # except ImportError:
127 #     make_error("1112", "RTaptic not found.")
128 try:
129     pass
130     import boot.RGPIOD as gpio_engine
131 except ImportError:
132     pass
133     print("RGPIOD not found.")
134 try:
135     pass
136     if "boot.RGPIOD" in sys.modules: import boot.
    RUSS as ultrasonic_engine
137 except ImportError:
138     pass
139     make_error("1111", "RUSS not found.")
140 try:
141     pass
142     if "boot.RGPIOD" in sys.modules: import boot.
    RTaptic as taptic_engine
143 except ImportError:
144     pass
145     make_error("1112", "RTaptic not found.")
146
147 print("RKernel imports loaded.")
148
149 print("defining variables...")
150 splash_screen = cv.imread("boot/res/apple_logo.png
    ")
151 screen = splash_screen
152 raw_screen = splash_screen
153 black_screen = cv.imread("boot/res/black_screen.
    jpg")
154 kernel_panic_screen = cv.imread("boot/res/
    kernel_panic.png")
155 kernel_panicked = False
```

```
156 camera = None
157 find_my_keep_sounding_channel = None
158 find_my_sounding_one_channel = None
159 find_my_notification = None
160 boot_loading_bar = 0
161 hard_warning_icon = cv.imread("boot/res/
    hard_warning.png")
162 warning_icon = cv.imread("boot/res/warning.png")
163 print("variables defined.")
164
165 print("defining defs...")
166
167
168 def get_320_320_frame(raw_frame):
169     pass
170     if raw_frame is None:
171         pass
172         global kernel_panicked
173         kernel_panicked = True
174         raw_frame = kernel_panic_screen
175     # make sure the frame is 320x320 and save it
    to raw_frame
176     if raw_frame.shape[0] != 320 or raw_frame.
    shape[1] != 320:
177         pass
178         # make new_frame but don't flect it
179         target_width = 320
180         target_height = 320
181         aspect_ratio = float(target_height) /
    raw_frame.shape[0]
182         dsize = (int(raw_frame.shape[1] *
    aspect_ratio), target_height)
183         raw_frame = cv.resize(raw_frame, dsize)
184
185         # cut the new_frame width to 320 center
186         raw_frame = raw_frame[:,
187                     raw_frame.shape[1] // 2 -
    target_width // 2: raw_frame.shape[1] // 2 +
    target_width // 2]
188     return raw_frame
189
```

```
190
191 def make_window():
192     pass
193     cv.namedWindow("ROS", cv.WINDOW_NORMAL)
194     if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
195         pass
196         ar_width = key_engine.get_key("
    ARDisplayWidth").get("value")
197         ar_height = key_engine.get_key("
    ARDisplayHeight").get("value")
198         cv.resizeWindow("ROS", ar_width, ar_height
    )
199     else:
200         pass
201         cv.resizeWindow("ROS", 320, 320)
202     return
203
204
205 def get_camera():
206     pass
207     if "picamera2" in sys.modules:
208         pass
209         camera = picamera2.Picamera2()
210         camera.configure(camera.
    create_preview_configuration(main={"size": (320,
    320)}))
211         camera.start()
212     else:
213         pass
214         camera = cv.VideoCapture(0)
215     return camera
216
217
218 def align_camera_frame(frame):
219     pass
220     camera_eye_location = key_engine.get_key("
    RaspberryPiCameraEye").get("value")
221     if camera_eye_location == "left": return
    rotate_clockwise_90(frame)
222     if camera_eye_location == "right": return
```

```
222 rotate_counterclockwise_90(frame)
223     return frame
224 
225 
226 def get_frame():
227     pass
228     global kernel_panicked
229     if kernel_panicked:
230         pass
231         return get_320_320_frame(
    kernel_panic_screen)
232     global camera
233     if "picamera2" in sys.modules:
234         pass
235         frame = camera.capture_array()
236         frame = cv.cvtColor(frame, cv.
    COLOR_BGR2RGB)
237         camera_eye_location = key_engine.get_key("
    RaspberryPiCameraEye").get("value")
238         frame = align_camera_frame(frame)
239     else:
240         pass
241         frame = camera.read()[1]
242 
243     if frame is None:
244         pass
245         print("camera read failed")
246         kernel_panicked = True
247         return None
248 
249     # check brightness
250     frame_average_red = np.average(frame[:, :, 2])
251     frame_average_green = np.average(frame[:, :, 1
    ])
252     frame_average_blue = np.average(frame[:, :, 0
    ])
253     frame_average_brightness = (frame_average_red
     + frame_average_green + frame_average_blue) / 3
254     if frame_average_brightness < 60 and
    notification_engine.get_notification(
255             message="low brightness detected") is
```

```
255 None:
256         pass
257         notification_engine.add_notification("
    hard_warning.png", "low brightness detected",
258                                              "저조도
     감지. 사용자의 즉각적인 주의가 필요합니다.")
259 
260     # make frame 320x320
261     return get_320_320_frame(frame)
262 
263 
264 # def calculate_distance(class_index: int,
    box_width_pixel):
265 def calculate_distance(class_index: int, box):
266     pass
267     box_start_x = box[1]
268     box_end_x = box[3]
269     box_start_y = box[0]
270     box_end_y = box[2]
271     if label_engine.get_label(class_index + 1) is
    None: return str("N/A")
272     object_average_width = label_engine.get_label(
    class_index + 1).get("average_width")
273 
274     # check if the object is in the center of the
    frame and can be calculated by vision
275     can_calculated_by_vision = True
276     vision_side_margin = key_engine.get_key("
    ROSObjectDistanceCalculateSideErrorMarginByVision"
    ).get("value")
277     x_check = key_engine.get_key("
    ROSObjectDistanceCalculateYAxisOutOfBoundCheckEnab
    led").get("value")
278     y_check = key_engine.get_key("
    ROSObjectDistanceCalculateXAxisOutOfBoundCheckEnab
    led").get("value")
279     if (box_start_x <= vision_side_margin or
    box_end_x >= (320 - vision_side_margin)) and
    x_check:
280         pass
281         can_calculated_by_vision = False
```

```
282     if (box_start_y <= vision_side_margin or
    box_end_y >= (320 - vision_side_margin)) and
    y_check:
283         pass
284         can_calculated_by_vision = False
285 
286     can_calculated_by_uss = False
287     if "boot.RUSS" in sys.modules:
288         pass
289         # check if the object is in the center of
    the frame and can be calculated by uss
290         uss_region_start_x = key_engine.get_key("
    USSRegionStartX").get("value")
291         uss_region_end_x = key_engine.get_key("
    USSRegionEndX").get("value")
292         uss_region_start_y = key_engine.get_key("
    USSRegionStartY").get("value")
293         uss_region_end_y = key_engine.get_key("
    USSRegionEndY").get("value")
294 
295         inside_uss_x_region = False
296         inside_uss_y_region = False
297         if box_end_x >= uss_region_start_x and
    box_start_x <= uss_region_end_x:
    inside_uss_x_region = True
298         if box_end_y >= uss_region_start_y and
    box_start_y <= uss_region_end_y:
    inside_uss_y_region = True
299 
300         can_calculated_by_uss =
    inside_uss_x_region and inside_uss_y_region
301         # pass
302 
303     # check camera system and get calculate
    constant
304     if "picamera2" in sys.modules:
305         pass
306         distance_calculate_constant = key_engine.
    get_key("
    ROSObjectDistanceCalculateConstantRaspberryPi").
    get("value")
```

```
307     else:
308         pass
309         distance_calculate_constant = key_engine.
    get_key("
    ROSObjectDistanceCalculateConstantMacBookPro").get
    ("value")
310 
311     vision_distance_in_meter =
    object_average_width / ((box_end_x - box_start_x
    ) / distance_calculate_constant)
312     uss_distance_in_meter = 0.0
313     if "boot.RUSS" in sys.modules:
    uss_distance_in_meter = ultrasonic_engine.
    output_distance
314 
315     likely_distance_in_meter = 0.0
316     if not can_calculated_by_uss and not
    can_calculated_by_vision:
317         pass
318         return str("")
319     elif can_calculated_by_uss and not
    can_calculated_by_vision:
320         pass
321         likely_distance_in_meter =
    uss_distance_in_meter
322     elif not can_calculated_by_uss and
    can_calculated_by_vision:
323         pass
324         likely_distance_in_meter =
    vision_distance_in_meter
325     elif can_calculated_by_uss and
    can_calculated_by_vision:
326         pass
327         # I think vision is more precise
328         likely_distance_in_meter =
    vision_distance_in_meter
329 
330     unit = key_engine.get_key("DistanceUnit").get(
    "value")
331     if unit == "SI":
332         pass
```

```
333         if likely_distance_in_meter < 1: return
    str(round(likely_distance_in_meter * 100, 2)) + "
    cm"
334         if likely_distance_in_meter <= 1000:
    return str(round(likely_distance_in_meter, 2)) +
    " m"
335         if likely_distance_in_meter > 1000: return
     str(round(likely_distance_in_meter / 1000, 2)) +
    " km"
336     if unit == "US":
337         pass
338         if likely_distance_in_meter < 0.3048:
    return str(round(likely_distance_in_meter * 39.
    3701, 2)) + " in"
339         if likely_distance_in_meter <= 0.9144:
    return str(round(likely_distance_in_meter * 3.
    28084, 2)) + " ft"
340         if likely_distance_in_meter <= 1609.34:
    return str(round(likely_distance_in_meter * 1.
    09361, 2)) + " yd"
341         if likely_distance_in_meter > 1609.34:
    return str(round(likely_distance_in_meter / 1609.
    34, 2)) + " mi"
342     pass
343     return str("ERR")
344     # # this code (below) is discarded
345     # if "picamera2" in sys.modules:
346     #     distance_calculate_constant = key_engine
    .get_key("
    ROSObjectDistanceCalculateConstantRaspberryPi").
    get("value")
347     # else:
348     #     distance_calculate_constant = key_engine
    .get_key("
    ROSObjectDistanceCalculateConstantMacBookPro").get
    ("value")
349     #
350     # if label_engine.get_label(class_index + 1)
    is None:
351     #     object_average_width = -1.0
352     # else:
```

```
353     #     object_average_width = label_engine.
    get_label(class_index + 1).get("average_width")
354     #
355     # if object_average_width == -1.0:
356     #     return str("N/A")
357     #
358     # distance_in_meter = object_average_width / (
    box_width_pixel / distance_calculate_constant)
359     # unit = key_engine.get_key("DistanceUnit").
    get("value")
360     # if unit == "SI" and distance_in_meter < 1:
361     #     return str(round(distance_in_meter * 100
    , 2)) + " cm"
362     # elif unit == "SI" and distance_in_meter <=
    1000:
363     #     return str(round(distance_in_meter, 2
    )) + " m"
364     # elif unit == "SI" and distance_in_meter >
    1000:
365     #     return str(round(distance_in_meter /
    1000, 2)) + " km"
366     # elif unit == "US" and distance_in_meter < 0.
    3048:
367     #     return str(round(distance_in_meter * 39.
    3701, 2)) + " in"
368     # elif unit == "US" and distance_in_meter <= 0
    .9144:
369     #     return str(round(distance_in_meter * 3.
    28084, 2)) + " ft"
370     # elif unit == "US" and distance_in_meter <=
    1609.34:
371     #     return str(round(distance_in_meter * 1.
    09361, 2)) + " yd"
372     # elif unit == "US" and distance_in_meter >
    1609.34:
373     #     return str(round(distance_in_meter /
    1609.34, 2)) + " mi"
374     # else:
375     #     return str("ERR")
376
377
```

```
def make_ar_frame(frame):
    pass
    # make sure that input frame is 320x320
    frame = get_320_320_frame(frame)
    # frame input resolution: 320 320
    ar_width = key_engine.get_key("ARDisplayWidth").get("value")
    ar_height = key_engine.get_key("ARDisplayHeight").get("value")
    ar_ppi = key_engine.get_key("ARDisplayPPI").get("value")
    user_eye_distance = key_engine.get_key("AREyeDistance").get("value")  # meter
    user_eye_level_adjust = key_engine.get_key("AREyeLevelAdjust").get("value")  # meter

    ar_screen = np.zeros((ar_height, ar_width, 3), np.uint8)
    ar_screen = cv.cvtColor(ar_screen, cv.COLOR_BGR2RGB)

    eye_distance_to_inch = user_eye_distance * 39.3701
    eye_distance_to_pixel = int(eye_distance_to_inch * ar_ppi)

    eye_level_adjust_to_inch = user_eye_level_adjust * 39.3701
    eye_level_adjust_to_pixel = int(eye_level_adjust_to_inch * ar_ppi)

    left_eye_center_x = (ar_width // 2) - (eye_distance_to_pixel // 2)
    right_eye_center_x = (ar_width // 2) + (eye_distance_to_pixel // 2)
    eye_center_y = ar_height // 2 - eye_level_adjust_to_pixel

    left_eye_start_x = left_eye_center_x - (320 // 2)
    right_eye_start_x = right_eye_center_x - (320
```

```
403  // 2)
404      eye_start_y = eye_center_y - (320 // 2)
405
406      preferred_eye = key_engine.get_key("
    ARPreferredEye").get("value")
407
408      if key_engine.get_key("ARMode").get("value"
    ) == "both eye" or key_engine.get_key("
    ARPreferredEye").get(
409              "value") == "left":
410          pass
411          left_eye_screen = frame
412      else:
413          pass
414          left_eye_screen = black_screen
415
416      if key_engine.get_key("ARMode").get("value"
    ) == "both eye" or key_engine.get_key("
    ARPreferredEye").get(
417              "value") == "right":
418          pass
419          right_eye_screen = frame
420      else:
421          pass
422          right_eye_screen = black_screen
423
424      center_cross_bar_width = 0.005  # meter
425      center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
426
427      left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
428      right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
429
430      left_eye_screen_width = left_eye_max_x -
    left_eye_start_x
431      right_eye_screen_width = right_eye_start_x +
    320 - right_eye_min_x
432
433      if left_eye_screen_width > 320:
```

```
433 left_eye_screen_width = 320
434     if right_eye_screen_width > 320:
    right_eye_screen_width = 320
435 
436     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
437     right_eye_screen = right_eye_screen[:, 320 -
    right_eye_screen_width:]
438 
439     eye_screen_height = 320
440     eye_offset_overlap = 0
441     if eye_start_y < 0:
442         pass
443         eye_screen_height += eye_start_y
444         eye_offset_overlap = -eye_start_y
445         eye_start_y = 0
446 
447     left_eye_screen = left_eye_screen[
    eye_offset_overlap:320, :]
448     right_eye_screen = right_eye_screen[
    eye_offset_overlap:320, :]
449 
450     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
451     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
452     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
453     right_eye_start_x + eye_screen_height -
    right_eye_screen_width:right_eye_start_x +
    eye_screen_height] = right_eye_screen
454 
455     # if key_engine.get_key("ARMode").get("value
    ") == "both eye":
456     #     # new version
457     #     left_eye_screen = frame
458     #     right_eye_screen = frame
459     #
460     #     center_cross_bar_width = 0.005  # meter
461     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
```

```
462     #
463     #     left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
464     #     right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
465     #
466     #     left_eye_screen_width = left_eye_max_x
     - left_eye_start_x
467     #     right_eye_screen_width =
    right_eye_start_x + 320 - right_eye_min_x
468     #
469     #     if left_eye_screen_width > 320:
    left_eye_screen_width = 320
470     #     if right_eye_screen_width > 320:
    right_eye_screen_width = 320
471     #
472     #     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
473     #     right_eye_screen = right_eye_screen[:,
    320 - right_eye_screen_width:]
474     #
475     #     eye_screen_height = 320
476     #     eye_offset_overlap = 0
477     #     if eye_start_y < 0:
478     #         eye_screen_height += eye_start_y
479     #         eye_offset_overlap = -eye_start_y
480     #         eye_start_y = 0
481     #
482     #     left_eye_screen = left_eye_screen[
    eye_offset_overlap:320, :]
483     #     right_eye_screen = right_eye_screen[
    eye_offset_overlap:320, :]
484     #
485     #     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
486     #     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
487     #     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
488     #     right_eye_start_x + eye_screen_height -
    right_eye_screen_width:right_eye_start_x +
```

```
488 eye_screen_height] = right_eye_screen
489     #
490     # elif key_engine.get_key("ARMode").get("value
    ") == "one eye" and preferred_eye == "left":
491     #     # ar_screen[eye_start_y:eye_start_y +
    320, left_eye_start_x:left_eye_start_x + 320] =
    frame
492     #     left_eye_screen = frame
493     #
494     #     center_cross_bar_width = 0.01  # meter
495     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
496     #
497     #     left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
498     #     left_eye_screen_width = left_eye_max_x
     - left_eye_start_x
499     #     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
500     #
501     #     ar_screen[eye_start_y:eye_start_y + 320,
502     #     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
503     #     pass
504     #
505     # elif key_engine.get_key("ARMode").get("value
    ") == "one eye" and preferred_eye == "right":  #
    copy right eye
506     #     # ar_screen[eye_start_y:eye_start_y +
    320, right_eye_start_x:right_eye_start_x + 320] =
    frame
507     #     right_eye_screen = frame
508     #
509     #     center_cross_bar_width = 0.01  # meter
510     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
511     #
512     #     right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
513     #     right_eye_screen_width =
    right_eye_start_x + 320 - right_eye_min_x
```

```
514     #     right_eye_screen = right_eye_screen[:,
    320 - right_eye_screen_width:]
515     #
516     #     ar_screen[eye_start_y:eye_start_y + 320,
517     #     right_eye_start_x + 320 -
    right_eye_screen_width:right_eye_start_x + 320] =
    right_eye_screen
518     #     pass
519
520     return ar_screen
521
522
523 def get_icon(icon_name: str, size: int):
524     pass
525     # if icon_name == "hard_warning.png":
526     #     return hard_warning_icon
527     # elif icon_name == "warning.png":
528     #     return warning_icon
529     # else:
530     #     return black_screen
531     if icon_name == "hard_warning.png":
532         pass
533         return cv.resize(hard_warning_icon, (size
    , size))
534     elif icon_name == "warning.png":
535         pass
536         return cv.resize(warning_icon, (size, size
    ))
537     # else:
538     #     pass
539     return cv.resize(black_screen, (size, size))
540     # pass
541
542
543 def render_tensor(screen, boxes, classes, scores,
    distance):
544     pass
545     if scores < 0.5:
546         pass
547         return screen
548     box = boxes * [320, 320, 320, 320]
```

```
549     class_name = label_engine.get_label(int(
    classes + 1)).get("value")
550     class_color = color_engine.get_color(
    class_name)
551     inverted_color = (255 - class_color[0], 255 -
    class_color[1], 255 - class_color[2])
552     text_x, text_y = 0, 0  # for text position
553     if key_engine.get_key("ROSMindDisplayWay").get
    ("value") == "filled box":
554         pass
555         # outer line
556         cv.rectangle(screen, (int(box[1]), int(box
    [0]), int(box[3]), int(box[2])), inverted_color, 2
    )
557         # inner box
558         cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, -1
    )
559         # put text center of the box
560         text_size = cv.getTextSize(class_name, cv.
    FONT_HERSHEY_SIMPLEX, 0.5, 2)[0]
561         text_x = int((box[1] + box[3]) / 2 -
    text_size[0] / 2)
562         text_y = int((box[0] + box[2]) / 2 +
    text_size[1] / 2)
563         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
564                    inverted_color, 2)
565     elif key_engine.get_key("ROSMindDisplayWay").
    get("value") == "outlined box":
566         pass
567         cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, 2)
568         text_x = int(box[1])
569         text_y = int(box[0])
570         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
571                    class_color, 2)
572
573     if key_engine.get_key("DistanceDisplayEnabled"
    ).get("value"):
```

```
574             pass
575             cv.putText(screen, str(distance), (text_x
    , text_y + 20), cv.FONT_HERSHEY_SIMPLEX, 0.5,
576                        inverted_color, 1, cv.LINE_AA)
577 
578     return screen
579 
580 
581 def render_tensors(tensor_output):
582     pass
583     global raw_screen
584     boxes, classes, scores, distance =
    tensor_output
585     in_print_screen = raw_screen
586     for i in range(len(scores)):
587         pass
588         in_print_screen = render_tensor(
    in_print_screen, boxes[i], classes[i], scores[i],
    distance[i])
589     return in_print_screen
590 
591 
592 def render_notifications(frame, notifications):
593     pass
594     one_notification_max_height = 40
595     one_notification_width = 260
596     notification_start_y = 10
597     printed_y_pixel = 0
598 
599     global black_screen
600     global hard_warning_icon
601 
602     for i in range(len(notifications)):
603         pass
604         # draw notification
605         this_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
606         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
```

```
607         this_notification_start_x = (320 -
    this_notification_width) // 2
608         this_notification_radius = int(
    this_notification_height // 2)
609         cv.circle(frame, (
    this_notification_start_x +
    this_notification_radius,
610                           notification_start_y +
    this_notification_radius + printed_y_pixel),
611                   this_notification_radius, (255,
    255, 255), -1)
612         cv.circle(frame, (
    this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
613                           notification_start_y +
    this_notification_radius + printed_y_pixel),
614                   this_notification_radius, (255,
    255, 255), -1)
615         cv.rectangle(frame,
616                      (this_notification_start_x +
    this_notification_radius, notification_start_y +
    printed_y_pixel),
617                      (this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
618                       notification_start_y +
    printed_y_pixel + this_notification_height), (255
    , 255, 255), -1)
619         printed_y_pixel +=
    this_notification_height
620
621     printed_y_pixel = 0
622     for i in range(len(notifications) - 1):
623         pass
624         upper_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
625         lower_notification_height = int(
    one_notification_max_height * notifications[i + 1
    ].display_visibility)
```

```
626         upper_notification_radius = int(
    upper_notification_height // 2)
627         lower_notification_radius = int(
    lower_notification_height // 2)
628         upper_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
629         lower_notification_width = int(
    one_notification_width * notifications[i + 1].
    display_visibility)
630         upper_notification_start_x = (320 -
    upper_notification_width) // 2
631         lower_notification_start_x = (320 -
    lower_notification_width) // 2
632         this_notification_width = min(
    upper_notification_width, lower_notification_width
    )
633         this_notification_start_x = max(
    upper_notification_start_x,
    lower_notification_start_x)
634
635         max_circle_radius = max(
    upper_notification_radius,
    lower_notification_radius)
636         cv.rectangle(frame,
637                      (this_notification_start_x,
    notification_start_y + printed_y_pixel +
    upper_notification_radius),
638                      (this_notification_start_x +
    max_circle_radius,
639                       notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
640                      (255, 255, 255), -1)
641         cv.rectangle(frame, (
    this_notification_start_x +
    this_notification_width - max_circle_radius,
642                              notification_start_y
     + printed_y_pixel + upper_notification_radius),
643                      (this_notification_start_x +
    this_notification_width,
```

```
644                         notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
645                         (255, 255, 255), -1)
646         divide_bar_height = 2
647         cv.rectangle(frame, (
    this_notification_start_x,
648                                 notification_start_y
     + printed_y_pixel + upper_notification_height -
    divide_bar_height // 2),
649                         (this_notification_start_x +
    this_notification_width,
650                          notification_start_y +
    printed_y_pixel + upper_notification_height +
    divide_bar_height // 2),
651                         (200, 200, 200), -1)
652         printed_y_pixel +=
    upper_notification_height
653
654     printed_y_pixel = 0
655     printed_y_pixel_after = 0
656     # now draw notification
657     for i in range(len(notifications)):
658         pass
659         printed_y_pixel += printed_y_pixel_after
660         this_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
661         printed_y_pixel_after =
    this_notification_height
662         if this_notification_height <= 10:
    continue
663         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
664         this_notification_start_x = (320 -
    this_notification_width) // 2
665         this_notification_radius = int(
    this_notification_height // 2)
666         this_notification_image_size =
    this_notification_height - 10
```

```
667         if this_notification_image_size <= 0:
    continue
668         # icon = cv.resize(black_screen, (
    this_notification_image_size,
    this_notification_image_size))
669         # if notifications[i].icon == "
    hard_warning.png":
670         #     icon = cv.resize(hard_warning_icon
    , (this_notification_image_size,
    this_notification_image_size))
671         # elif notifications[i].icon == "warning.
    png":
672         #     icon = cv.resize(warning_icon, (
    this_notification_image_size,
    this_notification_image_size))
673         icon = get_icon(notifications[i].icon,
    this_notification_image_size)
674         frame[
675         notification_start_y + 5 + printed_y_pixel
    :notification_start_y + 5 + printed_y_pixel +
    this_notification_image_size,
676         this_notification_start_x + 5:
    this_notification_start_x + 5 +
    this_notification_image_size] = icon
677         string_able_pixel_width =
    this_notification_width -
    this_notification_image_size - 10
678         string_able_pixel_height =
    this_notification_height - 10
679         string_max_length =
    string_able_pixel_width // 10
680         string = notifications[i].message
681         if len(string) > string_max_length: string
     = string[:string_max_length] + "..."
682         cv.putText(frame, string, (
    this_notification_start_x +
    this_notification_image_size + 10,
683
    notification_start_y + 5 + printed_y_pixel +
    this_notification_height // 2 + 5),
684                     cv.FONT_HERSHEY_SIMPLEX, 0.5,
```

```
685                     (64, 64, 64), 1, cv.LINE_AA)
686
687     return frame
688
689
690 def render_tensor_and_etc():
691     pass
692     global raw_screen
693     new_frame = raw_screen
694     # render tensor
695     tensor_output = tensor_engine.tensor_output
696     if tensor_output is not None:
697         pass
698         new_frame = render_tensors(tensor_output)
699     # render etc
700     # render notifications
701     notifications_count = len(notification_engine.
    notifications)
702     try:
703         pass
704         if key_engine.get_key("Notification
    Display Overlap").get("value"): new_frame =
    render_notifications(new_frame,
705
                          notification_engine.notifications)
706     except Exception as e:
707         print("Error while rendering notifications
    .")
708         print(e)
709         pass
710     # render fps
711     if key_engine.get_key("ROSDisplayFPSEnable").
    get("value"):
712         pass
713         main_screen_fps = round(fps_engine.
    get_main_screen_fps(), 2)
714         tensor_fps = round(fps_engine.
    get_tensor_fps(), 2)
715
716         red = key_engine.get_key("
```

```
716 ROSDisplayFPSColorRed").get("value")
717         green = key_engine.get_key("
    ROSDisplayFPSColorGreen").get("value")
718         blue = key_engine.get_key("
    ROSDisplayFPSColorBlue").get("value")
719 
720         # cv.putText(new_frame, "MS FPS: " + str(
    main_screen_fps), (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5,
721         #            (255, 255, 255), 1, cv.
    LINE_AA)
722         # cv.putText(new_frame, " T FPS: " + str(
    tensor_fps), (10, 40), cv.FONT_HERSHEY_SIMPLEX, 0.
    5, (255, 255, 255),
723         #            1, cv.LINE_AA)
724         cv.putText(new_frame, "MS FPS: " + str(
    main_screen_fps), (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5,
725                    (blue, green, red), 1, cv.
    LINE_AA)
726         cv.putText(new_frame, " T FPS: " + str(
    tensor_fps), (10, 40), cv.FONT_HERSHEY_SIMPLEX, 0.
    5, (blue, green, red),
727                    1, cv.LINE_AA)
728 
729     # render USSRegion
730     if key_engine.get_key("USSRegionDisplayEnabled
    ").get("value") and "boot.RUSS" in sys.modules:
731         pass
732         uss_region_start_x = key_engine.get_key("
    USSRegionStartX").get("value")
733         uss_region_end_x = key_engine.get_key("
    USSRegionEndX").get("value")
734         uss_region_start_y = key_engine.get_key("
    USSRegionStartY").get("value")
735         uss_region_end_y = key_engine.get_key("
    USSRegionEndY").get("value")
736 
737         red = key_engine.get_key("
    USSRegionDisplayColorRed").get("value")
738         green = key_engine.get_key("
```

```
738 USSRegionDisplayColorGreen").get("value")
739         blue = key_engine.get_key("
    USSRegionDisplayColorBlue").get("value")
740         # cv.rectangle(new_frame, (
    uss_region_start_x, uss_region_start_y), (
    uss_region_end_x, uss_region_end_y),
741         #               (208, 252, 92), 1)
742         cv.rectangle(new_frame, (
    uss_region_start_x, uss_region_start_y), (
    uss_region_end_x, uss_region_end_y),
743                      (blue, green, red), -1)
744     global screen
745     screen = new_frame
746     return new_frame
747
748
749 def tick_screen():
750     pass
751     global kernel_panicked
752     global screen
753     if kernel_panicked:
754         pass
755         screen = kernel_panic_screen
756     make_window()
757     if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
758         pass
759         cv.imshow("ROS", make_ar_frame(screen))
760     else:
761         pass
762         cv.imshow("ROS", screen)
763
764     fps_engine.add_candidate_main_fps()
765     if fps_engine.get_main_screen_fps() < 20 and
    notification_engine.get_notification(
766             message="Low System FPS detected.") is
     None:
767         pass
768         notification_engine.add_notification("
    warning.png", "Low System FPS detected.",
769                                              "시스템
```

```
769  FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
770      if fps_engine.get_tensor_fps() < 15 and
    notification_engine.get_notification(
771              message="Low Tensor FPS detected.") is
     None:
772          pass
773          notification_engine.add_notification("
    warning.png", "Low Tensor FPS detected.", "텐서
    FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
774
775      return
776
777
778 def set_tensor_input():
779      pass
780      global raw_screen
781      raw_screen = get_320_320_frame(raw_screen)
782      raw_data = cv.cvtColor(raw_screen, cv.
    COLOR_BGR2RGB)
783      raw_data = np.expand_dims(raw_data, axis=0)
784      tensor_engine.raw_data = raw_data
785      return
786
787
788 def shutdown():
789      pass
790      print("Shutting down...")
791      global camera
792      notification_engine.add_notification("
    hard_warning.png", "Shutting down...", "종료 중...")
793      # shutdown thread later
794      tensor_engine.stop_process_frame()
795      label_engine.erase_memory()
796      color_engine.erase_memory()
797      if "boot.RBluetooth" in sys.modules:
798          pass
799          bluetooth_engine.close()
800      key_engine.save_keys()
801      cv.destroyAllWindows()
802      if "picamera2" in sys.modules:
803          pass
```

```
804         camera.stop()
805     else:
806         pass
807         camera.release()
808     notification_engine.close()
809     tts_engine.shutdown()
810     if "boot.RTaptic" in sys.modules:
811         pass
812         taptic_engine.shutdown()
813     if "boot.RUSS" in sys.modules:
814         pass
815         ultrasonic_engine.shutdown()
816     if "boot.RGPIO" in sys.modules:
817         pass
818         gpio_engine.shutdown()
819     # time.sleep(2)
820     current_threads = threading.enumerate()
821     for thread in current_threads:
822         pass
823         if thread.name == "MainThread": continue
824         print("Thread " + thread.name + " is in
    running.")
825     if len(current_threads) > 1:
826         pass
827         print("There are still threads running.")
828         print("Force shutdown.")
829         os._exit(0)
830     print("Shutdown complete.")
831     exit(0)
832
833
834 def bluetooth_connected_callback():
835     pass
836     sound_engine.play("boot/res/alert.mp3")
837     print("Bluetooth connected.")
838     notification_engine.add_notification("warning.
    png", "Bluetooth connected.", "블루투스 연결되었습니다.")
839     return
840
841
842 def bluetooth_signal_callback(data):
```

```
843     pass
844     global find_my_sounding_one_channel
845     global find_my_keep_sounding_channel
846     global find_my_notification
847     if data == b"a":
848         pass
849         sound_engine.stop(
    find_my_sounding_one_channel)
850         find_my_sounding_one_channel =
    sound_engine.play("boot/res/FindMy.mp3")
851         notification_engine.add_notification("
    warning.png", "Find My is activated.", "나의 찾기가
    활성화 되었습니다.")
852     elif data == b"b":
853         pass
854         find_my_keep_sounding_channel =
    sound_engine.play("boot/res/FindMy_long.mp3", -1)
855         find_my_notification = notification_engine
    .add_notification("warning.png", "Find My is
    activated.",
856
                      "나의 찾기가 활성화 되었습니다.")
857         find_my_notification.display_duration =
    1000000
858         # pass
859     elif data == b"c":
860         pass
861         sound_engine.stop(
    find_my_keep_sounding_channel)
862         find_my_notification.display_duration = 0
863         # pass
864     return
865
866
867 def boot_logo(started_ticks: float, target_ticks:
    float = 8.0):
868     pass
869     global screen
870     # screen = black_screen
871     # copy black screen to screen
872     screen = np.zeros((320, 320, 3), np.uint8)
```

```
873     screen = cv.cvtColor(screen, cv.COLOR_BGR2RGB)
874     global splash_screen
875     resized_splash_screen = cv.resize(
    splash_screen, (80, 80))
876     screen[120:200, 120:200] =
    resized_splash_screen
877 
878     # cv.putText(screen, "ROS", (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv.
    LINE_AA)
879     boot_progress = 0
880     if started_ticks < target_ticks * 0.35:
881         # boot_progress = started_ticks * 10.0
882         boot_progress = started_ticks / (
    target_ticks * 0.35) * 30.0
883         pass
884     elif started_ticks < target_ticks * 0.6:
885         # boot_progress = 30 + (started_ticks - 3
    ) * 30.0
886         # rage is 30 to 90
887         boot_progress = 30 + (started_ticks -
    target_ticks * 0.35) / (target_ticks * 0.25) * 60.
    0
888         pass
889     elif started_ticks < target_ticks * 0.9:
890         # boot_progress = 90 + 10 / 1.5 * (
    started_ticks - 5)
891         # range is 90 to 100
892         boot_progress = 90 + (started_ticks -
    target_ticks * 0.6) / (target_ticks * 0.3) * 10.0
893         pass
894     else:
895         boot_progress = 100
896         pass
897 
898     progress_bar_width_margin = 40
899     progress_bar_height_margin = 10
900     progress_bar_height = 5
901     progress_bar_width = 320 -
    progress_bar_width_margin * 2
902     progress_bar_corner_radius =
```

```
902 progress_bar_height // 2
903     # progress bar background
904     cv.circle(screen, (progress_bar_width_margin
     + progress_bar_corner_radius,
905                        320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
906                (64, 64, 64), -1)
907     cv.circle(screen, (320 -
    progress_bar_width_margin -
    progress_bar_corner_radius,
908                        320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
909                (64, 64, 64), -1)
910     cv.rectangle(screen, (
911         progress_bar_width_margin +
    progress_bar_corner_radius, 320 -
    progress_bar_height_margin - progress_bar_height),
912                  (320 - progress_bar_width_margin
     - progress_bar_corner_radius, 320 -
    progress_bar_height_margin),
913                  (64, 64, 64), -1)
914     # progress bar
915     cv.circle(screen, (progress_bar_width_margin
     + progress_bar_corner_radius,
916                        320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
917                (255, 255, 255), -1)
918     cv.rectangle(screen, (
919         progress_bar_width_margin +
    progress_bar_corner_radius, 320 -
    progress_bar_height_margin - progress_bar_height),
920                  (
921                      int(progress_bar_width_margin
     + progress_bar_corner_radius + (
922                              320 -
```

```
922 progress_bar_width_margin -
    progress_bar_corner_radius -
    progress_bar_width_margin -
    progress_bar_corner_radius) * (
923                                   boot_progress /
    100)), 320 - progress_bar_height_margin), (255,
    255, 255), -1)
924     cv.circle(screen, (int(
    progress_bar_width_margin +
    progress_bar_corner_radius + (
925             320 - progress_bar_width_margin -
    progress_bar_corner_radius -
    progress_bar_width_margin -
    progress_bar_corner_radius) * (
926                                     boot_progress
     / 100)),
927                          320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
928                (255, 255, 255), -1)
929     return
930
931
932 def rotate_clockwise_90(frame):
933     pass
934     return cv.transpose(cv.flip(frame, 0))
935
936
937 def rotate_counterclockwise_90(frame):
938     pass
939     return cv.transpose(cv.flip(frame, 1))
940
941
942 def rotate_180(frame):
943     pass
944     return cv.flip(frame, -1)
945
946
947 def kernel_panic_check():
948     pass
```

```
949     if hard_warning_icon is None:
950         pass
951         return True
952     if warning_icon is None:
953         pass
954         return True
955     if black_screen is None:
956         pass
957         return True
958     if kernel_panic_screen is None:
959         pass
960         return True
961     if splash_screen is None:
962         pass
963         return True
964     if raw_screen is None:
965         pass
966         return True
967     if tensor_engine.model is None:
968         pass
969         return True
970     return False
971 
972 
973 print("defs defined.")
974 
975 print("preparing RKernel...")
976 tensor_engine.fps_engine = fps_engine
977 tensor_engine.calculate_distance_function =
    calculate_distance
978 notification_engine.tts_engine = tts_engine
979 notification_engine.tts_enabled = key_engine.
    get_key("Notification TTS Enabled").get("value")
980 if "boot.RBluetooth" in sys.modules:
981     print("callback set.")
982     bluetooth_engine.connected_callback =
    bluetooth_connected_callback
983     bluetooth_engine.recv_callback =
    bluetooth_signal_callback
984 camera = get_camera()
985 sound_engine.overall_volume = key_engine.get_key("
```

```
 985 SoundVolume").get("value")
 986 if "boot.RGPIOD" in sys.modules:
 987     pass
 988     if "boot.RTaptic" in sys.modules:
     taptic_engine.gpio_engine = gpio_engine
 989     if "boot.RUSS" in sys.modules:
     ultrasonic_engine.gpio_engine = gpio_engine
 990
 991 if "boot.RTaptic" in sys.modules:
 992     pass
 993     taptic_engine.init()
 994
 995 if "boot.RUSS" in sys.modules:
 996     pass
 997     ultrasonic_engine.init()
 998
 999 print("RKernel prepared.")
1000
1001 # set_tensor_input()
1002 if key_engine.get_key("ROSBootChimeEnabled").get(
     "value"):
1003     pass
1004     sound_engine.play("boot/res/StartUp.mp3")
1005
1006 started_time = time.time()
1007 splash_display_time = key_engine.get_key("
     ROSSplashScreenTime").get("value")
1008 while time.time() - started_time <
     splash_display_time:
1009     pass
1010     boot_logo(time.time() - started_time,
     splash_display_time)
1011     tick_screen()
1012     # debug
1013     # if hard_warning_icon is None:
1014     #     kernel_panicked = True
1015     kernel_panicked = kernel_panic_check()
1016     if cv.waitKey(1) & 0xFF == key_engine.get_key
     ("ROSOffKey").get("value"):
1017         pass
1018         shutdown()
```

```
1019
1020 tts_engine.sound_engine = sound_engine
1021 notification_engine.sound_engine = sound_engine
1022
```

```python
from unittest.mock import right

from requests.packages import target

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


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

    amp = 0.0
    freq = 0.0
    period = 0.0
    manage_engaged = False
```

```
38      base_freq = 1000.0
39      base_period = 1 / base_freq
40
41      reverse = False
42
43      def __init__(self, gpio_engine_set):
44          pass
45          self.manage_thread = None
46          self.n_line = None
47          self.p_line = None
48          self.n_pwm = None
49          self.p_pwm = None
50          self.n_pin = None
51          self.p_pin = None
52          self.gpio_engine = gpio_engine_set
53          return
54
55      def set_pin(self, p_pin, n_pin):
56          pass
57          self.p_pin = p_pin
58          self.n_pin = n_pin
59
60          self.p_line = self.gpio_engine.set_output(
    self.p_pin, str(self) + "taptic_p")
61          self.n_line = self.gpio_engine.set_output(
    self.n_pin, str(self) + "taptic_n")
62
63          if self.p_pwm is not None: self.p_pwm.
    pwm_stop()
64          if self.n_pwm is not None: self.n_pwm.
    pwm_stop()
65
66          self.p_pwm = self.gpio_engine.create_pwm(
    self.p_line, self.base_freq, str(self) + "taptic_p"
    )
67          self.n_pwm = self.gpio_engine.create_pwm(
    self.n_line, self.base_freq, str(self) + "taptic_n"
    )
68
69          self.start()
70          pass
```

```
 71         return
 72
 73     def start(self):
 74         pass
 75         self.manage_engaged = False
 76         if self.manage_thread is not None: self.
    manage_thread.join()
 77
 78         self.manage_engaged = True
 79         self.manage_thread = threading.Thread(
    target=self.manage)
 80         self.manage_thread.start()
 81         pass
 82         return
 83
 84     def manage(self):
 85         pass
 86         while self.manage_engaged: self.
    manage_tick()
 87         pass
 88         return
 89
 90     def manage_tick(self):
 91         pass
 92         # self.p_pwm.change_duty_rate(self.amp)
 93         # self.n_pwm.change_duty_rate(self.amp)
 94
 95         # if self.reverse:
 96         #     self.p_pwm.change_duty_rate(0)
 97         # else:
 98         #     self.p_pwm.change_duty_rate(self.amp
    )
 99         # if self.reverse:
100         #     self.n_pwm.change_duty_rate(self.amp
    )
101         # else:
102         #     self.n_pwm.change_duty_rate(0)
103
104         p_amp = 0
105         n_amp = 0
106         if self.reverse: p_amp = self.amp
```

```
107         if not self.reverse: n_amp = self.amp
108 
109         self.p_pwm.change_duty_rate(p_amp)
110         self.n_pwm.change_duty_rate(n_amp)
111 
112         self.time.sleep(self.period)
113         self.reverse = not self.reverse
114         pass
115         return
116 
117     def shutdown(self):
118         pass
119         self.manage_engaged = False
120         if self.manage_thread is not None: self.
    manage_thread.join()
121         if self.p_pwm is not None: self.p_pwm.
    pwm_stop()
122         if self.n_pwm is not None: self.n_pwm.
    pwm_stop()
123         pass
124         return
125 
126     def change_amp(self, amp):
127         pass
128         self.amp = amp
129         pass
130         return
131 
132     def change_freq(self, freq):
133         pass
134         self.freq = freq
135         self.period = 1 / freq
136         pass
137         return
138 
139     pass
140 
141 
142 gpio_engine = None
143 left_taptic = None
144 right_taptic = None
```

```
145
146 pin_left_p = 16
147 pin_left_n = 19
148 pin_right_p = 20
149 pin_right_n = 26
150
151
152 def shutdown():
153     pass
154     global left_taptic
155     global right_taptic
156
157     if left_taptic is not None: left_taptic.
    shutdown()
158     if right_taptic is not None: right_taptic.
    shutdown()
159     pass
160     return
161
162
163 def init():
164     pass
165     global gpio_engine
166
167     if gpio_engine is None:
168         pass
169         print("very fucked!!! shit, gpio_engine is
     missing here in RTaptic lol.")
170         raise OSError
171
172     global left_taptic
173     global right_taptic
174
175     left_taptic = Taptic(gpio_engine)
176     right_taptic = Taptic(gpio_engine)
177
178     global pin_left_p
179     global pin_left_n
180     left_taptic.set_pin(p_pin=pin_left_p, n_pin=
    pin_left_n)
181
```

```
182     global pin_right_p
183     global pin_right_n
184     right_taptic.set_pin(p_pin=pin_right_p, n_pin=
    pin_right_n)
185 
186     # debug
187     left_taptic.change_amp(1.0)
188     left_taptic.change_freq(10)
189 
190     right_taptic.change_amp(0.5)
191     right_taptic.change_freq(30)
192 
193     pass
194     return
195 
196 # line_left_p = None
197 # line_left_n = None
198 # line_right_p = None
199 # line_right_n = None
200 #
201 # pwm_left_p = None
202 # pwm_left_n = None
203 # pwm_right_p = None
204 # pwm_right_n = None
205 #
206 # pin_left_p = 16
207 # pin_left_n = 19
208 # pin_right_p = 20
209 # pin_right_n = 26
210 #
211 # target_left_freq = 0
212 # target_right_freq = 0
213 #
214 # target_left_amp = 0
215 # target_right_amp = 0
216 #
217 # taptic_freq_manage_thread = None
218 # taptic_freq_manage_thread_shutdown_flag = False
219 #
220 #
221 # def shutdown():
```

```
222 #     global
    taptic_freq_manage_thread_shutdown_flag
223 #     taptic_freq_manage_thread_shutdown_flag =
    True
224 #
225 #     global taptic_freq_manage_thread
226 #     if taptic_freq_manage_thread is not None:
227 #         taptic_freq_manage_thread.join()
228 #     pass
229 #
230 #
231 # def taptic_freq_manage_tick():
232 #     global pwm_left_p
233 #     global pwm_left_n
234 #     global pwm_right_p
235 #     global pwm_right_n
236 #
237 #     global target_left_freq
238 #     global target_right_freq
239 #
240 #     global target_left_amp
241 #     global target_right_amp
242 #
243 #     target_left
244 #
245 #
246 # def taptic_freq_manage():
247 #     global
    taptic_freq_manage_thread_shutdown_flag
248 #     while not
    taptic_freq_manage_thread_shutdown_flag:
249 #         taptic_freq_manage_tick()
250 #         time.sleep(0.01)
251 #         pass
252 #     pass
253 #
254 #
255 # def init():
256 #     global gpio_engine
257 #     global line_left_p
258 #     global line_left_n
```

```
259 #     global line_right_p
260 #     global line_right_n
261 #     global pwm_left_p
262 #     global pwm_left_n
263 #     global pwm_right_p
264 #     global pwm_right_n
265 #     if gpio_engine is None:
266 #         print("very fucked: there is no
    gpio_engine lol")
267 #         raise OSError
268 #     line_left_p = gpio_engine.set_output(
    pin_left_p, "taptic_left_p")
269 #     line_left_n = gpio_engine.set_output(
    pin_left_n, "taptic_left_n")
270 #     line_right_p = gpio_engine.set_output(
    pin_right_p, "taptic_right_p")
271 #     line_right_n = gpio_engine.set_output(
    pin_right_n, "taptic_right_n")
272 #
273 #     pwm_left_p = gpio_engine.create_pwm(
    line_left_p, 100, "taptic_left_p")
274 #     pwm_left_n = gpio_engine.create_pwm(
    line_left_n, 100, "taptic_left_n")
275 #     pwm_right_p = gpio_engine.create_pwm(
    line_right_p, 100, "taptic_right_p")
276 #     pwm_right_n = gpio_engine.create_pwm(
    line_right_n, 100, "taptic_right_n")
277 #
278 #     global taptic_freq_manage_thread
279 #     taptic_freq_manage_thread = threading.Thread
    (target=taptic_freq_manage)
280 #     taptic_freq_manage_thread.start()
281 #
282 #     # debug
283 #     # pwm_left_p.change_duty_rate(0.5)
284 #     # pwm_left_n.change_duty_rate(0.2)
285 #     # pwm_right_p.change_duty_rate(0.8)
286 #     # pwm_right_n.change_duty_rate(0.5)
287 #     # pwm_right_n.change_freq(1)
288 #
289 #     pass
```

```
290 #
291 #
292 # # def set_analog(pwm_target: str = None, value:
    float = 0.0):
293 # #     if pwm_target is None: return
294 # #     pwm_target = pwm_target.lower()
295 # #     if pwm_target == "pwm_left_p":
296 # #         global target_left_amp
297 # #     pass
298
```

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

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

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

```

```
38 try:
39     pass
40     import threading
41 except ImportError:
42     pass
43     raise ImportError("Threading not found or
   failed to load.")
44
45 tensor_output = None
46 raw_data = None
47 fps_engine = None
48 tensor_running = True
49 calculate_distance_function = None
50
51
52 def process_frame():
53     pass
54     global raw_data
55     if raw_data is None: return
56
57     global model
58     global model_input_details
59     global model_output_details
60
61     # set input tensor
62     model.set_tensor(model_input_details[0]['index'
   ], raw_data)
63
64     # invoke model
65     model.invoke()
66
67     # get output tensor
68     boxes_idx, classes_idx, scores_idx = 0, 1, 2
69     boxes = model.get_tensor(model_output_details[
   boxes_idx]['index'])[0]
70     classes = model.get_tensor(model_output_details
   [classes_idx]['index'])[0]
71     scores = model.get_tensor(model_output_details[
   scores_idx]['index'])[0]
72     distances = [None] * len(boxes)
73
```

```
74      for i in range(len(boxes)):
75          pass
76          if scores[i] < 0.5: continue
77          # get box width
78          # box_width = (boxes[i][3] - boxes[i][1
    ]) * 320
79          box = boxes[i] * 320
80          if calculate_distance_function is not None
    : distances[i] = calculate_distance_function(
    classes[i], box)
81          else: distances[i] = None
82
83      global tensor_output
84      tensor_output = (boxes, classes, scores,
    distances)
85
86      global fps_engine
87      if fps_engine is not None:
88          pass
89          fps_engine.add_candidate_tensor_fps()
90
91      return
92
93
94  def process_frame_logic():
95      pass
96      while tensor_running:
97          pass
98          process_frame()
99      return
100
101
102 def stop_process_frame():
103     pass
104     print("Stopping process frame...")
105     global process_frame_thread
106     global tensor_running
107     tensor_running = False
108     if process_frame_thread is not None:
109         pass
110         process_frame_thread.join()
```

```
111             process_frame_thread = None
112         print("Process frame stopped.")
113         return
114 
115 
116 process_frame_thread = threading.Thread(target=
    process_frame_logic)
117 process_frame_thread.start()
118 
```

```
 1 try:
 2     import bluetooth
 3 except ImportError:
 4     raise ImportError("Bluetooth not found or
   failed to load.")
 5 try:
 6     import threading
 7 except ImportError:
 8     raise ImportError("Threading not found or
   failed to load.")
 9 try:
10     import time
11 except ImportError:
12     raise ImportError("Time not found or failed to
   load.")
13
14 client_info = None
15 client_sock = None
16 bluetooth_rx_thread = None
17 bluetooth_connected = False
18 server_sock = bluetooth.BluetoothSocket(bluetooth.
   RFCOMM)
19 server_sock.bind(("", bluetooth.PORT_ANY))
20 server_sock.listen(1)
21 connected_callback = None
22 recv_callback = None
23
24
25 port = server_sock.getsockname()[1]
26 uuid = "00001101-0000-1000-8000-00805F9B34FB"
27 service_name = "FindMy"
28
29 bluetooth_connect_try_enabled = True
30
31
32 def try_routine():
33     global bluetooth_connected
34     if bluetooth_connected:
35         time.sleep(1)
36         return
37
```

```
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:
    try_routine()
67
68
69  def recv():
70      while bluetooth_connected:
```

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

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

```
notifications = []

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

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


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

    def __init__(self, icon, message):
        pass
        self.icon = icon
```

```
39          self.message = message
40          self.notification_start_time = time.time()
41          self.notification_thread = threading.Thread
    (target=self.notification_thread_routine).start()
42          return
43
44      def notification_thread_routine(self):
45          pass
46          while self.function_x < 2.0: self.
    increase_notification_size()
47          self.display_visibility = 1.0
48          while time.time() - self.
    notification_start_time < Notification.
    display_duration: time.sleep(0.1)
49          self.function_x = 0
50          while self.function_x < 1.0: self.
    decrease_notification_size()
51          self.display_visibility = 0.0
52          self.notification_finished = True
53          pass
54          return
55
56      def increase_notification_size(self):
57          pass
58          # if self.function_x <= 1:
59          #     self.display_visibility = math.sqrt(
    self.function_x)
60          # elif self.function_x <= 2:
61          #     self.display_visibility = 1 + 0.
    3535533906 * (2 - self.function_x) * math.sin(self.
    function_x - 1)
62          self.display_visibility = self.
    increase_function()
63          self.function_x += 0.01
64          time.sleep(0.005)
65          return
66
67      def decrease_notification_size(self):
68          pass
69          self.display_visibility = math.pow(self.
    function_x - 1, 2)
```

```
 70         self.function_x += 0.01
 71         time.sleep(0.005)
 72         return
 73 
 74     def increase_function(self):
 75         pass
 76         if self.function_x <= 1: return math.sqrt(
    self.function_x)
 77         if self.function_x <= 2: return 1 + 0.
    3535533906 * (2 - self.function_x) * math.sin(self
    .function_x - 1)
 78         return 0.0
 79 
 80 
 81 def add_notification(icon, notification, saying):
 82     pass
 83     global notifications
 84     global sound_engine
 85     global tts_engine
 86     notifications.append(Notification(icon,
    notification))
 87     if sound_engine is not None: sound_engine.play
    ("boot/res/alert.mp3", volume=2.0)
 88     if tts_engine is not None and tts_enabled:
    tts_engine.order_tts(saying)
 89     return notifications[-1]
 90 
 91 
 92 def get_notification(icon=None, message=None):
 93     pass
 94     global notifications
 95     if icon is None and message is None: return
    notifications
 96 
 97     candidates = []
 98     for notification in notifications:
 99         pass
100         if icon is not None and notification.icon
     != icon: continue
101         if message is not None and notification.
    message != message: continue
```

```
102         candidates.append(notification)
103 
104     if len(candidates) == 0:
105         pass
106         return None
107     elif len(candidates) == 1:
108         pass
109         return candidates[0]
110     # else:
111     #     pass
112     #     return candidates
113     return candidates
114 
115 
116 def notification_management_routine(notification):
117     pass
118     global notifications
119     if notification.notification_finished:
120         pass
121         notifications.remove(notification)
122         return
123     pass
124     return
125 
126 
127 def notifications_management_routine():
128     pass
129     global notifications
130     global notifications_management_enabled
131     while notifications_management_enabled:
132         pass
133         for notification in notifications:
    notification_management_routine(notification)
134         time.sleep(0.01)
135     return
136 
137 
138 def close():
139     pass
140     global notifications
141     for notification in notifications:
```

```
141 notification.notification_finished = True
142     global notifications_management_enabled
143     notifications_management_enabled = False
144     global notifications_management_thread
145     if notifications_management_thread is not None
    : notifications_management_thread.join()
146     pass
147     return
148 
149 
150 notifications_management_thread = threading.Thread
    (target=notifications_management_routine).start()
151 
```

```
 1 <name> ROSVersion </> <type> str </> <value>
   EdgeRunner.6 </> <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> ROSDisplayFPSColorRed </> <type> int </> <
   value> 138 </> <comment>
   ROSDisplayFPSColorRedNoticeROS display FPS color
```

```
13 red </>
14 <name> ROSDisplayFPSColorGreen </> <type> int </> <
   value> 43 </> <comment>
   ROSDisplayFPSColorGreenNoticeROS display FPS color
   green </>
15 <name> ROSDisplayFPSColorBlue </> <type> int </> <
   value> 226 </> <comment>
   ROSDisplayFPSColorBlueNoticeROS display FPS color
   blue </>
16 <name> SLDDeviceIP </> <type> str </> <value> 0.0.0
   .0 </> <comment> SLDDeviceIPNoticeSLD device IP </>
17 <name> SLDDevicePort </> <type> int </> <value>
   5001 </> <comment> SLDDevicePortNoticeSLD device
   port </>
18 <name> SLDConnectTimeout </> <type> int </> <value
   > 5 </> <comment> SLDConnectTimeoutNoticeSLD
   connect timeout </>
19 <name> ROSObjectDistanceCalculateConstantDefault
    </> <type> float </> <value> 500.0 </> <comment>
   ROSObjectDistanceCalculateConstantNoticeROS object
   distance calculate constant </>
20 <name> DistanceUnit </> <type> str </> <value> SI
    </> <comment> DistanceUnitNoticeDistance unit </>
21 <name> DistanceDisplayEnabled </> <type> bool </> <
   value> True </> <comment>
   DistanceDisplayEnabledNoticeDistance display is
   enabled or not </>
22 <name> DistanceDisplayWay </> <type> str </> <value
   > invertedColorInBox </> <comment>
   DistanceDisplayWayNoticeDistance display way </>
23 <name> ROSARDisplayEnabled </> <type> bool </> <
   value> True </> <comment>
   ROSARDisplayEnabledNoticeROS AR display is enabled
   or not </>
24 <name> AREyeDistance </> <type> float </> <value> 0
   .068 </> <comment> AREyeDistanceNoticeAR eye
   distance in meter </>
25 <name> AREyeLevelAdjust </> <type> float </> <value
   > 0.018 </> <comment> AREyeLevelAdjustNoticeAR eye
   level adjust in meter </>
26 <name> ARDisplayPPI </> <type> int </> <value> 131
```

```
26  </> <comment> ARDisplayPPINoticeAR display PPI </>
27 <name> ARDisplayWidth </> <type> int </> <value>
   800 </> <comment> ARDisplayWidthNoticeAR display
   width </>
28 <name> ARDisplayHeight </> <type> int </> <value>
   480 </> <comment> ARDisplayHeightNoticeAR display
   height </>
29 <name> ARPreferredEye </> <type> str </> <value>
   right </> <comment> ARPreferredEyeNoticeAR
   preferred eye </>
30 <name> ARMode </> <type> str </> <value> one eye
    </> <comment> ARModeFor auto or Both Eye </>
31 <name>
   ROSObjectDistanceCalculateConstantRaspberryPi </> <
   type> float </> <value> 500.0 </> <comment>
   ROSObjectDistanceCalculateConstantRaspberryPiNotice
   ROS object distance calculate constant for
   Raspberry Pi </>
32 <name> ROSObjectDistanceCalculateConstantMacBookPro
    </> <type> float </> <value> 500.0 </> <comment>
   ROSObjectDistanceCalculateConstantMacBookProNoticeR
   OS object distance calculate constant for MacBook
   Pro </>
33 <name> ROSObjectDistanceCalculateConstantIphone
    </> <type> float </> <value> 500.0 </> <comment>
   ROSObjectDistanceCalculateConstantIphoneNoticeROS
   object distance calculate constant for iPhone </>
34 <name>
   ROSObjectDistanceCalculateSideErrorMarginByVision
    </> <type> int </> <value> 5 </> <comment>
   ROSObjectDistanceCalculateSideErrorMarginByVisionNo
   ticeROS object distance calculate side error margin
    by vision </>
35 <name>
   ROSObjectDistanceCalculateXAxisOutOfBoundCheckEnabl
   ed </> <type> bool </> <value> True </> <comment>
   ROSObjectDistanceCalculateXAxisOutOfBoundCheckEnabl
   edNoticeROS object distance calculate X axis out of
    bound check is enabled or not </>
36 <name>
   ROSObjectDistanceCalculateYAxisOutOfBoundCheckEnabl
```

```
36 ed </> <type> bool </> <value> False </> <comment>
   ROSObjectDistanceCalculateYAxisOutOfBoundCheckEnabl
   edNoticeROS object distance calculate Y axis out of
    bound check is enabled or not </>
37 <name> SoundVolume </> <type> float </> <value> 0.3
    </> <comment> SoundVolumeNoticeSound volume </>
38 <name> Notification Display Overlap </> <type> bool
    </> <value> True </> <comment> Notification
   Display OverlapNoticeNotification display overlap
    </>
39 <name> Notification TTS Enabled </> <type> bool
    </> <value> True </> <comment> Notification TTS
   EnabledNoticeNotification TTS is enabled or not </>
40 <name> ROSOffKey </> <type> int </> <value> 27
    </> <comment> ROS Shutdown Key </>
41 <name> USSRegionStartX </> <type> int </> <value>
   140 </> <comment> USSRegionStartXNoticeUSS region
   start X </>
42 <name> USSRegionEndX </> <type> int </> <value> 180
    </> <comment> USSRegionEndXNoticeUSS region end X
    </>
43 <name> USSRegionStartY </> <type> int </> <value>
   140 </> <comment> USSRegionStartYNoticeUSS region
   start Y </>
44 <name> USSRegionEndY </> <type> int </> <value> 180
    </> <comment> USSRegionEndYNoticeUSS region end Y
    </>
45 <name> USSRegionDisplayEnabled </> <type> bool
    </> <value> True </> <comment>
   USSRegionDisplayEnabledNoticeUSS region display is
   enabled or not </>
46 <name> USSRegionDisplayColorRed </> <type> int
    </> <value> 208 </> <comment>
   USSRegionDisplayColorRedNoticeUSS region display
   color red </>
47 <name> USSRegionDisplayColorGreen </> <type> int
    </> <value> 252 </> <comment>
   USSRegionDisplayColorGreenNoticeUSS region display
   color green </>
48 <name> USSRegionDisplayColorBlue </> <type> int
    </> <value> 92 </> <comment>
```

```
48 USSRegionDisplayColorBlueNoticeUSS region display
   color blue </>
49 <name> RaspberryPiCameraEye </> <type> str </> <
   value> right </> <comment>
   RaspberryPiCameraEyeNoticeRaspberry Pi camera eye
    </>
50
```

```
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
```