```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("gpiochip0")
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          if self.t_on < 0: self.t_on = 0
45          if self.t_off < 0: self.t_off = 0
46          self.pwm_thread = threading.Thread(target=
    self.pwm_routine).start()
47          return
48
49      def pwm_routine(self):
50          pass
51          while self.pwm_run: self.pwm_tick()
52          return
53
54      def pwm_tick(self):
55          pass
56          if self.t_on == 0 and self.t_off == 0:
    return
57          if self.t_on != 0: self.line.set_value(1)
58          time.sleep(self.t_on)
59          if self.t_off != 0: self.line.set_value(0)
60          time.sleep(self.t_off)
61          return
62
63      def pwm_stop(self):
64          pass
65          self.pwm_run = False
66          if self.pwm_thread is not None: self.
    pwm_thread.join()
67          return
68
69      def change_duty_rate(self, duty_rate):  # 0.0
    to 1.0
70          pass
71          self.duty_rate = duty_rate
72          self.t_on = self.period * duty_rate
73          self.t_off = self.period - self.t_on
74          if self.t_on < 0: self.t_on = 0
```

```
 75         if self.t_off < 0: self.t_off = 0
 76         return
 77
 78     def change_freq(self, freq):  # Hz
 79         pass
 80         self.freq = freq
 81         self.period = 1 / freq
 82         self.t_on = self.period * self.duty_rate
 83         self.t_off = self.period - self.t_on
 84         if self.t_on < 0: self.t_on = 0
 85         if self.t_off < 0: self.t_off = 0
 86         return
 87
 88     def pwm_restart(self):
 89         pass
 90         self.pwm_run = False
 91         if self.pwm_thread is not None: self.
    pwm_thread.join()
 92         self.pwm_run = True
 93         self.pwm_thread = threading.Thread(target=
    self.pwm_routine)
 94         self.pwm_thread.start()
 95         return
 96
 97
 98 def set_output(pin, name, original_state=False):
 99     pass
100     line = chip.get_line(pin)
101     line.request(consumer=name, type=gpiod.
    LINE_REQ_DIR_OUT, default_vals=[original_state])
102     lines.append(line)
103     return line
104
105
106 def set_input(pin, name):
107     pass
108     line = chip.get_line(pin)
109     line.request(consumer=name, type=gpiod.
    LINE_REQ_DIR_IN)
110     lines.append(line)
111     return line
```

```
112
113
114 def output_write(line, value):
115     pass
116     line.set_value(value)
117     return
118
119
120 def input_read(line):
121     pass
122     return line.get_value()
123
124
125 def create_pwm(line, freq, name):
126     pass
127     pwm = PWM(line, freq, 0, name)
128     pwms[name] = pwm
129     return pwm
130
131
132 def get_pwm(name):
133     pass
134     return pwms[name]
135
136
137 def shutdown():
138     pass
139     for pwm in pwms.values():
140         pass
141         pwm.pwm_stop()
142     for line in lines:
143         pass
144         line.release()
145     chip.close()
146     return
147
```

```
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
```

```
 1 from boot.RTensor import model
 2
 3 print("RKernel is booting up...")
 4
 5 print("defining pre def methods...")
 6
 7
 8 def make_error(error_code: str, error_message: str
   ):
 9     pass
10     print("E" + error_code + ": " + error_message)
11     exit(error_code)
12
13
14 print("methods pre def defined.")
15
16 print("Loading Third Party imports...")
17 try:
18     pass
19     import cv2 as cv
20 except ImportError:
21     pass
22     make_error("1001", "cv2 not found.")
23 try:
24     pass
25     import time
26 except ImportError:
27     pass
28     make_error("1002", "time not found.")
29 try:
30     pass
31     import picamera2
32 except ImportError:
33     pass
34     print("Picamera2 not found.")
35 try:
36     pass
37     import threading
38 except ImportError:
39     pass
40     make_error("1005", "threading not found.")
```

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

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

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

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

```
192                     raw_frame.shape[1] // 2 -
    target_width // 2: raw_frame.shape[1] // 2 +
    target_width // 2]
193     return raw_frame
194
195
196 def make_window():
197     pass
198     cv.namedWindow("ROS", cv.WINDOW_NORMAL)
199     if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
200         pass
201         ar_width = key_engine.get_key("
    ARDisplayWidth").get("value")
202         ar_height = key_engine.get_key("
    ARDisplayHeight").get("value")
203         cv.resizeWindow("ROS", ar_width, ar_height
    )
204     else:
205         pass
206         cv.resizeWindow("ROS", 320, 320)
207     return
208
209
210 def get_camera():
211     pass
212     if "picamera2" in sys.modules:
213         pass
214         camera = picamera2.Picamera2()
215         camera.configure(camera.
    create_preview_configuration(main={"size": (320,
    320)}))
216         camera.start()
217     else:
218         pass
219         camera = cv.VideoCapture(0)
220     return camera
221
222
223 def align_camera_frame(frame):
224     pass
```

```
225     camera_eye_location = key_engine.get_key("
    RaspberryPiCameraEye").get("value")
226     if camera_eye_location == "left": return
    rotate_clockwise_90(frame)
227     if camera_eye_location == "right": return
    rotate_counterclockwise_90(frame)
228     return frame
229 
230 
231 def get_frame():
232     pass
233     global kernel_panicked
234     if kernel_panicked:
235         pass
236         return get_320_320_frame(
    kernel_panic_screen)
237     global camera
238     if "picamera2" in sys.modules:
239         pass
240         frame = camera.capture_array()
241         frame = cv.cvtColor(frame, cv.
    COLOR_BGR2RGB)
242         camera_eye_location = key_engine.get_key("
    RaspberryPiCameraEye").get("value")
243         frame = align_camera_frame(frame)
244     else:
245         pass
246         frame = camera.read()[1]
247 
248     if frame is None:
249         pass
250         print("camera read failed")
251         kernel_panicked = True
252         return None
253 
254     # check brightness
255     frame_average_red = np.average(frame[:, :, 2])
256     frame_average_green = np.average(frame[:, :, 1
    ])
257     frame_average_blue = np.average(frame[:, :, 0
    ])
```

```
258     frame_average_brightness = (frame_average_red
     + frame_average_green + frame_average_blue) / 3
259     if frame_average_brightness < 60 and
    notification_engine.get_notification(
260             message="low brightness detected") is
    None:
261         pass
262         notification_engine.add_notification("
    hard_warning.png", "low brightness detected",
263                                              "저조도
     감지. 사용자의 즉각적인 주의가 필요합니다.")
264
265     # make frame 320x320
266     return get_320_320_frame(frame)
267
268
269 # def calculate_distance(class_index: int,
    box_width_pixel):
270 def calculate_distance(class_index: int, box):
271     pass
272     box_start_x = box[1]
273     box_end_x = box[3]
274     box_start_y = box[0]
275     box_end_y = box[2]
276     if label_engine.get_label(class_index + 1) is
    None: return str("N/A")
277     object_average_width = label_engine.get_label(
    class_index + 1).get("average_width")
278
279     # check if the object is in the center of the
    frame and can be calculated by vision
280     can_calculated_by_vision = True
281     vision_side_margin = key_engine.get_key("
    ROSObjectDistanceCalculateSideErrorMarginByVision"
    ).get("value")
282     x_check = key_engine.get_key("
    ROSObjectDistanceCalculateYAxisOutOfBoundCheckEnab
    led").get("value")
283     y_check = key_engine.get_key("
    ROSObjectDistanceCalculateXAxisOutOfBoundCheckEnab
    led").get("value")
```

```
284     if (box_start_x <= vision_side_margin or
    box_end_x >= (320 - vision_side_margin)) and
    x_check:
285         pass
286         can_calculated_by_vision = False
287     if (box_start_y <= vision_side_margin or
    box_end_y >= (320 - vision_side_margin)) and
    y_check:
288         pass
289         can_calculated_by_vision = False
290 
291     can_calculated_by_uss = False
292     if "boot.RUSS" in sys.modules:
293         pass
294         # check if the object is in the center of
    the frame and can be calculated by uss
295         uss_region_start_x = key_engine.get_key("
    USSRegionStartX").get("value")
296         uss_region_end_x = key_engine.get_key("
    USSRegionEndX").get("value")
297         uss_region_start_y = key_engine.get_key("
    USSRegionStartY").get("value")
298         uss_region_end_y = key_engine.get_key("
    USSRegionEndY").get("value")
299 
300         inside_uss_x_region = False
301         inside_uss_y_region = False
302         if box_end_x >= uss_region_start_x and
    box_start_x <= uss_region_end_x:
    inside_uss_x_region = True
303         if box_end_y >= uss_region_start_y and
    box_start_y <= uss_region_end_y:
    inside_uss_y_region = True
304 
305         can_calculated_by_uss =
    inside_uss_x_region and inside_uss_y_region
306         # pass
307 
308     # check camera system and get calculate
    constant
309     if "picamera2" in sys.modules:
```

```
310         pass
311         distance_calculate_constant = key_engine.
    get_key("
    ROSObjectDistanceCalculateConstantRaspberryPi").
    get("value")
312     else:
313         pass
314         distance_calculate_constant = key_engine.
    get_key("
    ROSObjectDistanceCalculateConstantMacBookPro").get
    ("value")
315 
316     vision_distance_in_meter =
    object_average_width / ((box_end_x - box_start_x
    ) / distance_calculate_constant)
317     uss_distance_in_meter = 0.0
318     if "boot.RUSS" in sys.modules:
    uss_distance_in_meter = ultrasonic_engine.
    output_distance
319     if uss_distance_in_meter == -1.0:
    can_calculated_by_uss = False
320 
321     likely_distance_in_meter = 0.0
322     if not can_calculated_by_uss and not
    can_calculated_by_vision:
323         pass
324         return str("")
325     elif can_calculated_by_uss and not
    can_calculated_by_vision:
326         pass
327         likely_distance_in_meter =
    uss_distance_in_meter
328     elif not can_calculated_by_uss and
    can_calculated_by_vision:
329         pass
330         likely_distance_in_meter =
    vision_distance_in_meter
331     elif can_calculated_by_uss and
    can_calculated_by_vision:
332         pass
333         # I think vision is more precise
```

```
334         likely_distance_in_meter =
    vision_distance_in_meter
335
336     unit = key_engine.get_key("DistanceUnit").get(
    "value")
337     if unit == "SI":
338         pass
339         if likely_distance_in_meter < 1: return
    str(round(likely_distance_in_meter * 100, 2)) + "
    cm"
340         if likely_distance_in_meter <= 1000:
    return str(round(likely_distance_in_meter, 2)) +
    " m"
341         if likely_distance_in_meter > 1000: return
     str(round(likely_distance_in_meter / 1000, 2)) +
    " km"
342     if unit == "US":
343         pass
344         if likely_distance_in_meter < 0.3048:
    return str(round(likely_distance_in_meter * 39.
    3701, 2)) + " in"
345         if likely_distance_in_meter <= 0.9144:
    return str(round(likely_distance_in_meter * 3.
    28084, 2)) + " ft"
346         if likely_distance_in_meter <= 1609.34:
    return str(round(likely_distance_in_meter * 1.
    09361, 2)) + " yd"
347         if likely_distance_in_meter > 1609.34:
    return str(round(likely_distance_in_meter / 1609.
    34, 2)) + " mi"
348     pass
349     return str("ERR")
350     # # this code (below) is discarded
351     # if "picamera2" in sys.modules:
352     #     distance_calculate_constant = key_engine
    .get_key("
    ROSObjectDistanceCalculateConstantRaspberryPi").
    get("value")
353     # else:
354     #     distance_calculate_constant = key_engine
    .get_key("
```

```
354 ROSObjectDistanceCalculateConstantMacBookPro").get
    ("value")
355     #
356     # if label_engine.get_label(class_index + 1)
    is None:
357     #     object_average_width = -1.0
358     # else:
359     #     object_average_width = label_engine.
    get_label(class_index + 1).get("average_width")
360     #
361     # if object_average_width == -1.0:
362     #     return str("N/A")
363     #
364     # distance_in_meter = object_average_width / (
    box_width_pixel / distance_calculate_constant)
365     # unit = key_engine.get_key("DistanceUnit").
    get("value")
366     # if unit == "SI" and distance_in_meter < 1:
367     #     return str(round(distance_in_meter * 100
    , 2)) + " cm"
368     # elif unit == "SI" and distance_in_meter <=
    1000:
369     #     return str(round(distance_in_meter, 2
    )) + " m"
370     # elif unit == "SI" and distance_in_meter >
    1000:
371     #     return str(round(distance_in_meter /
    1000, 2)) + " km"
372     # elif unit == "US" and distance_in_meter < 0.
    3048:
373     #     return str(round(distance_in_meter * 39.
    3701, 2)) + " in"
374     # elif unit == "US" and distance_in_meter <= 0
    .9144:
375     #     return str(round(distance_in_meter * 3.
    28084, 2)) + " ft"
376     # elif unit == "US" and distance_in_meter <=
    1609.34:
377     #     return str(round(distance_in_meter * 1.
    09361, 2)) + " yd"
378     # elif unit == "US" and distance_in_meter >
```

```
378 1609.34:
379     #     return str(round(distance_in_meter /
    1609.34, 2)) + " mi"
380     # else:
381     #     return str("ERR")
382
383
384 def make_ar_frame(frame):
385     pass
386     # make sure that input frame is 320x320
387     frame = get_320_320_frame(frame)
388     # frame input resolution: 320 320
389     ar_width = key_engine.get_key("ARDisplayWidth"
    ).get("value")
390     ar_height = key_engine.get_key("
    ARDisplayHeight").get("value")
391     ar_ppi = key_engine.get_key("ARDisplayPPI").
    get("value")
392     user_eye_distance = key_engine.get_key("
    AREyeDistance").get("value")  # meter
393     user_eye_level_adjust = key_engine.get_key("
    AREyeLevelAdjust").get("value")  # meter
394
395     ar_screen = np.zeros((ar_height, ar_width, 3
    ), np.uint8)
396     ar_screen = cv.cvtColor(ar_screen, cv.
    COLOR_BGR2RGB)
397
398     eye_distance_to_inch = user_eye_distance * 39.
    3701
399     eye_distance_to_pixel = int(
    eye_distance_to_inch * ar_ppi)
400
401     eye_level_adjust_to_inch =
    user_eye_level_adjust * 39.3701
402     eye_level_adjust_to_pixel = int(
    eye_level_adjust_to_inch * ar_ppi)
403
404     left_eye_center_x = (ar_width // 2) - (
    eye_distance_to_pixel // 2)
405     right_eye_center_x = (ar_width // 2) + (
```

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

```
435
436     left_eye_screen_width = left_eye_max_x -
    left_eye_start_x
437     right_eye_screen_width = right_eye_start_x +
    320 - right_eye_min_x
438
439     if left_eye_screen_width > 320:
    left_eye_screen_width = 320
440     if right_eye_screen_width > 320:
    right_eye_screen_width = 320
441
442     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
443     right_eye_screen = right_eye_screen[:, 320 -
    right_eye_screen_width:]
444
445     eye_screen_height = 320
446     eye_offset_overlap = 0
447     if eye_start_y < 0:
448         pass
449         eye_screen_height += eye_start_y
450         eye_offset_overlap = -eye_start_y
451         eye_start_y = 0
452         left_eye_screen = left_eye_screen[
    eye_offset_overlap:320, :]
453         right_eye_screen = right_eye_screen[
    eye_offset_overlap:320, :]
454
455     if eye_start_y + eye_screen_height > ar_height
    :
456         pass
457         eye_screen_height = ar_height -
    eye_start_y
458         left_eye_screen = left_eye_screen[0:
    eye_screen_height, :]
459         right_eye_screen = right_eye_screen[0:
    eye_screen_height, :]
460
461     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
462     left_eye_start_x:left_eye_start_x +
```

```
462 left_eye_screen_width] = left_eye_screen
463     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
464     right_eye_start_x + eye_screen_height -
    right_eye_screen_width:right_eye_start_x +
    eye_screen_height] = right_eye_screen
465 
466     # if key_engine.get_key("ARMode").get("value
    ") == "both eye":
467     #     # new version
468     #     left_eye_screen = frame
469     #     right_eye_screen = frame
470     #
471     #     center_cross_bar_width = 0.005  # meter
472     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
473     #
474     #     left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
475     #     right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
476     #
477     #     left_eye_screen_width = left_eye_max_x
     - left_eye_start_x
478     #     right_eye_screen_width =
    right_eye_start_x + 320 - right_eye_min_x
479     #
480     #     if left_eye_screen_width > 320:
    left_eye_screen_width = 320
481     #     if right_eye_screen_width > 320:
    right_eye_screen_width = 320
482     #
483     #     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
484     #     right_eye_screen = right_eye_screen[:,
    320 - right_eye_screen_width:]
485     #
486     #     eye_screen_height = 320
487     #     eye_offset_overlap = 0
488     #     if eye_start_y < 0:
489     #         eye_screen_height += eye_start_y
```

```
490     #         eye_offset_overlap = -eye_start_y
491     #         eye_start_y = 0
492     #
493     #     left_eye_screen = left_eye_screen[
    eye_offset_overlap:320, :]
494     #     right_eye_screen = right_eye_screen[
    eye_offset_overlap:320, :]
495     #
496     #     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
497     #     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
498     #     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
499     #     right_eye_start_x + eye_screen_height -
    right_eye_screen_width:right_eye_start_x +
    eye_screen_height] = right_eye_screen
500     #
501     # elif key_engine.get_key("ARMode").get("value
    ") == "one eye" and preferred_eye == "left":
502     #     # ar_screen[eye_start_y:eye_start_y +
    320, left_eye_start_x:left_eye_start_x + 320] =
    frame
503     #     left_eye_screen = frame
504     #
505     #     center_cross_bar_width = 0.01  # meter
506     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
507     #
508     #     left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
509     #     left_eye_screen_width = left_eye_max_x
     - left_eye_start_x
510     #     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
511     #
512     #     ar_screen[eye_start_y:eye_start_y + 320,
513     #     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
514     #     pass
515     #
```

```
516     # elif key_engine.get_key("ARMode").get("value
    ") == "one eye" and preferred_eye == "right":  #
    copy right eye
517     #     # ar_screen[eye_start_y:eye_start_y +
    320, right_eye_start_x:right_eye_start_x + 320] =
    frame
518     #     right_eye_screen = frame
519     #
520     #     center_cross_bar_width = 0.01  # meter
521     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
522     #
523     #     right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
524     #     right_eye_screen_width =
    right_eye_start_x + 320 - right_eye_min_x
525     #     right_eye_screen = right_eye_screen[:,
    320 - right_eye_screen_width:]
526     #
527     #     ar_screen[eye_start_y:eye_start_y + 320,
528     #     right_eye_start_x + 320 -
    right_eye_screen_width:right_eye_start_x + 320] =
    right_eye_screen
529     #     pass
530
531     return ar_screen
532
533
534 def get_icon(icon_name: str, size: int):
535     pass
536     # if icon_name == "hard_warning.png":
537     #     return hard_warning_icon
538     # elif icon_name == "warning.png":
539     #     return warning_icon
540     # else:
541     #     return black_screen
542     if icon_name == "hard_warning.png":
543         pass
544         return cv.resize(hard_warning_icon, (size
    , size))
545     elif icon_name == "warning.png":
```

```
546             pass
547             return cv.resize(warning_icon, (size, size
    ))
548         # else:
549         #     pass
550         return cv.resize(black_screen, (size, size))
551         # pass
552 
553 
554 def render_tensor(screen, boxes, classes, scores,
    distance):
555         pass
556         if scores < 0.5:
557             pass
558             return screen
559         box = boxes * [320, 320, 320, 320]
560         class_name = label_engine.get_label(int(
    classes + 1)).get("value")
561         class_color = color_engine.get_color(
    class_name)
562         inverted_color = (255 - class_color[0], 255 -
    class_color[1], 255 - class_color[2])
563         text_x, text_y = 0, 0  # for text position
564         if key_engine.get_key("ROSMindDisplayWay").get
    ("value") == "filled box":
565             pass
566             # outer line
567             cv.rectangle(screen, (int(box[1]), int(box
    [0]), int(box[3]), int(box[2])), inverted_color, 2
    )
568             # inner box
569             cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, -1
    )
570             # put text center of the box
571             text_size = cv.getTextSize(class_name, cv.
    FONT_HERSHEY_SIMPLEX, 0.5, 2)[0]
572             text_x = int((box[1] + box[3]) / 2 -
    text_size[0] / 2)
573             text_y = int((box[0] + box[2]) / 2 +
    text_size[1] / 2)
```

```
574         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
575                    inverted_color, 2)
576     elif key_engine.get_key("ROSMindDisplayWay").
    get("value") == "outlined box":
577         pass
578         cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, 2)
579         text_x = int(box[1])
580         text_y = int(box[0])
581         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
582                    class_color, 2)
583 
584     if key_engine.get_key("DistanceDisplayEnabled"
    ).get("value"):
585         pass
586         cv.putText(screen, str(distance), (text_x
    , text_y + 20), cv.FONT_HERSHEY_SIMPLEX, 0.5,
587                    inverted_color, 1, cv.LINE_AA)
588 
589     return screen
590 
591 
592 def render_tensors(tensor_output):
593     pass
594     global raw_screen
595     boxes, classes, scores, distance =
    tensor_output
596     in_print_screen = raw_screen
597     # min_distance = 1000.0
598     for i in range(len(scores)):
599         pass
600         in_print_screen = render_tensor(
    in_print_screen, boxes[i], classes[i], scores[i],
    distance[i])
601         # if distance[i].endswith("cm") and float(
    distance[i][:-2]) < min_distance: min_distance =
    float(distance[i][:-2])
602     # if min_distance != 1000.0 and "boot.RTaptic
    " in sys.modules:
```

```
603     #     pass
604     #
605     return in_print_screen
606
607
608 def render_notifications(frame, notifications):
609     pass
610     one_notification_max_height = 40
611     one_notification_width = 260
612     notification_start_y = 10
613     printed_y_pixel = 0
614
615     global black_screen
616     global hard_warning_icon
617
618     for i in range(len(notifications)):
619         pass
620         # draw notification
621         this_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
622         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
623         this_notification_start_x = (320 -
    this_notification_width) // 2
624         this_notification_radius = int(
    this_notification_height // 2)
625         cv.circle(frame, (
    this_notification_start_x +
    this_notification_radius,
626                           notification_start_y +
    this_notification_radius + printed_y_pixel),
627                   this_notification_radius, (255,
    255, 255), -1)
628         cv.circle(frame, (
    this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
629                           notification_start_y +
    this_notification_radius + printed_y_pixel),
```

```
630                     this_notification_radius, (255,
    255, 255), -1)
631             cv.rectangle(frame,
632                          (this_notification_start_x +
    this_notification_radius, notification_start_y +
    printed_y_pixel),
633                          (this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
634                           notification_start_y +
    printed_y_pixel + this_notification_height), (255
    , 255, 255), -1)
635             printed_y_pixel +=
    this_notification_height
636
637         printed_y_pixel = 0
638         for i in range(len(notifications) - 1):
639             pass
640             upper_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
641             lower_notification_height = int(
    one_notification_max_height * notifications[i + 1
    ].display_visibility)
642             upper_notification_radius = int(
    upper_notification_height // 2)
643             lower_notification_radius = int(
    lower_notification_height // 2)
644             upper_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
645             lower_notification_width = int(
    one_notification_width * notifications[i + 1].
    display_visibility)
646             upper_notification_start_x = (320 -
    upper_notification_width) // 2
647             lower_notification_start_x = (320 -
    lower_notification_width) // 2
648             this_notification_width = min(
    upper_notification_width, lower_notification_width
    )
```

```
649         this_notification_start_x = max(
    upper_notification_start_x,
    lower_notification_start_x)
650 
651         max_circle_radius = max(
    upper_notification_radius,
    lower_notification_radius)
652         cv.rectangle(frame,
653                      (this_notification_start_x,
    notification_start_y + printed_y_pixel +
    upper_notification_radius),
654                      (this_notification_start_x +
    max_circle_radius,
655                       notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
656                      (255, 255, 255), -1)
657         cv.rectangle(frame, (
    this_notification_start_x +
    this_notification_width - max_circle_radius,
658                              notification_start_y
     + printed_y_pixel + upper_notification_radius),
659                      (this_notification_start_x +
    this_notification_width,
660                       notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
661                      (255, 255, 255), -1)
662         divide_bar_height = 2
663         cv.rectangle(frame, (
    this_notification_start_x,
664                              notification_start_y
     + printed_y_pixel + upper_notification_height -
    divide_bar_height // 2),
665                      (this_notification_start_x +
    this_notification_width,
666                       notification_start_y +
    printed_y_pixel + upper_notification_height +
    divide_bar_height // 2),
667                      (200, 200, 200), -1)
668         printed_y_pixel +=
```

```
668 upper_notification_height
669 
670     printed_y_pixel = 0
671     printed_y_pixel_after = 0
672     # now draw notification
673     for i in range(len(notifications)):
674         pass
675         printed_y_pixel += printed_y_pixel_after
676         this_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
677         printed_y_pixel_after =
    this_notification_height
678         if this_notification_height <= 10:
    continue
679         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
680         this_notification_start_x = (320 -
    this_notification_width) // 2
681         this_notification_radius = int(
    this_notification_height // 2)
682         this_notification_image_size =
    this_notification_height - 10
683         if this_notification_image_size <= 0:
    continue
684         # icon = cv.resize(black_screen, (
    this_notification_image_size,
    this_notification_image_size))
685         # if notifications[i].icon == "
    hard_warning.png":
686         #     icon = cv.resize(hard_warning_icon
    , (this_notification_image_size,
    this_notification_image_size))
687         # elif notifications[i].icon == "warning.
    png":
688         #     icon = cv.resize(warning_icon, (
    this_notification_image_size,
    this_notification_image_size))
689         icon = get_icon(notifications[i].icon,
    this_notification_image_size)
```

```
690         frame[
691         notification_start_y + 5 + printed_y_pixel
    :notification_start_y + 5 + printed_y_pixel +
    this_notification_image_size,
692         this_notification_start_x + 5:
    this_notification_start_x + 5 +
    this_notification_image_size] = icon
693         string_able_pixel_width =
    this_notification_width -
    this_notification_image_size - 10
694         string_able_pixel_height =
    this_notification_height - 10
695         string_max_length =
    string_able_pixel_width // 10
696         string = notifications[i].message
697         if len(string) > string_max_length: string
     = string[:string_max_length] + "..."
698         cv.putText(frame, string, (
    this_notification_start_x +
    this_notification_image_size + 10,
699
    notification_start_y + 5 + printed_y_pixel +
    this_notification_height // 2 + 5),
700                     cv.FONT_HERSHEY_SIMPLEX, 0.5,
701                     (64, 64, 64), 1, cv.LINE_AA)
702
703     return frame
704
705
706 def render_tensor_and_etc():
707     pass
708     global raw_screen
709     new_frame = raw_screen
710     # render tensor
711     tensor_output = tensor_engine.tensor_output
712     if tensor_output is not None:
713         pass
714         new_frame = render_tensors(tensor_output)
715     # render etc
716     # render notifications
717     notifications_count = len(notification_engine.
```

```
717 notifications)
718     try:
719         pass
720         if key_engine.get_key("Notification
    Display Overlap").get("value"): new_frame =
    render_notifications(new_frame,
721 
                          notification_engine.notifications)
722     except Exception as e:
723         print("Error while rendering notifications
    .")
724         print(e)
725         pass
726     # render fps
727     if key_engine.get_key("ROSDisplayFPSEnable").
    get("value"):
728         pass
729         main_screen_fps = round(fps_engine.
    get_main_screen_fps(), 2)
730         tensor_fps = round(fps_engine.
    get_tensor_fps(), 2)
731 
732         red = key_engine.get_key("
    ROSDisplayFPSColorRed").get("value")
733         green = key_engine.get_key("
    ROSDisplayFPSColorGreen").get("value")
734         blue = key_engine.get_key("
    ROSDisplayFPSColorBlue").get("value")
735 
736         # cv.putText(new_frame, "MS FPS: " + str(
    main_screen_fps), (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5,
737         #            (255, 255, 255), 1, cv.
    LINE_AA)
738         # cv.putText(new_frame, " T FPS: " + str(
    tensor_fps), (10, 40), cv.FONT_HERSHEY_SIMPLEX, 0.
    5, (255, 255, 255),
739         #            1, cv.LINE_AA)
740         cv.putText(new_frame, "MS FPS: " + str(
    main_screen_fps), (10, 20), cv.
```

```
740 FONT_HERSHEY_SIMPLEX, 0.5,
741                     (blue, green, red), 1, cv.
    LINE_AA)
742         cv.putText(new_frame, " T FPS: " + str(
    tensor_fps), (10, 40), cv.FONT_HERSHEY_SIMPLEX, 0.
    5, (blue, green, red),
743                     1, cv.LINE_AA)
744 
745     # render USSRegion
746     if key_engine.get_key("USSRegionDisplayEnabled
    ").get("value") and "boot.RUSS" in sys.modules:
747         pass
748         uss_region_start_x = key_engine.get_key("
    USSRegionStartX").get("value")
749         uss_region_end_x = key_engine.get_key("
    USSRegionEndX").get("value")
750         uss_region_start_y = key_engine.get_key("
    USSRegionStartY").get("value")
751         uss_region_end_y = key_engine.get_key("
    USSRegionEndY").get("value")
752 
753         red = key_engine.get_key("
    USSRegionDisplayColorRed").get("value")
754         green = key_engine.get_key("
    USSRegionDisplayColorGreen").get("value")
755         blue = key_engine.get_key("
    USSRegionDisplayColorBlue").get("value")
756         # cv.rectangle(new_frame, (
    uss_region_start_x, uss_region_start_y), (
    uss_region_end_x, uss_region_end_y),
757         #              (208, 252, 92), 1)
758         cv.rectangle(new_frame, (
    uss_region_start_x, uss_region_start_y), (
    uss_region_end_x, uss_region_end_y),
759                      (blue, green, red), -1)
760     global screen
761     screen = new_frame
762     return new_frame
763 
764 
765 def tick_screen():
```

```
766     pass
767     global kernel_panicked
768     global screen
769     if kernel_panicked:
770         pass
771         screen = kernel_panic_screen
772     make_window()
773     if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
774         pass
775         cv.imshow("ROS", make_ar_frame(screen))
776     else:
777         pass
778         cv.imshow("ROS", screen)
779 
780     fps_engine.add_candidate_main_fps()
781     if fps_engine.get_main_screen_fps() < 20 and
    notification_engine.get_notification(
782             message="Low System FPS detected.") is
     None:
783         pass
784         notification_engine.add_notification("
    warning.png", "Low System FPS detected.",
785                                              "시스템
     FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
786     if fps_engine.get_tensor_fps() < 15 and
    notification_engine.get_notification(
787             message="Low Tensor FPS detected.") is
     None:
788         pass
789         notification_engine.add_notification("
    warning.png", "Low Tensor FPS detected.", "텐서
    FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
790 
791     return
792 
793 
794 def set_tensor_input():
795     pass
796     global raw_screen
797     raw_screen = get_320_320_frame(raw_screen)
```

```
798     raw_data = cv.cvtColor(raw_screen, cv.
    COLOR_BGR2RGB)
799     raw_data = np.expand_dims(raw_data, axis=0)
800     tensor_engine.raw_data = raw_data
801     return
802 
803 
804 def shutdown():
805     pass
806     print("Shutting down...")
807     global camera
808     notification_engine.add_notification("
    hard_warning.png", "Shutting down...", "종료 중...")
809     # shutdown thread later
810     tensor_engine.stop_process_frame()
811     label_engine.erase_memory()
812     color_engine.erase_memory()
813     if "boot.RBluetooth" in sys.modules:
814         pass
815         bluetooth_engine.close()
816     key_engine.save_keys()
817     cv.destroyAllWindows()
818     if "picamera2" in sys.modules:
819         pass
820         camera.stop()
821     else:
822         pass
823         camera.release()
824     notification_engine.close()
825     tts_engine.shutdown()
826     if "boot.RTaptic" in sys.modules:
827         pass
828         taptic_engine.shutdown()
829         global taptic_command_thread_run
830         taptic_command_thread_run = False
831         global taptic_command_thread
832         if taptic_command_thread is not None:
    taptic_command_thread.join()
833     if "boot.RUSS" in sys.modules:
834         pass
835         ultrasonic_engine.shutdown()
```

```
836     if "boot.RGPIO" in sys.modules:
837         pass
838         gpio_engine.shutdown()
839     # time.sleep(2)
840     current_threads = threading.enumerate()
841     for thread in current_threads:
842         pass
843         if thread.name == "MainThread": continue
844         print("Thread " + thread.name + " is in
    running.")
845     if len(current_threads) > 1:
846         pass
847         print("There are still threads running.")
848         print("Force shutdown.")
849         os._exit(0)
850     print("Shutdown complete.")
851     exit(0)
852
853
854 def bluetooth_connected_callback():
855     pass
856     sound_engine.play("boot/res/alert.mp3")
857     print("Bluetooth connected.")
858     notification_engine.add_notification("warning.
    png", "Bluetooth connected.", "블루투스 연결되었습니다.")
859     return
860
861
862 def bluetooth_signal_callback(data):
863     pass
864     global find_my_sounding_one_channel
865     global find_my_keep_sounding_channel
866     global find_my_notification
867     if data == b"a":
868         pass
869         sound_engine.stop(
    find_my_sounding_one_channel)
870         find_my_sounding_one_channel =
    sound_engine.play("boot/res/FindMy.mp3")
871         notification_engine.add_notification("
    warning.png", "Find My is activated.", "나의 찾기가
```

```
871 활성화 되었습니다.")
872     elif data == b"b":
873         pass
874         find_my_keep_sounding_channel =
    sound_engine.play("boot/res/FindMy_long.mp3", -1)
875         find_my_notification = notification_engine
    .add_notification("warning.png", "Find My is
    activated.",
876 
                      "나의 찾기가 활성화 되었습니다.")
877         find_my_notification.display_duration =
    1000000
878         # pass
879     elif data == b"c":
880         pass
881         sound_engine.stop(
    find_my_keep_sounding_channel)
882         find_my_notification.display_duration = 0
883         # pass
884     return
885 
886 
887 def boot_logo(started_ticks: float, target_ticks:
    float = 8.0):
888     pass
889     global screen
890     # screen = black_screen
891     # copy black screen to screen
892     screen = np.zeros((320, 320, 3), np.uint8)
893     screen = cv.cvtColor(screen, cv.COLOR_BGR2RGB)
894     global splash_screen
895     resized_splash_screen = cv.resize(
    splash_screen, (80, 80))
896     screen[120:200, 120:200] =
    resized_splash_screen
897 
898     # cv.putText(screen, "ROS", (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv.
    LINE_AA)
899     boot_progress = 0
900     if started_ticks < target_ticks * 0.25:
```

```
901             pass
902             return
903         elif started_ticks < target_ticks * 0.35:
904             # boot_progress = started_ticks * 10.0
905             # boot_progress = started_ticks / (
    target_ticks * 0.35) * 30.0
906             boot_progress = (started_ticks -
    target_ticks * 0.25) / (target_ticks * 0.1) * 30.0
907             pass
908         elif started_ticks < target_ticks * 0.6:
909             # boot_progress = 30 + (started_ticks - 3
    ) * 30.0
910             # rage is 30 to 90
911             boot_progress = 30 + (started_ticks -
    target_ticks * 0.35) / (target_ticks * 0.25) * 60.
    0
912             pass
913         elif started_ticks < target_ticks * 0.9:
914             # boot_progress = 90 + 10 / 1.5 * (
    started_ticks - 5)
915             # range is 90 to 100
916             boot_progress = 90 + (started_ticks -
    target_ticks * 0.6) / (target_ticks * 0.3) * 10.0
917             pass
918         else:
919             boot_progress = 100
920             pass
921 
922         progress_bar_width_margin = 40
923         progress_bar_height_margin = 10
924         progress_bar_height = 5
925         progress_bar_width = 320 -
    progress_bar_width_margin * 2
926         progress_bar_corner_radius =
    progress_bar_height // 2
927         # progress bar background
928         cv.circle(screen, (progress_bar_width_margin
     + progress_bar_corner_radius,
929                            320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
```

```
929 progress_bar_corner_radius,
930                (64, 64, 64), -1)
931     cv.circle(screen, (320 -
    progress_bar_width_margin -
    progress_bar_corner_radius,
932                        320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
933                (64, 64, 64), -1)
934     cv.rectangle(screen, (
935         progress_bar_width_margin +
    progress_bar_corner_radius, 320 -
    progress_bar_height_margin - progress_bar_height),
936                  (320 - progress_bar_width_margin
     - progress_bar_corner_radius, 320 -
    progress_bar_height_margin),
937                  (64, 64, 64), -1)
938     # progress bar
939     cv.circle(screen, (progress_bar_width_margin
     + progress_bar_corner_radius,
940                        320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
941                (255, 255, 255), -1)
942     cv.rectangle(screen, (
943         progress_bar_width_margin +
    progress_bar_corner_radius, 320 -
    progress_bar_height_margin - progress_bar_height),
944                  (
945                      int(progress_bar_width_margin
     + progress_bar_corner_radius + (
946                              320 -
    progress_bar_width_margin -
    progress_bar_corner_radius -
    progress_bar_width_margin -
    progress_bar_corner_radius) * (
947                                  boot_progress /
    100)), 320 - progress_bar_height_margin), (255,
    255, 255), -1)
```

```
948     cv.circle(screen, (int(
    progress_bar_width_margin +
    progress_bar_corner_radius + (
949             320 - progress_bar_width_margin -
    progress_bar_corner_radius -
    progress_bar_width_margin -
    progress_bar_corner_radius) * (
950                                     boot_progress
     / 100)),
951                         320 -
    progress_bar_height_margin -
    progress_bar_corner_radius),
    progress_bar_corner_radius,
952                (255, 255, 255), -1)
953     return
954
955
956 def rotate_clockwise_90(frame):
957     pass
958     return cv.transpose(cv.flip(frame, 0))
959
960
961 def rotate_counterclockwise_90(frame):
962     pass
963     return cv.transpose(cv.flip(frame, 1))
964
965
966 def rotate_180(frame):
967     pass
968     return cv.flip(frame, -1)
969
970
971 def kernel_panic_check():
972     pass
973     # debug
974     # return True
975     if hard_warning_icon is None:
976         pass
977         return True
978     if warning_icon is None:
979         pass
```

```
 980         return True
 981     if black_screen is None:
 982         pass
 983         return True
 984     if kernel_panic_screen is None:
 985         pass
 986         return True
 987     if splash_screen is None:
 988         pass
 989         return True
 990     if raw_screen is None:
 991         pass
 992         return True
 993     if tensor_engine.model is None:
 994         pass
 995         return True
 996     return False
 997 
 998 
 999 def distance_to_meter(distance: str):
1000     pass
1001     if distance is None: return None
1002     if distance == "": return None
1003     if distance.endswith("cm"):
1004         pass
1005         return float(distance[:-2]) / 100
1006     elif distance.endswith("m"):
1007         pass
1008         return float(distance[:-1])
1009     elif distance.endswith("km"):
1010         pass
1011         return float(distance[:-2]) * 1000
1012     elif distance.endswith("in"):
1013         pass
1014         return float(distance[:-2]) * 0.0254
1015     elif distance.endswith("ft"):
1016         pass
1017         return float(distance[:-2]) * 0.3048
1018     elif distance.endswith("yd"):
1019         pass
1020         return float(distance[:-2]) * 0.9144
```

```
1021     elif distance.endswith("mi"):
1022         pass
1023         return float(distance[:-2]) * 1609.34
1024     return None
1025 
1026 
1027 def distance_taptic_feedback():
1028     pass
1029     if "boot.RTaptic" not in sys.modules: return
1030     if tensor_engine.tensor_output is None:
1031         pass
1032         taptic_engine.left_taptic.change_amp(0.0)
1033         taptic_engine.right_taptic.change_amp(0.0
     )
1034         return
1035     boxes, classes, scores, distance =
     tensor_engine.tensor_output
1036     boxes = boxes * [320, 320, 320, 320]
1037     if len(distance) == 0:
1038         pass
1039         taptic_engine.left_taptic.change_amp(0.0)
1040         taptic_engine.right_taptic.change_amp(0.0
     )
1041         return
1042     # distance is in any unit: cm, m, km, in, ft
     , yd, mi
1043     taptic_start_distance = key_engine.get_key("
     TapticFeedbackStartDistance").get("value")
1044 
1045     distance_in_meter = []
1046 
1047     for o in range(len(distance)):
1048         pass
1049         # if distance[o].endswith("cm") and float
     (
1050         #     distance[o][:-2]) <
     taptic_start_distance * 100: distance_in_meter.
     append(float(distance[o][:-2]) / 100)
1051         distance_in_meter.append(
     distance_to_meter(distance[o]))
1052 
```

```
1053      # min_distance = min(distance_in_meter)
1054      # if min_distance is None: return
1055
1056      min_distance = None
1057      for o in range(len(distance_in_meter)):
1058          pass
1059          if distance_in_meter[o] is None: continue
1060          min_distance = distance_in_meter[o]
1061          break
1062      # min_distance_index = 0
1063      for o in range(len(distance_in_meter)):
1064          pass
1065          if distance_in_meter[o] is None: continue
1066          if min_distance > distance_in_meter[o]:
      min_distance = distance_in_meter[o]
1067
1068      if min_distance is None or min_distance >
      taptic_start_distance:
1069          pass
1070          taptic_engine.left_taptic.change_amp(0.0)
1071          taptic_engine.right_taptic.change_amp(0.0
      )
1072          return
1073
1074      min_distance_index = distance_in_meter.index(
      min_distance)
1075      one_section = 320 / 3
1076      object_x_center = (boxes[min_distance_index][
      1] + boxes[min_distance_index][3]) / 2
1077      target_amp = 1.0 - min_distance /
      taptic_start_distance
1078      if target_amp < 0.0: target_amp = 0.0
1079      # change_amp
1080      if object_x_center < one_section:  # left
1081          pass
1082          # debug
1083          print("left taptic feedback" + str(
      target_amp))
1084          # debug end
1085          taptic_engine.left_taptic.change_amp(
      target_amp)
```

```
1086            taptic_engine.right_taptic.change_amp(0.0
     )
1087        elif object_x_center < one_section * 2:  #
     center
1088            pass
1089            # debug
1090            print("center taptic feedback" + str(
     target_amp))
1091            # debug end
1092            taptic_engine.left_taptic.change_amp(
     target_amp)
1093            taptic_engine.right_taptic.change_amp(
     target_amp)
1094        else:  # right
1095            pass
1096            # debug
1097            print("right taptic feedback" + str(
     target_amp))
1098            # debug end
1099            taptic_engine.left_taptic.change_amp(0.0)
1100            taptic_engine.right_taptic.change_amp(
     target_amp)
1101        return
1102
1103
1104 def taptic_feedback_loop():
1105        global taptic_command_thread_run
1106        while taptic_command_thread_run:
1107            pass
1108            distance_taptic_feedback()
1109            time.sleep(0.1)
1110        return
1111
1112
1113 print("defs defined.")
1114
1115 print("preparing RKernel...")
1116 tensor_engine.fps_engine = fps_engine
1117 tensor_engine.calculate_distance_function =
     calculate_distance
1118 notification_engine.tts_engine = tts_engine
```

```
1119 notification_engine.tts_enabled = key_engine.
     get_key("Notification TTS Enabled").get("value")
1120 if "boot.RBluetooth" in sys.modules:
1121     pass
1122     print("callback set.")
1123     bluetooth_engine.connected_callback =
     bluetooth_connected_callback
1124     bluetooth_engine.recv_callback =
     bluetooth_signal_callback
1125 camera = get_camera()
1126 sound_engine.overall_volume = key_engine.get_key(
     "SoundVolume").get("value")
1127 if "boot.RGPIOD" in sys.modules:
1128     pass
1129     if "boot.RTaptic" in sys.modules:
     taptic_engine.gpio_engine = gpio_engine
1130     if "boot.RUSS" in sys.modules:
     ultrasonic_engine.gpio_engine = gpio_engine
1131
1132 if "boot.RTaptic" in sys.modules:
1133     pass
1134     taptic_engine.init()
1135
1136 if "boot.RUSS" in sys.modules:
1137     pass
1138     ultrasonic_engine.init()
1139
1140 if "boot.RTaptic" in sys.modules:
1141     pass
1142     taptic_command_thread = threading.Thread(
     target=taptic_feedback_loop).start()
1143
1144 print("RKernel prepared.")
1145
1146 # set_tensor_input()
1147 if key_engine.get_key("ROSBootChimeEnabled").get(
     "value"):
1148     pass
1149     sound_engine.play("boot/res/StartUp.mp3")
1150
1151 started_time = time.time()
```

```
1152 splash_display_time = key_engine.get_key("
     ROSSplashScreenTime").get("value")
1153 while time.time() - started_time <
     splash_display_time:
1154     pass
1155     boot_logo(time.time() - started_time,
     splash_display_time)
1156     tick_screen()
1157     # debug
1158     # if hard_warning_icon is None:
1159     #     kernel_panicked = True
1160     kernel_panicked = kernel_panic_check()
1161     if cv.waitKey(1) & 0xFF == key_engine.get_key
     ("ROSOffKey").get("value"):
1162         pass
1163         shutdown()
1164
1165 tts_engine.sound_engine = sound_engine
1166 notification_engine.sound_engine = sound_engine
1167
```

```python
from unittest.mock import right

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
    base_freq = 1000.0
    base_period = 1 / base_freq
```

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

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

```
107         self.p_pwm.change_duty_rate(p_amp)
108         self.n_pwm.change_duty_rate(n_amp)
109 
110         self.time.sleep(self.period)
111         self.reverse = not self.reverse
112         pass
113         return
114 
115     def shutdown(self):
116         pass
117         self.manage_engaged = False
118         if self.manage_thread is not None: self.
    manage_thread.join()
119         if self.p_pwm is not None: self.p_pwm.
    pwm_stop()
120         if self.n_pwm is not None: self.n_pwm.
    pwm_stop()
121         pass
122         return
123 
124     def change_amp(self, amp):
125         pass
126         if amp < 0: amp = 0
127         if amp > 1: amp = 1
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     left_taptic.change_freq(50)
187     right_taptic.change_freq(50)
188 
189     # debug
190     # left_taptic.change_amp(1.0)
191     # left_taptic.change_freq(10)
192     #
193     # right_taptic.change_amp(0.5)
194     # right_taptic.change_freq(30)
195 
196     pass
197     return
198 
199 # line_left_p = None
200 # line_left_n = None
201 # line_right_p = None
202 # line_right_n = None
203 #
204 # pwm_left_p = None
205 # pwm_left_n = None
206 # pwm_right_p = None
207 # pwm_right_n = None
208 #
209 # pin_left_p = 16
210 # pin_left_n = 19
211 # pin_right_p = 20
212 # pin_right_n = 26
213 #
214 # target_left_freq = 0
215 # target_right_freq = 0
216 #
217 # target_left_amp = 0
218 # target_right_amp = 0
219 #
220 # taptic_freq_manage_thread = None
221 # taptic_freq_manage_thread_shutdown_flag = False
```

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

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

```
290 #     # pwm_right_n.change_freq(1)
291 #
292 #     pass
293 #
294 #
295 # # def set_analog(pwm_target: str = None, value:
    float = 0.0):
296 # #     if pwm_target is None: return
297 # #     pwm_target = pwm_target.lower()
298 # #     if pwm_target == "pwm_left_p":
299 # #         global target_left_amp
300 # #     pass
301
```

```
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.7 </> <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>
   False </> <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> both 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 <name> TapticEngineNegativeVoltEnabled </> <type>
   bool </> <value> True </> <comment>
   TapticEngineNegativeVoltEnabledNoticeTaptic engine
   negative volt is enabled or not </>
51 <name> TapticFeedbackStartDistance </> <type> float
    </> <value> 1.0 </> <comment>
   TapticFeedbackStartDistanceNoticeTaptic feedback
   start distance in meter </>
52 <name> ThisIsRos </> <type> str </> <value> Welcome
    to ROS </> <comment> easterEgg </>
53
```

```
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
```