```
print("ROS is booting up...")
try:
    pass
    import boot.RKernel as kernel
except ImportError as ie:
    pass
    print("RKernel not found or failed to load.")
    print(ie)
    exit(11)
except Exception as e:
    pass
    print("RKernel not found or failed to load.")
    print(e)
    exit(12)

print("ROS booted up.")

# import time

# debug_start_time = time.time()

while True:
    pass
    frame = kernel.get_frame()
    kernel.raw_screen = frame
    kernel.set_tensor_input()
    kernel.render_tensor_and_etc()
    kernel.tick_screen()
    if kernel.cv.waitKey(1) & 0xFF == kernel.key_engine.get_key("ROSOffKey").get("value"): break

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

kernel.shutdown()
```

```
 1 # **rOS EdgeRunner**
 2
 3 ## 오픈 소스 소개
 4
 5 rOS는 모든 하위 버전을 포함하여 오픈소스 프로젝트입니다. 누구나 코드를
    참고하거나 수정할 수 있습니다.
 6
 7 rOS를 어떤 방식으로든 사용할 경우, 다음 조건을 준수해야 합니다:
 8
 9 > ### A. 출처 표기
10 > rOS 코드를 사용하는 모든 프로젝트와 발표에서는 **반드시 출처를
   명시**해야 합니다. 출처를 명시하지 않을 경우, 코드에 대한 사용 권한을
    상실하게 되며, 이를 기반으로 한 **소유권을 주장할 수 없습니다**.
11 > 출처 표시는 다음 위치에 포함되어야 합니다:
12 > - 코드 파일 주석
13 > - README 파일
14 > - 프로젝트의 사용자 인터페이스(해당하는 경우)
15 > - 학교 또는 학술 발표 자료
16 >
17 > ---
18 >
19 > ### B. 비상업적 및 윤리적 사용
20 > rOS는 기본적으로 모두에게 오픈소스 프로젝트로 제공됩니다. 그러나
    **상업적 목적**이나 **비윤리적인 목적**으로 사용되는 것을
   금지합니다. 상업적 사용을 원할 경우, devMaxTrauma Inc.와
   개발자의 사전 서면 동의가 필요합니다.
21 >
22 > ---
23 >
24 > ### C. 수정 코드에 대한 소유권
25 > rOS 코드를 참고하거나 기반으로 수정한 코드를 개발한 모든 개인 또는
    단체는 해당 코드에 대해 **devMaxTrauma. Inc.** 와 개발자가
   소유권을 행사할 수 있습니다. 수정된 코드는 동일한 라이선스 조건을
   따라야 하며, **클로즈드소스**로 전환할 수 없습니다.
26 >
27 > ---
28 >
29 > ### D. 책임 면책
30 > 이 프로젝트의 코드는 **있는 그대로 제공**되며, 사용 중 발생할 수
   있는 문제에 대해 어떠한 책임도 지지 않습니다. 코드 사용으로 인해
   발생하는 법적 문제, 손해, 또는 기타 불이익에 대해 **개발자는 책임을
```

지지 않으며**, 모든 책임은 사용자에게 있습니다.
>
> ---
>
> ### E. 보증 없음
> rOS 코드는 **어떠한 보증도 제공하지 않습니다**. 이 코드는 명시적이든 묵시적이든 특정 목적에 대한 보증 없이 제공됩니다. 코드가 기대한 대로 동작하지 않거나, 문제가 발생해도 개발자는 이에 대한 책임이 없습니다.
>
> ---
>
> ### F. 배포 조건
> rOS 코드를 수정하거나 배포할 경우, 이 라이선스 조건을 **동일하게 유지**해야 합니다. 이 프로젝트에 기여하는 모든 코드 역시 동일한 조건에 따라 배포되어야 합니다.
>
> ---
>
> ### G. 동의
> rOS를 사용하는 경우, 위의 모든 조건에 동의하는 것으로 간주합니다. 출처를 명시하지 않은 경우, 사용자는 해당 코드의 권리 및 소유권을 주장할 수 없으며, 법적 제재의 대상이 될 수 있습니다.

rOS EdgeRunner 소개
rOS EdgeRunner는 이름에서 EdgeRunner를 보고 알 수 있듯, Ayaka, Bleach, Clannad, Demonslayer를 이은 5번째 메이저 버전의 rOS입니다.

다른 버전의 rOS도 git branch로 확인할 수 있습니다.

rOS는 모두 Python기반으로 작성되었으며 RaspberryPI OS와 **macOS**, Windows에서 모두 필요 라이브러리만 설치하면 사용할 수 있습니다.

rOS의 구성은 ROS.py, RKernel.py, 그외 R모듈.py로 이루어져 있습니다.

기술 설명
rOS는 각 메이저 버전업마다 신기술을 적용해왔고 업데이트된 내용은

업데이트명 Target.txt 파일로 확인 가능합니다.

rOS EdgeRunner의 DemonSlayer대비 업데이트된 내용은 다음과 같습니다.
> - OpenCV와 TensorFlow의 Thread를 분리하여 성능을 향상시켰습니다.
>
> - 프로젝트 구조를 변경하여 더욱 효율적인 코드를 작성할 수 있도록 했습니다.
>
> - 부팅시 심심하지 않도록 새로운 스플래쉬 스크린을 적용했습니다.
>
> - 초음파센서를 활성화 했습니다.
>
> - RGPIOD.py를 통하여 자체적인 PWM을 구현했습니다.

rOS는 라이센스비용을 줄이고자 자체개발한 Python IDE인 [SIDE](https://github.com/ellystargram/SIDE)를 이용하여 개발되었습니다.

기능
rOS EdgeRunner는 다음과 같은 기능을 제공합니다.
> - **사물 구분**: Tensorflow Lite를 통해 사물을 구분할 수 있습니다.
>
> - **사물 거리 측정**: 초음파센서와 카메라비전을 이용하여 물체와 떨어진 거리를 측정할 수 있습니다.
>
> - **간편한 설정 공유**: RKey.RKY 파일을 통해 설정을 공유할 수 있습니다. 이 파일은 누구나 알아보기 쉽도록 제작된 자체개발한 UTF-8 기반의 파일입니다.
>
> - **RGPIOD**: 자체개발한 RGPIOD를 통해 GPIO를 쉽게 제어할 수 있습니다. PWM도 지원합니다.
>
> - **TTS**: TTS를 이용해 경고 메시지를 사용자에게 알려줍니다. 저조도주의나 성능저하주의가 대표적인 예입니다.
>
> - **안드로이드 기기 연결**: 안드로이드 기기와 블루투스로 연결한 다음 [SLD](https://github.com/devMaxTrauma/SLD)를 통해 FindMy 기능이나 색약 색보정이 가능합니다.

```
86 >
87 > - **Taptic Engine**: Taptic Engine을 통해 사용자에게 진동을 전달할 수 있습니다.
88 >
89 > - **AR**: AR을 통해 사용자에게 보여줍니다. 이를위해 이미지를 양안에 맞게 처리할 수 있는 능력이 있습니다.
90
91 ## 사용법
92 ROS.py가 있는 Directory에서 ROS.py를 실행하면 됩니다.
93
94 > macOS
95 >```bash
96 >~/rOS $ python3 ROS.py
97 >```
98 ---
99 >Linux
100 >```bash
101 >~/rOS $ python ROS.py
102 >```
103 ---
104 >Windows
105 >```cmd
106 >C:\rOS> python ROS.py
107 >```
108
109 ## 사용한 라이브러리
110 rOS EdgeRunner는 다음과 같은 라이브러리를 사용합니다.
111 > - **OpenCV**: 그래픽 렌더를 위해 사용합니다.
112 > - **TensorFlow Lite**: 사물 구분을 위해 사용합니다.
113 > - **Pygame**: RSound.py를 위해 사용합니다.
114 > - **threading**: TensorFlow Lite와 OpenCV를 분리하기 위해, 멀티스레딩을 위해 사용합니다.
115 > - **time**: 초음파센서와 카메라비전을 위해 사용합니다. 또한 자원절약을 위해 사용합니다.
116 > - **gpiod**: GPIO를 제어하기 위해 RGPIOD에서 사용합니다.
117 > - **picamera2**: 카메라를 사용하기 위해 사용합니다.
118 > - **numpy**: TensorFlow Lite에서 사용합니다.
119 > - **sys**: 시스템을 제어하기 위해 사용합니다.
120 > - **os**: 시스템을 제어하기 위해 사용합니다.
121
122 ## 놀라운 사실
```

rOS EdgeRunner는 다음과 같은 놀라운 사실을 가지고 있습니다.
> - 모든 소스코드줄의 합은 3000줄을 가볍게 넘습니다.
> - rOS EdgeRunner는 github Copilot과 함께 개발되었습니다.
> - 라즈베리파이로 코드를 옮길땐 github repository를 이용하여 코드를 옮겼습니다.

[//]: # (## Introduction)

[//]: # (* rOS is Software for Blind Navigation and Object Detection.)

[//]: # (* rOS is based on python and uses OpenCV and TensorFlow for Object Detection.)

[//]: # (* rOS can calculate distance between user and object.)

[//]: # (* rOS tensorflow and opencv is seperated from main thread to increase performance.)

[//]: # ()
[//]: # (## How to use)

[//]: # (* run ROS.py)

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

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

```python
import time

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


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


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


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

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

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


def read_key_line(key_line):
    pass
    global keys
    if key_line == "\n": return
    # key format: <name> name </> <type> str </> <value> hello, world! </> <comment> This is a comment. </>
    key_data = key_line.split("</>")
    key_name = key_data[0].split("<name>")[1].strip()
    key_type = key_data[1].split("<type>")[1].strip()
    key_value = key_data[2].split("<value>")[1].strip()
    key_comment = key_data[3].split("<comment>")[1].strip()
    if key_type == "int":
        pass
        key_value = int(key_value)
        key_type = type(key_value)
    elif key_type == "float":
        pass
        key_value = float(key_value)
        key_type = type(key_value)
    elif key_type == "bool" and key_value in positive_signs:
        pass
        key_value = True
        key_type = type(key_value)
    elif key_type == "bool" and key_value in negative_signs:
        pass
        key_value = False
        key_type = type(key_value)
```

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

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

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

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

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

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

sound_engine = None
generator_working = True
tts_order_list = []


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

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

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

gpio_engine = None

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


def shutdown():
    pass
    global ultra_sonic_sensor_read_enabled
    ultra_sonic_sensor_read_enabled = False

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


def init():
    pass
```

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

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

```
# try:
#     import RPi.GPIO as GPIO
# except ImportError:
#     raise ImportError("RPi.GPIO not found. Please install it using 'sudo apt-get install python3-rpi.gpio'.")
# try:
#     import threading
# except ImportError:
#     raise ImportError("threading not found. Please install it using 'pip install threading'.")
# try:
#     import time
# except ImportError:
#     raise ImportError("time not found. Please install it using 'pip install time'.")
#
# pwms = {}
#
# try:
#     GPIO.setmode(GPIO.BCM)
# except Exception as e:
#     print("Failed to set GPIO mode.")
#     print(e)
#     raise e
#
#
# def set_output(pin):
#     GPIO.setup(pin, GPIO.OUT)
#
#
# def set_input(pin):
#     GPIO.setup(pin, GPIO.IN)
#
#
# def output_write(pin, state):
#     GPIO.output(pin, state)
#
#
# def input_read(pin):
#     return GPIO.input(pin)
```

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

```
colors = {}


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


def __load_colors__():
    pass
    try:
        pass
        r_color_file = open("boot/res/RClassColor.RCC", "r", encoding="utf-8")
        color_list = r_color_file.readlines()
        r_color_file.close()
        for one_color in color_list: __read_color_save_line__(one_color)
    except FileNotFoundError:
        pass
        print("RColor.RCL not found.")
        return
    except Exception as e:
        pass
        print("Error loading colors.")
        print(e)
        exit(999)

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

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

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

```
 1 try:
 2     pass
 3     import gpiod
 4 except ImportError:
 5     pass
 6     raise ImportError("gpiod not found. Please
   install it using 'pip install gpiod'.")
 7 try:
 8     pass
 9     import time
10 except ImportError:
11     pass
12     raise ImportError("time not found. Please
   install it using 'pip install time'.")
13 try:
14     pass
15     import threading
16 except ImportError:
17     pass
18     raise ImportError("threading not found. Please
   install it using 'pip install threading'.")
19
20 chip = gpiod.Chip("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
```

```python
key_engine = None
mobile_filter = "normal"


def adjust_frame(frame, boost, ratio):
    pass
    frame_copy = frame.astype("float32")

    # frame[:, :, 2] = frame[:, :, 2] * ratio[0] + boost[0]
    # frame[:, :, 1] = frame[:, :, 1] * ratio[1] + boost[1]
    # frame[:, :, 0] = frame[:, :, 0] * ratio[2] + boost[2]
    frame_copy[:, :, 2] = frame[:, :, 2] * ratio[0] + boost[0]
    frame_copy[:, :, 1] = frame[:, :, 1] * ratio[1] + boost[1]
    frame_copy[:, :, 0] = frame[:, :, 0] * ratio[2] + boost[2]

    frame_copy[frame_copy > 255] = 255
    frame_copy[frame_copy < 0] = 0

    frame = frame_copy.astype("uint8")

    # frame[frame > 255] = 255
    # frame[frame < 0] = 0

    return frame


def black_white(frame):
    pass
    frame = frame.astype("float32")
    frame[:, :, 2] = frame[:, :, 2] * 0.299 + frame[:, :, 1] * 0.587 + frame[:, :, 0] * 0.114
    frame[:, :, 1] = frame[:, :, 2]
    frame[:, :, 0] = frame[:, :, 2]
    frame[frame > 255] = 255
    frame[frame < 0] = 0
```

```
35      return frame.astype("uint8")
36
37
38  def red_weak(frame):
39      pass
40      return adjust_frame(frame, [-10, 5, 0], [0.6, 1
    .3, 1.0])
41
42
43  def green_weak(frame):
44      pass
45      return adjust_frame(frame, [5, -10, 0], [0.9, 0
    .4, 1.0])
46
47
48  def blue_weak(frame):
49      pass
50      return adjust_frame(frame, [0, -10, -10], [1.0
    , 0.9, 0.6])
51
52
53  def all_weak(frame):
54      pass
55      return black_white(frame)
56
57
58  def color_adjust(frame):
59      pass
60      if key_engine is None:
61          pass
62          print("ColorFilterMode: key_engine is None
    .")
63          return frame
64
65      blue_light_filter_enabled = key_engine.get_key(
    "BlueLightFilterEnabled").get("value")
66      if blue_light_filter_enabled:
67          pass
68          blue_light_filter_strength = key_engine.
    get_key("BlueLightFilterStrength").get("value")
69          frame = adjust_frame(frame, [0, 0, 0], [1.0
```

```
69 , 1.0, 1.0 - blue_light_filter_strength])
70
71     color_adjust_mode = key_engine.get_key("ColorFilterMode").get("value")
72     if color_adjust_mode == "sync mobile":
73         pass
74         if mobile_filter is None: return frame
75         if mobile_filter == "red_weak": return red_weak(frame)
76         if mobile_filter == "green_weak": return green_weak(frame)
77         if mobile_filter == "blue_weak": return blue_weak(frame)
78         if mobile_filter == "all_weak": return all_weak(frame)
79         return frame
80     elif color_adjust_mode == "custom":
81         pass
82         red_boost = key_engine.get_key("ColorFilterRedBoost").get("value")
83         green_boost = key_engine.get_key("ColorFilterGreenBoost").get("value")
84         blue_boost = key_engine.get_key("ColorFilterBlueBoost").get("value")
85
86         red_ratio = key_engine.get_key("ColorFilterRedRatio").get("value")
87         green_ratio = key_engine.get_key("ColorFilterGreenRatio").get("value")
88         blue_ratio = key_engine.get_key("ColorFilterBlueRatio").get("value")
89
90         frame = adjust_frame(frame, [red_boost, green_boost, blue_boost], [red_ratio, green_ratio, blue_ratio])
91
92         # frame[:, :, 2] = frame[:, :, 2] * red_ratio + red_boost
93         # frame[:, :, 1] = frame[:, :, 1] * green_ratio + green_boost
94         # frame[:, :, 0] = frame[:, :, 0] *
```

```
 94 blue_ratio + blue_boost
 95         # for x in range(320):
 96         #     pass
 97         #     for y in range(320):
 98         #         pass
 99         #         frame[y, x, 2] = frame[y, x, 2
    ] * red_ratio + red_boost
100         #         frame[y, x, 1] = frame[y, x, 1
    ] * green_ratio + green_boost
101         #         frame[y, x, 0] = frame[y, x, 0
    ] * blue_ratio + blue_boost
102     return frame
103
```

```python
from boot.RTensor import model

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

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


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


print("methods pre def defined.")

print("Loading Third Party imports...")
try:
    pass
    import cv2 as cv
except ImportError:
    pass
    make_error("1001", "cv2 not found.")
except Exception as e:
    pass
    make_error("1001-1", str(e))
try:
    pass
    import time
except ImportError:
    pass
    make_error("1002", "time not found.")
except Exception as e:
    pass
    make_error("1002-1", str(e))
try:
    pass
    import picamera2
except ImportError:
    pass
    print("Picamera2 not found.")
```

```
41 except Exception as e:
42     pass
43     make_error("1003", str(e))
44 try:
45     pass
46     import threading
47 except ImportError:
48     pass
49     make_error("1005", "threading not found.")
50 except Exception as e:
51     pass
52     make_error("1005-1", str(e))
53 try:
54     pass
55     import numpy as np
56 except ImportError:
57     pass
58     make_error("1006", "numpy not found.")
59 except Exception as e:
60     pass
61     make_error("1006-1", str(e))
62 try:
63     pass
64     import sys
65 except ImportError:
66     pass
67     make_error("1007", "sys not found.")
68 except Exception as e:
69     pass
70     make_error("1007-1", str(e))
71 try:
72     pass
73     import os
74 except ImportError:
75     pass
76     make_error("1008", "os not found.")
77 except Exception as e:
78     pass
79     make_error("1008-1", str(e))
80 print("Third Party Imports loaded.")
81
```

```
 82 print("Loading RKernel imports...")
 83 try:
 84     pass
 85     import boot.RKey as key_engine
 86 except ImportError:
 87     pass
 88     make_error("1101", "RKey not found.")
 89 except Exception as e:
 90     pass
 91     make_error("1101-1", str(e))
 92 try:
 93     pass
 94     import boot.RSound as sound_engine
 95 except ImportError:
 96     pass
 97     make_error("1102", "RSound not found.")
 98 except Exception as e:
 99     pass
100     make_error("1102-1", str(e))
101 try:
102     pass
103     import boot.RFPS as fps_engine
104 except ImportError:
105     pass
106     make_error("1103", "RFPS not found.")
107 except Exception as e:
108     pass
109     make_error("1103-1", str(e))
110 try:
111     pass
112     import boot.RTensor as tensor_engine
113 except ImportError:
114     pass
115     make_error("1104", "RTensor not found.")
116 except Exception as e:
117     pass
118     make_error("1104-1", str(e))
119 try:
120     pass
121     import boot.RLabel as label_engine
122 except ImportError:
```

```
123     pass
124     make_error("1105", "RLabel not found.")
125 except Exception as e:
126     pass
127     make_error("1105-1", str(e))
128 try:
129     pass
130     import boot.RColor as color_engine
131 except ImportError:
132     pass
133     make_error("1106", "RColor not found.")
134 except Exception as e:
135     pass
136     make_error("1106-1", str(e))
137 try:
138     pass
139     import boot.RBluetooth as bluetooth_engine
140 except ImportError as e:
141     pass
142     print("RBluetooth not found.")
143 except Exception as e:
144     pass
145     make_error("1107", str(e))
146 try:
147     pass
148     import boot.RNotification as
    notification_engine
149 except ImportError:
150     pass
151     make_error("1108", "RNotification not found.")
152 except Exception as e:
153     pass
154     make_error("1108-1", str(e))
155 try:
156     pass
157     import boot.RTTS as tts_engine
158 except ImportError:
159     pass
160     make_error("1109", "RTTS not found.")
161 except Exception as e:
162     pass
```

```
163     make_error("1109-1", str(e))
164 # try:
165 #     import boot.RGPIO as gpio_engine
166 # except ImportError:
167 #     print("RGPIO not found.")
168 # try:
169 #     if "boot.RGPIO" in sys.modules: import boot.
    RUSS as ultrasonic_engine
170 # except ImportError:
171 #     make_error("1111", "RUSS not found.")
172 # try:
173 #     if "boot.RGPIO" in sys.modules: import boot.
    RTaptic as taptic_engine
174 # except ImportError:
175 #     make_error("1112", "RTaptic not found.")
176 try:
177     pass
178     import boot.RGPIOD as gpio_engine
179 except ImportError:
180     pass
181     print("RGPIOD not found.")
182 except Exception as e:
183     pass
184     make_error("1110", str(e))
185 try:
186     pass
187     if "boot.RGPIOD" in sys.modules: import boot.
    RUSS as ultrasonic_engine
188 except ImportError:
189     pass
190     make_error("1111", "RUSS not found.")
191 except Exception as e:
192     pass
193     make_error("1111-1", str(e))
194 try:
195     pass
196     if "boot.RGPIOD" in sys.modules: import boot.
    RTaptic as taptic_engine
197 except ImportError:
198     pass
199     make_error("1112", "RTaptic not found.")
```

```
200 except Exception as e:
201     pass
202     make_error("1112-1", str(e))
203 try:
204     pass
205     import boot.RFilter as filter_engine
206 except ImportError:
207     pass
208     make_error("1113", "RFilter not found.")
209 except Exception as e:
210     pass
211     make_error("1113-1", str(e))
212 
213 print("RKernel imports loaded.")
214 
215 print("defining variables...")
216 splash_screen = cv.imread("boot/res/apple_logo.png
    ")
217 screen = splash_screen
218 raw_screen = splash_screen
219 black_screen = cv.imread("boot/res/black_screen.
    jpg")
220 kernel_panic_screen = cv.imread("boot/res/
    kernel_panic.png")
221 kernel_panicked = False
222 camera = None
223 find_my_keep_sounding_channel = None
224 find_my_sounding_one_channel = None
225 find_my_notification = None
226 boot_loading_bar = 0
227 hard_warning_icon = cv.imread("boot/res/
    hard_warning.png")
228 warning_icon = cv.imread("boot/res/warning.png")
229 taptic_command_thread = None
230 taptic_command_thread_run = True
231 color_weakness_compensation_notification = None
232 print("variables defined.")
233 
234 print("defining defs...")
235 
236 
```

```
237 def get_320_320_frame(raw_frame):
238     pass
239     if raw_frame is None:
240         pass
241         global kernel_panicked
242         kernel_panicked = True
243         raw_frame = kernel_panic_screen
244     # make sure the frame is 320x320 and save it
    to raw_frame
245     if raw_frame.shape[0] != 320 or raw_frame.
    shape[1] != 320:
246         pass
247         # make new_frame but don't flect it
248         target_width = 320
249         target_height = 320
250         aspect_ratio = float(target_height) /
    raw_frame.shape[0]
251         dsize = (int(raw_frame.shape[1] *
    aspect_ratio), target_height)
252         raw_frame = cv.resize(raw_frame, dsize)
253 
254         # cut the new_frame width to 320 center
255         raw_frame = raw_frame[:,
256                     raw_frame.shape[1] // 2 -
    target_width // 2: raw_frame.shape[1] // 2 +
    target_width // 2]
257     return raw_frame
258 
259 
260 def make_window():
261     pass
262     cv.namedWindow("ROS", cv.WINDOW_NORMAL)
263     if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
264         pass
265         ar_width = key_engine.get_key("
    ARDisplayWidth").get("value")
266         ar_height = key_engine.get_key("
    ARDisplayHeight").get("value")
267         cv.resizeWindow("ROS", ar_width, ar_height
    )
```

```
268     else:
269         pass
270         cv.resizeWindow("ROS", 320, 320)
271     return
272 
273 
274 def get_camera():
275     pass
276     if "picamera2" in sys.modules:
277         pass
278         camera = picamera2.Picamera2()
279         camera.configure(camera.
    create_preview_configuration(main={"size": (320,
    320)}))
280         camera.start()
281     else:
282         pass
283         camera = cv.VideoCapture(0)
284     return camera
285 
286 
287 def align_camera_frame(frame):
288     pass
289     camera_eye_location = key_engine.get_key("
    RaspberryPiCameraEye").get("value")
290     if camera_eye_location == "left": return
    rotate_clockwise_90(frame)
291     if camera_eye_location == "right": return
    rotate_counterclockwise_90(frame)
292     return frame
293 
294 
295 def get_frame():
296     pass
297     global kernel_panicked
298     if kernel_panicked:
299         pass
300         return get_320_320_frame(
    kernel_panic_screen)
301     global camera
302     if "picamera2" in sys.modules:
```

```
303         pass
304         frame = camera.capture_array()
305         frame = cv.cvtColor(frame, cv.
    COLOR_BGR2RGB)
306         camera_eye_location = key_engine.get_key("
    RaspberryPiCameraEye").get("value")
307         frame = align_camera_frame(frame)
308     else:
309         pass
310         frame = camera.read()[1]
311 
312     if frame is None:
313         pass
314         print("camera read failed")
315         kernel_panicked = True
316         return None
317 
318     # check brightness
319     frame_average_red = np.average(frame[:, :, 2])
320     frame_average_green = np.average(frame[:, :, 1
    ])
321     frame_average_blue = np.average(frame[:, :, 0
    ])
322     frame_average_brightness = (frame_average_red
     + frame_average_green + frame_average_blue) / 3
323     if frame_average_brightness < 5 and
    notification_engine.get_notification(
324             message="low brightness detected") is
    None:
325         pass
326         notification_engine.add_notification("
    hard_warning.png", "low brightness detected",
327                                              "저조도
     감지. 사용자의 즉각적인 주의가 필요합니다.")
328     elif frame_average_brightness >= 250 and
    notification_engine.get_notification(
329             message="high brightness detected") is
     None:
330         pass
331         notification_engine.add_notification("
    hard_warning.png", "high brightness detected",
```

```
332                                                     "고조도
    감지. 사용자의 즉각적인 주의가 필요합니다.")
333
334     # make frame 320x320
335     return get_320_320_frame(frame)
336
337
338 # def calculate_distance(class_index: int,
    box_width_pixel):
339 def calculate_distance(class_index: int, box):
340     pass
341     box_start_x = box[1]
342     box_end_x = box[3]
343     box_start_y = box[0]
344     box_end_y = box[2]
345     if label_engine.get_label(class_index + 1) is
    None: return str("N/A")
346     object_average_width = label_engine.get_label(
    class_index + 1).get("average_width")
347
348     # check if the object is in the center of the
    frame and can be calculated by vision
349     can_calculated_by_vision = True
350     vision_side_margin = key_engine.get_key("
    ROSObjectDistanceCalculateSideErrorMarginByVision"
    ).get("value")
351     x_check = key_engine.get_key("
    ROSObjectDistanceCalculateYAxisOutOfBoundCheckEnab
    led").get("value")
352     y_check = key_engine.get_key("
    ROSObjectDistanceCalculateXAxisOutOfBoundCheckEnab
    led").get("value")
353     if (box_start_x <= vision_side_margin or
    box_end_x >= (320 - vision_side_margin)) and
    x_check:
354         pass
355         can_calculated_by_vision = False
356     if (box_start_y <= vision_side_margin or
    box_end_y >= (320 - vision_side_margin)) and
    y_check:
357         pass
```

```
358         can_calculated_by_vision = False
359 
360     can_calculated_by_uss = False
361     if "boot.RUSS" in sys.modules:
362         pass
363         # check if the object is in the center of
    the frame and can be calculated by uss
364         uss_region_start_x = key_engine.get_key("
    USSRegionStartX").get("value")
365         uss_region_end_x = key_engine.get_key("
    USSRegionEndX").get("value")
366         uss_region_start_y = key_engine.get_key("
    USSRegionStartY").get("value")
367         uss_region_end_y = key_engine.get_key("
    USSRegionEndY").get("value")
368 
369         inside_uss_x_region = False
370         inside_uss_y_region = False
371         if box_end_x >= uss_region_start_x and
    box_start_x <= uss_region_end_x:
    inside_uss_x_region = True
372         if box_end_y >= uss_region_start_y and
    box_start_y <= uss_region_end_y:
    inside_uss_y_region = True
373 
374         can_calculated_by_uss =
    inside_uss_x_region and inside_uss_y_region
375         # pass
376 
377     # check camera system and get calculate
    constant
378     if "picamera2" in sys.modules:
379         pass
380         distance_calculate_constant = key_engine.
    get_key("
    ROSObjectDistanceCalculateConstantRaspberryPi").
    get("value")
381     else:
382         pass
383         distance_calculate_constant = key_engine.
    get_key("
```

```
383 ROSObjectDistanceCalculateConstantMacBookPro").get
    ("value")
384 
385     vision_distance_in_meter =
    object_average_width / ((box_end_x - box_start_x
    ) / distance_calculate_constant)
386     uss_distance_in_meter = 0.0
387     if "boot.RUSS" in sys.modules:
    uss_distance_in_meter = ultrasonic_engine.
    output_distance
388     if uss_distance_in_meter == -1.0:
    can_calculated_by_uss = False
389 
390     likely_distance_in_meter = 0.0
391     if not can_calculated_by_uss and not
    can_calculated_by_vision:
392         pass
393         return str("")
394     elif can_calculated_by_uss and not
    can_calculated_by_vision:
395         pass
396         likely_distance_in_meter =
    uss_distance_in_meter
397     elif not can_calculated_by_uss and
    can_calculated_by_vision:
398         pass
399         likely_distance_in_meter =
    vision_distance_in_meter
400     elif can_calculated_by_uss and
    can_calculated_by_vision:
401         pass
402         # I think vision is more precise
403         likely_distance_in_meter =
    vision_distance_in_meter
404 
405     unit = key_engine.get_key("DistanceUnit").get(
    "value")
406     if unit == "SI":
407         pass
408         if likely_distance_in_meter < 1: return
    str(round(likely_distance_in_meter * 100, 2)) + "
```

```
408 cm"
409         if likely_distance_in_meter <= 1000:
    return str(round(likely_distance_in_meter, 2)) +
    " m"
410         if likely_distance_in_meter > 1000: return
     str(round(likely_distance_in_meter / 1000, 2)) +
    " km"
411     if unit == "US":
412         pass
413         if likely_distance_in_meter < 0.3048:
    return str(round(likely_distance_in_meter * 39.
    3701, 2)) + " in"
414         if likely_distance_in_meter <= 0.9144:
    return str(round(likely_distance_in_meter * 3.
    28084, 2)) + " ft"
415         if likely_distance_in_meter <= 1609.34:
    return str(round(likely_distance_in_meter * 1.
    09361, 2)) + " yd"
416         if likely_distance_in_meter > 1609.34:
    return str(round(likely_distance_in_meter / 1609.
    34, 2)) + " mi"
417     pass
418     return str("ERR")
419     # # this code (below) is discarded
420     # if "picamera2" in sys.modules:
421     #     distance_calculate_constant = key_engine
    .get_key("
    ROSObjectDistanceCalculateConstantRaspberryPi").
    get("value")
422     # else:
423     #     distance_calculate_constant = key_engine
    .get_key("
    ROSObjectDistanceCalculateConstantMacBookPro").get
    ("value")
424     #
425     # if label_engine.get_label(class_index + 1)
    is None:
426     #     object_average_width = -1.0
427     # else:
428     #     object_average_width = label_engine.
    get_label(class_index + 1).get("average_width")
```

```
429     #
430     # if object_average_width == -1.0:
431     #     return str("N/A")
432     #
433     # distance_in_meter = object_average_width / (
    box_width_pixel / distance_calculate_constant)
434     # unit = key_engine.get_key("DistanceUnit").
    get("value")
435     # if unit == "SI" and distance_in_meter < 1:
436     #     return str(round(distance_in_meter * 100
    , 2)) + " cm"
437     # elif unit == "SI" and distance_in_meter <=
    1000:
438     #     return str(round(distance_in_meter, 2
    )) + " m"
439     # elif unit == "SI" and distance_in_meter >
    1000:
440     #     return str(round(distance_in_meter /
    1000, 2)) + " km"
441     # elif unit == "US" and distance_in_meter < 0.
    3048:
442     #     return str(round(distance_in_meter * 39.
    3701, 2)) + " in"
443     # elif unit == "US" and distance_in_meter <= 0
    .9144:
444     #     return str(round(distance_in_meter * 3.
    28084, 2)) + " ft"
445     # elif unit == "US" and distance_in_meter <=
    1609.34:
446     #     return str(round(distance_in_meter * 1.
    09361, 2)) + " yd"
447     # elif unit == "US" and distance_in_meter >
    1609.34:
448     #     return str(round(distance_in_meter /
    1609.34, 2)) + " mi"
449     # else:
450     #     return str("ERR")
451
452
453 def make_ar_frame(frame):
454     pass
```

```
455     # make sure that input frame is 320x320
456     frame = get_320_320_frame(frame)
457     # frame input resolution: 320 320
458     ar_width = key_engine.get_key("ARDisplayWidth"
    ).get("value")
459     ar_height = key_engine.get_key("
    ARDisplayHeight").get("value")
460     ar_ppi = key_engine.get_key("ARDisplayPPI").
    get("value")
461     user_eye_distance = key_engine.get_key("
    AREyeDistance").get("value")  # meter
462     user_eye_level_adjust = key_engine.get_key("
    AREyeLevelAdjust").get("value")  # meter
463
464     ar_screen = np.zeros((ar_height, ar_width, 3
    ), np.uint8)
465     ar_screen = cv.cvtColor(ar_screen, cv.
    COLOR_BGR2RGB)
466
467     eye_distance_to_inch = user_eye_distance * 39.
    3701
468     eye_distance_to_pixel = int(
    eye_distance_to_inch * ar_ppi)
469
470     eye_level_adjust_to_inch =
    user_eye_level_adjust * 39.3701
471     eye_level_adjust_to_pixel = int(
    eye_level_adjust_to_inch * ar_ppi)
472
473     left_eye_center_x = (ar_width // 2) - (
    eye_distance_to_pixel // 2)
474     right_eye_center_x = (ar_width // 2) + (
    eye_distance_to_pixel // 2)
475     eye_center_y = ar_height // 2 -
    eye_level_adjust_to_pixel
476
477     left_eye_start_x = left_eye_center_x - (320
     // 2)
478     right_eye_start_x = right_eye_center_x - (320
     // 2)
479     eye_start_y = eye_center_y - (320 // 2)
```

```
480
481     preferred_eye = key_engine.get_key("
    ARPreferredEye").get("value")
482
483     if key_engine.get_key("ARMode").get("value"
    ) == "both eye" or key_engine.get_key("
    ARPreferredEye").get(
484             "value") == "left":
485         pass
486         left_eye_screen = frame
487     else:
488         pass
489         left_eye_screen = black_screen
490
491     if key_engine.get_key("ARMode").get("value"
    ) == "both eye" or key_engine.get_key("
    ARPreferredEye").get(
492             "value") == "right":
493         pass
494         right_eye_screen = frame
495     else:
496         pass
497         right_eye_screen = black_screen
498
499     center_cross_bar_width = 0.005  # meter
500     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
501
502     left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
503     right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
504
505     left_eye_screen_width = left_eye_max_x -
    left_eye_start_x
506     right_eye_screen_width = right_eye_start_x +
    320 - right_eye_min_x
507
508     if left_eye_screen_width > 320:
    left_eye_screen_width = 320
509     if right_eye_screen_width > 320:
```

```
509 right_eye_screen_width = 320
510
511     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
512     right_eye_screen = right_eye_screen[:, 320 -
    right_eye_screen_width:]
513
514     eye_screen_height = 320
515     eye_offset_overlap = 0
516     if eye_start_y < 0:
517         pass
518         eye_screen_height += eye_start_y
519         eye_offset_overlap = -eye_start_y
520         eye_start_y = 0
521         left_eye_screen = left_eye_screen[
    eye_offset_overlap:320, :]
522         right_eye_screen = right_eye_screen[
    eye_offset_overlap:320, :]
523
524     if eye_start_y + eye_screen_height > ar_height
    :
525         pass
526         eye_screen_height = ar_height -
    eye_start_y
527         left_eye_screen = left_eye_screen[0:
    eye_screen_height, :]
528         right_eye_screen = right_eye_screen[0:
    eye_screen_height, :]
529
530     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
531     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
532     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
533     right_eye_start_x + eye_screen_height -
    right_eye_screen_width:right_eye_start_x +
    eye_screen_height] = right_eye_screen
534
535     # if key_engine.get_key("ARMode").get("value
    ") == "both eye":
```

```
536     #     # new version
537     #     left_eye_screen = frame
538     #     right_eye_screen = frame
539     #
540     #     center_cross_bar_width = 0.005  # meter
541     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
542     #
543     #     left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
544     #     right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
545     #
546     #     left_eye_screen_width = left_eye_max_x
     - left_eye_start_x
547     #     right_eye_screen_width =
    right_eye_start_x + 320 - right_eye_min_x
548     #
549     #     if left_eye_screen_width > 320:
    left_eye_screen_width = 320
550     #     if right_eye_screen_width > 320:
    right_eye_screen_width = 320
551     #
552     #     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
553     #     right_eye_screen = right_eye_screen[:,
    320 - right_eye_screen_width:]
554     #
555     #     eye_screen_height = 320
556     #     eye_offset_overlap = 0
557     #     if eye_start_y < 0:
558     #         eye_screen_height += eye_start_y
559     #         eye_offset_overlap = -eye_start_y
560     #         eye_start_y = 0
561     #
562     #     left_eye_screen = left_eye_screen[
    eye_offset_overlap:320, :]
563     #     right_eye_screen = right_eye_screen[
    eye_offset_overlap:320, :]
564     #
565     #     ar_screen[eye_start_y:eye_start_y +
```

```
565 eye_screen_height,
566     #     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
567     #     ar_screen[eye_start_y:eye_start_y +
    eye_screen_height,
568     #     right_eye_start_x + eye_screen_height -
    right_eye_screen_width:right_eye_start_x +
    eye_screen_height] = right_eye_screen
569     #
570     # elif key_engine.get_key("ARMode").get("value
    ") == "one eye" and preferred_eye == "left":
571     #     # ar_screen[eye_start_y:eye_start_y +
    320, left_eye_start_x:left_eye_start_x + 320] =
    frame
572     #     left_eye_screen = frame
573     #
574     #     center_cross_bar_width = 0.01  # meter
575     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
576     #
577     #     left_eye_max_x = (ar_width -
    center_cross_bar_width_pixel) // 2
578     #     left_eye_screen_width = left_eye_max_x
     - left_eye_start_x
579     #     left_eye_screen = left_eye_screen[:, 0:
    left_eye_screen_width]
580     #
581     #     ar_screen[eye_start_y:eye_start_y + 320,
582     #     left_eye_start_x:left_eye_start_x +
    left_eye_screen_width] = left_eye_screen
583     #     pass
584     #
585     # elif key_engine.get_key("ARMode").get("value
    ") == "one eye" and preferred_eye == "right":  #
    copy right eye
586     #     # ar_screen[eye_start_y:eye_start_y +
    320, right_eye_start_x:right_eye_start_x + 320] =
    frame
587     #     right_eye_screen = frame
588     #
589     #     center_cross_bar_width = 0.01  # meter
```

```
590     #     center_cross_bar_width_pixel = int(
    center_cross_bar_width * ar_ppi * 39.3701)
591     #
592     #     right_eye_min_x = (ar_width +
    center_cross_bar_width_pixel) // 2
593     #     right_eye_screen_width =
    right_eye_start_x + 320 - right_eye_min_x
594     #     right_eye_screen = right_eye_screen[:,
    320 - right_eye_screen_width:]
595     #
596     #     ar_screen[eye_start_y:eye_start_y + 320,
597     #     right_eye_start_x + 320 -
    right_eye_screen_width:right_eye_start_x + 320] =
    right_eye_screen
598     #     pass
599
600     return ar_screen
601
602
603 def get_icon(icon_name: str, size: int):
604     pass
605     # if icon_name == "hard_warning.png":
606     #     return hard_warning_icon
607     # elif icon_name == "warning.png":
608     #     return warning_icon
609     # else:
610     #     return black_screen
611     if icon_name == "hard_warning.png":
612         pass
613         return cv.resize(hard_warning_icon, (size
    , size))
614     elif icon_name == "warning.png":
615         pass
616         return cv.resize(warning_icon, (size, size
    ))
617     # else:
618     #     pass
619     return cv.resize(black_screen, (size, size))
620     # pass
621
622
```

```
623 def render_tensor(screen, boxes, classes, scores,
    distance):
624     pass
625     if scores < 0.5:
626         pass
627         return screen
628     box = boxes * [320, 320, 320, 320]
629     class_name = label_engine.get_label(int(
    classes + 1)).get("value")
630     class_color = color_engine.get_color(
    class_name)
631     inverted_color = (255 - class_color[0], 255 -
    class_color[1], 255 - class_color[2])
632     text_x, text_y = 0, 0  # for text position
633     if key_engine.get_key("ROSMindDisplayWay").get
    ("value") == "filled box":
634         pass
635         # outer line
636         cv.rectangle(screen, (int(box[1]), int(box
    [0]), int(box[3]), int(box[2])), inverted_color, 2
    )
637         # inner box
638         cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, -1
    )
639         # put text center of the box
640         text_size = cv.getTextSize(class_name, cv.
    FONT_HERSHEY_SIMPLEX, 0.5, 2)[0]
641         text_x = int((box[1] + box[3]) / 2 -
    text_size[0] / 2)
642         text_y = int((box[0] + box[2]) / 2 +
    text_size[1] / 2)
643         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
644                    inverted_color, 2)
645     elif key_engine.get_key("ROSMindDisplayWay").
    get("value") == "outlined box":
646         pass
647         cv.rectangle(screen, (int(box[1]), int(box
    [0])), (int(box[3]), int(box[2])), class_color, 2)
648         text_x = int(box[1])
```

```
649         text_y = int(box[0])
650         cv.putText(screen, class_name, (text_x,
    text_y), cv.FONT_HERSHEY_SIMPLEX, 0.5,
651                    class_color, 2)
652 
653     if key_engine.get_key("DistanceDisplayEnabled"
    ).get("value"):
654         pass
655         cv.putText(screen, str(distance), (text_x
    , text_y + 20), cv.FONT_HERSHEY_SIMPLEX, 0.5,
656                    inverted_color, 1, cv.LINE_AA)
657 
658     return screen
659 
660 
661 def render_tensors(tensor_output):
662     pass
663     global raw_screen
664     boxes, classes, scores, distance =
    tensor_output
665     in_print_screen = raw_screen
666     # min_distance = 1000.0
667     for i in range(len(scores)):
668         pass
669         in_print_screen = render_tensor(
    in_print_screen, boxes[i], classes[i], scores[i],
    distance[i])
670         # if distance[i].endswith("cm") and float(
    distance[i][:-2]) < min_distance: min_distance =
    float(distance[i][:-2])
671     # if min_distance != 1000.0 and "boot.RTaptic
    " in sys.modules:
672     #     pass
673     #
674     return in_print_screen
675 
676 
677 def render_notifications(frame, notifications):
678     pass
679     one_notification_max_height = 40
680     one_notification_width = 260
```

```
681     notification_start_y = 10
682     printed_y_pixel = 0
683 
684     global black_screen
685     global hard_warning_icon
686 
687     for i in range(len(notifications)):
688         pass
689         # draw notification
690         this_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
691         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
692         this_notification_start_x = (320 -
    this_notification_width) // 2
693         this_notification_radius = int(
    this_notification_height // 2)
694         cv.circle(frame, (
    this_notification_start_x +
    this_notification_radius,
695                           notification_start_y +
    this_notification_radius + printed_y_pixel),
696                   this_notification_radius, (255,
    255, 255), -1)
697         cv.circle(frame, (
    this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
698                           notification_start_y +
    this_notification_radius + printed_y_pixel),
699                   this_notification_radius, (255,
    255, 255), -1)
700         cv.rectangle(frame,
701                      (this_notification_start_x +
    this_notification_radius, notification_start_y +
    printed_y_pixel),
702                      (this_notification_start_x +
    this_notification_width - this_notification_radius
    ,
```

```
703                         notification_start_y +
    printed_y_pixel + this_notification_height), (255
    , 255, 255), -1)
704             printed_y_pixel +=
    this_notification_height
705 
706         printed_y_pixel = 0
707         for i in range(len(notifications) - 1):
708             pass
709             upper_notification_height = int(
    one_notification_max_height * notifications[i].
    display_visibility)
710             lower_notification_height = int(
    one_notification_max_height * notifications[i + 1
    ].display_visibility)
711             upper_notification_radius = int(
    upper_notification_height // 2)
712             lower_notification_radius = int(
    lower_notification_height // 2)
713             upper_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
714             lower_notification_width = int(
    one_notification_width * notifications[i + 1].
    display_visibility)
715             upper_notification_start_x = (320 -
    upper_notification_width) // 2
716             lower_notification_start_x = (320 -
    lower_notification_width) // 2
717             this_notification_width = min(
    upper_notification_width, lower_notification_width
    )
718             this_notification_start_x = max(
    upper_notification_start_x,
    lower_notification_start_x)
719 
720             max_circle_radius = max(
    upper_notification_radius,
    lower_notification_radius)
721             cv.rectangle(frame,
722                          (this_notification_start_x,
```

```
722 notification_start_y + printed_y_pixel +
    upper_notification_radius),
723                       (this_notification_start_x +
    max_circle_radius,
724                        notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
725                       (255, 255, 255), -1)
726         cv.rectangle(frame, (
    this_notification_start_x +
    this_notification_width - max_circle_radius,
727                                notification_start_y
     + printed_y_pixel + upper_notification_radius),
728                       (this_notification_start_x +
    this_notification_width,
729                        notification_start_y +
    printed_y_pixel + upper_notification_height +
    lower_notification_radius),
730                       (255, 255, 255), -1)
731         divide_bar_height = 2
732         cv.rectangle(frame, (
    this_notification_start_x,
733                                notification_start_y
     + printed_y_pixel + upper_notification_height -
    divide_bar_height // 2),
734                       (this_notification_start_x +
    this_notification_width,
735                        notification_start_y +
    printed_y_pixel + upper_notification_height +
    divide_bar_height // 2),
736                       (200, 200, 200), -1)
737         printed_y_pixel +=
    upper_notification_height
738
739     printed_y_pixel = 0
740     printed_y_pixel_after = 0
741     # now draw notification
742     for i in range(len(notifications)):
743         pass
744         printed_y_pixel += printed_y_pixel_after
745         this_notification_height = int(
```

```
745 one_notification_max_height * notifications[i].
    display_visibility)
746         printed_y_pixel_after =
    this_notification_height
747         if this_notification_height <= 10:
    continue
748         this_notification_width = int(
    one_notification_width * notifications[i].
    display_visibility)
749         this_notification_start_x = (320 -
    this_notification_width) // 2
750         this_notification_radius = int(
    this_notification_height // 2)
751         this_notification_image_size =
    this_notification_height - 10
752         if this_notification_image_size <= 0:
    continue
753         # icon = cv.resize(black_screen, (
    this_notification_image_size,
    this_notification_image_size))
754         # if notifications[i].icon == "
    hard_warning.png":
755         #     icon = cv.resize(hard_warning_icon
    , (this_notification_image_size,
    this_notification_image_size))
756         # elif notifications[i].icon == "warning.
    png":
757         #     icon = cv.resize(warning_icon, (
    this_notification_image_size,
    this_notification_image_size))
758         icon = get_icon(notifications[i].icon,
    this_notification_image_size)
759         frame[
760         notification_start_y + 5 + printed_y_pixel
    :notification_start_y + 5 + printed_y_pixel +
    this_notification_image_size,
761         this_notification_start_x + 5:
    this_notification_start_x + 5 +
    this_notification_image_size] = icon
762         string_able_pixel_width =
    this_notification_width -
```

```
762 this_notification_image_size - 10
763         string_able_pixel_height =
    this_notification_height - 10
764         string_max_length =
    string_able_pixel_width // 10
765         string = notifications[i].message
766         if len(string) > string_max_length: string
     = string[:string_max_length] + "..."
767         cv.putText(frame, string, (
    this_notification_start_x +
    this_notification_image_size + 10,
768
    notification_start_y + 5 + printed_y_pixel +
    this_notification_height // 2 + 5),
769                     cv.FONT_HERSHEY_SIMPLEX, 0.5,
770                     (64, 64, 64), 1, cv.LINE_AA)
771
772     return frame
773
774
775 # def color_adjust(frame):
776 #     pass
777 #     color_adjust_mode = key_engine.get_key("
    ColorFilterMode").get("value")
778 #     if color_adjust_mode == "sync mobile":
779 #         pass
780 #     elif color_adjust_mode == "custom":
781 #         pass
782 #         red_boost = key_engine.get_key("
    ColorFilterRedBoost").get("value")
783 #         green_boost = key_engine.get_key("
    ColorFilterGreenBoost").get("value")
784 #         blue_boost = key_engine.get_key("
    ColorFilterBlueBoost").get("value")
785 #
786 #         red_ratio = key_engine.get_key("
    ColorFilterRedRatio").get("value")
787 #         green_ratio = key_engine.get_key("
    ColorFilterGreenRatio").get("value")
788 #         blue_ratio = key_engine.get_key("
    ColorFilterBlueRatio").get("value")
```

```
789 #
790 #         # frame[:, :, 2] = frame[:, :, 2] *
    red_ratio + red_boost
791 #         # frame[:, :, 1] = frame[:, :, 1] *
    green_ratio + green_boost
792 #         # frame[:, :, 0] = frame[:, :, 0] *
    blue_ratio + blue_boost
793 #         # for x in range(320):
794 #         #     pass
795 #         #     for y in range(320):
796 #         #         pass
797 #         #         frame[y, x, 2] = frame[y, x, 2
    ] * red_ratio + red_boost
798 #         #         frame[y, x, 1] = frame[y, x, 1
    ] * green_ratio + green_boost
799 #         #         frame[y, x, 0] = frame[y, x, 0
    ] * blue_ratio + blue_boost
800 #     return frame
801
802
803 def render_tensor_and_etc():
804     pass
805     global raw_screen
806     new_frame = raw_screen
807     # render tensor
808     tensor_output = tensor_engine.tensor_output
809     if tensor_output is not None:
810         pass
811         new_frame = render_tensors(tensor_output)
812     # render etc
813     # render notifications
814     notifications_count = len(notification_engine.
    notifications)
815     try:
816         pass
817         if key_engine.get_key("Notification
    Display Overlap").get("value"): new_frame =
    render_notifications(new_frame,
818
                          notification_engine.notifications)
```

```
819     except Exception as e:
820         print("Error while rendering notifications
    .")
821         print(e)
822         pass
823     # render fps
824     if key_engine.get_key("ROSDisplayFPSEnable").
    get("value"):
825         pass
826         main_screen_fps = round(fps_engine.
    get_main_screen_fps(), 2)
827         tensor_fps = round(fps_engine.
    get_tensor_fps(), 2)
828 
829         red = key_engine.get_key("
    ROSDisplayFPSColorRed").get("value")
830         green = key_engine.get_key("
    ROSDisplayFPSColorGreen").get("value")
831         blue = key_engine.get_key("
    ROSDisplayFPSColorBlue").get("value")
832 
833         # cv.putText(new_frame, "MS FPS: " + str(
    main_screen_fps), (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5,
834         #            (255, 255, 255), 1, cv.
    LINE_AA)
835         # cv.putText(new_frame, " T FPS: " + str(
    tensor_fps), (10, 40), cv.FONT_HERSHEY_SIMPLEX, 0.
    5, (255, 255, 255),
836         #            1, cv.LINE_AA)
837         cv.putText(new_frame, "MS FPS: " + str(
    main_screen_fps), (10, 20), cv.
    FONT_HERSHEY_SIMPLEX, 0.5,
838                    (blue, green, red), 1, cv.
    LINE_AA)
839         cv.putText(new_frame, " T FPS: " + str(
    tensor_fps), (10, 40), cv.FONT_HERSHEY_SIMPLEX, 0.
    5, (blue, green, red),
840                    1, cv.LINE_AA)
841 
842     # render USSRegion
```

```
843     if key_engine.get_key("USSRegionDisplayEnabled
    ").get("value") and "boot.RUSS" in sys.modules:
844         pass
845         uss_region_start_x = key_engine.get_key("
    USSRegionStartX").get("value")
846         uss_region_end_x = key_engine.get_key("
    USSRegionEndX").get("value")
847         uss_region_start_y = key_engine.get_key("
    USSRegionStartY").get("value")
848         uss_region_end_y = key_engine.get_key("
    USSRegionEndY").get("value")
849 
850         red = key_engine.get_key("
    USSRegionDisplayColorRed").get("value")
851         green = key_engine.get_key("
    USSRegionDisplayColorGreen").get("value")
852         blue = key_engine.get_key("
    USSRegionDisplayColorBlue").get("value")
853         # cv.rectangle(new_frame, (
    uss_region_start_x, uss_region_start_y), (
    uss_region_end_x, uss_region_end_y),
854         #               (208, 252, 92), 1)
855         cv.rectangle(new_frame, (
    uss_region_start_x, uss_region_start_y), (
    uss_region_end_x, uss_region_end_y),
856                      (blue, green, red), -1)
857 
858     # filter_engine.color_adjust(new_frame)
859     # new_frame = filter_engine.color_adjust(
    new_frame)
860 
861     global screen
862     screen = new_frame
863     return new_frame
864 
865 
866 def tick_screen():
867     pass
868     global kernel_panicked
869     global screen
870     if kernel_panicked:
```

```
871             pass
872             screen = kernel_panic_screen
873         make_window()
874         if key_engine.get_key("ROSARDisplayEnabled").
    get("value"):
875             pass
876             cv.imshow("ROS", make_ar_frame(
    filter_engine.color_adjust(screen)))
877         else:
878             pass
879             cv.imshow("ROS", filter_engine.
    color_adjust(screen))
880 
881         fps_engine.add_candidate_main_fps()
882         if fps_engine.get_main_screen_fps() < 10 and
    notification_engine.get_notification(
883                 message="Low System FPS detected.") is
     None:
884             pass
885             notification_engine.add_notification("
    warning.png", "Low System FPS detected.",
886                                                  "시스템
     FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
887         if fps_engine.get_tensor_fps() < 6 and
    notification_engine.get_notification(
888                 message="Low Tensor FPS detected.") is
     None:
889             pass
890             notification_engine.add_notification("
    warning.png", "Low Tensor FPS detected.", "텐서
    FPS가 낮습니다. 늦은 반응에 대비 해주세요.")
891 
892         return
893 
894 
895 def set_tensor_input():
896         pass
897         global raw_screen
898         raw_screen = get_320_320_frame(raw_screen)
899         raw_data = cv.cvtColor(raw_screen, cv.
    COLOR_BGR2RGB)
```

```
900     raw_data = np.expand_dims(raw_data, axis=0)
901     tensor_engine.raw_data = raw_data
902     return
903 
904 
905 def shutdown():
906     pass
907     print("Shutting down...")
908     global camera
909     notification_engine.add_notification("
    hard_warning.png", "Shutting down...", "종료 중...")
910     # shutdown thread later
911     tensor_engine.stop_process_frame()
912     label_engine.erase_memory()
913     color_engine.erase_memory()
914     if "boot.RBluetooth" in sys.modules:
915         pass
916         bluetooth_engine.close()
917     key_engine.save_keys()
918     cv.destroyAllWindows()
919     if "picamera2" in sys.modules:
920         pass
921         camera.stop()
922     else:
923         pass
924         camera.release()
925     notification_engine.close()
926     tts_engine.shutdown()
927     if "boot.RTaptic" in sys.modules:
928         pass
929         taptic_engine.shutdown()
930         global taptic_command_thread_run
931         taptic_command_thread_run = False
932         global taptic_command_thread
933         if taptic_command_thread is not None:
    taptic_command_thread.join()
934     if "boot.RUSS" in sys.modules:
935         pass
936         ultrasonic_engine.shutdown()
937     if "boot.RGPIO" in sys.modules:
938         pass
```

```
939             gpio_engine.shutdown()
940         # time.sleep(2)
941         current_threads = threading.enumerate()
942         for thread in current_threads:
943             pass
944             if thread.name == "MainThread": continue
945             print("Thread " + thread.name + " is in
    running.")
946         if len(current_threads) > 1:
947             pass
948             print("There are still threads running.")
949             print("Force shutdown.")
950             os._exit(0)
951         print("Shutdown complete.")
952         exit(0)
953 
954 
955 def bluetooth_connected_callback():
956     pass
957     sound_engine.play("boot/res/alert.mp3")
958     print("Bluetooth connected.")
959     notification_engine.add_notification("warning.
    png", "Bluetooth connected.", "블루투스 연결되었습니다.")
960     return
961 
962 
963 def bluetooth_signal_callback(data):
964     pass
965     global find_my_sounding_one_channel
966     global find_my_keep_sounding_channel
967     global find_my_notification
968     global
    color_weakness_compensation_notification
969     if data == b"a":
970         pass
971         sound_engine.stop(
    find_my_sounding_one_channel)
972         find_my_sounding_one_channel =
    sound_engine.play("boot/res/FindMy.mp3")
973         notification_engine.add_notification("
    warning.png", "Find My is activated.", "나의 찾기가
```

```
973 활성화 되었습니다.")
974     elif data == b"b":
975         pass
976         find_my_keep_sounding_channel =
    sound_engine.play("boot/res/FindMy_long.mp3", -1)
977         find_my_notification =
    notification_engine.add_notification("warning.png
    ", "Find My is activated.",
978                                      "나의 찾기가 활성화 되었습니다.")
979         find_my_notification.display_duration =
    1000000
980         # pass
981     elif data == b"c":
982         pass
983         sound_engine.stop(
    find_my_keep_sounding_channel)
984         find_my_notification.display_duration = 0
985         # pass
986     elif data == b"normal":
987         pass
988         filter_engine.mobile_filter = "normal"
989         if
    color_weakness_compensation_notification is not
    None: color_weakness_compensation_notification.
    display_duration = 0
990         color_weakness_compensation_notification
     = notification_engine.add_notification("warning.
    png", "Color weakness compensation disabled.",
991                                                "색약
    보정이 비활성화 되었습니다.")
992     elif data == b"red":
993         pass
994         filter_engine.mobile_filter = "red_weak"
995         if
    color_weakness_compensation_notification is not
    None: color_weakness_compensation_notification.
    display_duration = 0
996         color_weakness_compensation_notification
     = notification_engine.add_notification("warning.
    png", "Color Red weakness compensation enabled.",
```

```
 997                                                                 "
     적색약 보정이 활성화 되었습니다.")
 998     elif data == b"blue":
 999         pass
1000         filter_engine.mobile_filter = "blue_weak"
1001         if
     color_weakness_compensation_notification is not
     None: color_weakness_compensation_notification.
     display_duration = 0
1002         color_weakness_compensation_notification
      = notification_engine.add_notification("warning.
     png", "Color Blue weakness compensation enabled."
     ,
1003                                                                 "
     청색약 보정이 활성화 되었습니다.")
1004     elif data == b"all":
1005         pass
1006         filter_engine.mobile_filter = "all_weak"
1007         if
     color_weakness_compensation_notification is not
     None: color_weakness_compensation_notification.
     display_duration = 0
1008         color_weakness_compensation_notification
      = notification_engine.add_notification("warning.
     png", "All Color weakness compensation enabled.",
1009                                                                 "
     전체 색약 보정이 활성화 되었습니다. 흑백으로 나타납니다.")
1010     return
1011
1012
1013 def boot_logo(started_ticks: float, target_ticks
     : float = 8.0):
1014     pass
1015     global screen
1016     # screen = black_screen
1017     # copy black screen to screen
1018     screen = np.zeros((320, 320, 3), np.uint8)
1019     screen = cv.cvtColor(screen, cv.COLOR_BGR2RGB
     )
1020     global splash_screen
1021     resized_splash_screen = cv.resize(
```

```
1021 splash_screen, (80, 80))
1022     screen[120:200, 120:200] =
     resized_splash_screen
1023
1024     # cv.putText(screen, "ROS", (10, 20), cv.
     FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv
     .LINE_AA)
1025     boot_progress = 0
1026     if started_ticks < target_ticks * 0.25:
1027         pass
1028         return
1029     elif started_ticks < target_ticks * 0.35:
1030         # boot_progress = started_ticks * 10.0
1031         # boot_progress = started_ticks / (
     target_ticks * 0.35) * 30.0
1032         boot_progress = (started_ticks -
     target_ticks * 0.25) / (target_ticks * 0.1) * 30.
     0
1033         pass
1034     elif started_ticks < target_ticks * 0.6:
1035         # boot_progress = 30 + (started_ticks - 3
     ) * 30.0
1036         # rage is 30 to 90
1037         boot_progress = 30 + (started_ticks -
     target_ticks * 0.35) / (target_ticks * 0.25) * 60
     .0
1038         pass
1039     elif started_ticks < target_ticks * 0.9:
1040         # boot_progress = 90 + 10 / 1.5 * (
     started_ticks - 5)
1041         # range is 90 to 100
1042         boot_progress = 90 + (started_ticks -
     target_ticks * 0.6) / (target_ticks * 0.3) * 10.0
1043         pass
1044     else:
1045         boot_progress = 100
1046         pass
1047
1048     progress_bar_width_margin = 40
1049     progress_bar_height_margin = 10
1050     progress_bar_height = 5
```

```
1051      progress_bar_width = 320 -
      progress_bar_width_margin * 2
1052      progress_bar_corner_radius =
      progress_bar_height // 2
1053      # progress bar background
1054      cv.circle(screen, (progress_bar_width_margin
       + progress_bar_corner_radius,
1055                         320 -
      progress_bar_height_margin -
      progress_bar_corner_radius),
      progress_bar_corner_radius,
1056                 (64, 64, 64), -1)
1057      cv.circle(screen, (320 -
      progress_bar_width_margin -
      progress_bar_corner_radius,
1058                         320 -
      progress_bar_height_margin -
      progress_bar_corner_radius),
      progress_bar_corner_radius,
1059                 (64, 64, 64), -1)
1060      cv.rectangle(screen, (
1061          progress_bar_width_margin +
      progress_bar_corner_radius, 320 -
      progress_bar_height_margin - progress_bar_height
      ),
1062                   (320 - progress_bar_width_margin
       - progress_bar_corner_radius, 320 -
      progress_bar_height_margin),
1063                   (64, 64, 64), -1)
1064      # progress bar
1065      cv.circle(screen, (progress_bar_width_margin
       + progress_bar_corner_radius,
1066                         320 -
      progress_bar_height_margin -
      progress_bar_corner_radius),
      progress_bar_corner_radius,
1067                 (255, 255, 255), -1)
1068      cv.rectangle(screen, (
1069          progress_bar_width_margin +
      progress_bar_corner_radius, 320 -
      progress_bar_height_margin - progress_bar_height
```

```
1069 ),
1070                 (
1071                     int(
     progress_bar_width_margin +
     progress_bar_corner_radius + (
1072                                 320 -
     progress_bar_width_margin -
     progress_bar_corner_radius -
     progress_bar_width_margin -
     progress_bar_corner_radius) * (
1073                                     boot_progress /
     100)), 320 - progress_bar_height_margin), (255,
     255, 255), -1)
1074     cv.circle(screen, (int(
     progress_bar_width_margin +
     progress_bar_corner_radius + (
1075             320 - progress_bar_width_margin -
     progress_bar_corner_radius -
     progress_bar_width_margin -
     progress_bar_corner_radius) * (
1076                                     boot_progress
      / 100)),
1077                         320 -
     progress_bar_height_margin -
     progress_bar_corner_radius),
     progress_bar_corner_radius,
1078                (255, 255, 255), -1)
1079     return
1080
1081
1082 def rotate_clockwise_90(frame):
1083     pass
1084     return cv.transpose(cv.flip(frame, 0))
1085
1086
1087 def rotate_counterclockwise_90(frame):
1088     pass
1089     return cv.transpose(cv.flip(frame, 1))
1090
1091
1092 def rotate_180(frame):
```

```
1093     pass
1094     return cv.flip(frame, -1)
1095 
1096 
1097 def kernel_panic_check():
1098     pass
1099     # debug
1100     # return True
1101     if hard_warning_icon is None:
1102         pass
1103         notification_engine.add_notification("
     hard_warning.png", "Kernel Panic, no
     hard_warning_icon",
1104                                              "
     장비를 당장 벗으십시요. 커널이 패닉 상태입니다.")
1105         return True
1106     if warning_icon is None:
1107         pass
1108         notification_engine.add_notification("
     hard_warning.png", "Kernel Panic, no warning_icon
     ",
1109                                              "
     장비를 당장 벗으십시요. 커널이 패닉 상태입니다.")
1110         return True
1111     if black_screen is None:
1112         pass
1113         notification_engine.add_notification("
     hard_warning.png", "Kernel Panic, no black_screen
     ",
1114                                              "
     장비를 당장 벗으십시요. 커널이 패닉 상태입니다.")
1115         return True
1116     if kernel_panic_screen is None:
1117         pass
1118         notification_engine.add_notification("
     hard_warning.png", "Kernel Panic, no
     kernel_panic_screen",
1119                                              "
     장비를 당장 벗으십시요. 커널이 패닉 상태입니다.")
1120         return True
1121     if splash_screen is None:
```

```
1122            pass
1123            notification_engine.add_notification("
     hard_warning.png", "Kernel Panic, no
     splash_screen",
1124                                                 "
     장비를 당장 벗으십시요. 커널이 패닉 상태입니다.")
1125            return True
1126        if raw_screen is None:
1127            pass
1128            notification_engine.add_notification("
     hard_warning.png", "Kernel Panic, no raw_screen",
1129                                                 "
     장비를 당장 벗으십시요. 커널이 패닉 상태입니다.")
1130            return True
1131        if tensor_engine.model is None:
1132            pass
1133            notification_engine.add_notification("
     hard_warning.png", "Kernel Panic, no model",
1134                                                 "
     장비를 당장 벗으십시요. 커널이 패닉 상태입니다.")
1135            return True
1136        return False
1137
1138
1139 def distance_to_meter(distance: str):
1140        pass
1141        if distance is None: return None
1142        if distance == "": return None
1143        if distance.endswith("cm"):
1144            pass
1145            return float(distance[:-2]) / 100
1146        elif distance.endswith("m"):
1147            pass
1148            return float(distance[:-1])
1149        elif distance.endswith("km"):
1150            pass
1151            return float(distance[:-2]) * 1000
1152        elif distance.endswith("in"):
1153            pass
1154            return float(distance[:-2]) * 0.0254
1155        elif distance.endswith("ft"):
```

```
1156         pass
1157         return float(distance[:-2]) * 0.3048
1158     elif distance.endswith("yd"):
1159         pass
1160         return float(distance[:-2]) * 0.9144
1161     elif distance.endswith("mi"):
1162         pass
1163         return float(distance[:-2]) * 1609.34
1164     return None
1165 
1166 
1167 def distance_taptic_feedback():
1168     pass
1169     if "boot.RTaptic" not in sys.modules: return
1170     if tensor_engine.tensor_output is None:
1171         pass
1172         taptic_engine.left_taptic.change_amp(0.0)
1173         taptic_engine.right_taptic.change_amp(0.0
     )
1174         return
1175     boxes, classes, scores, distance =
     tensor_engine.tensor_output
1176     boxes = boxes * [320, 320, 320, 320]
1177     if len(distance) == 0:
1178         pass
1179         taptic_engine.left_taptic.change_amp(0.0)
1180         taptic_engine.right_taptic.change_amp(0.0
     )
1181         return
1182     # distance is in any unit: cm, m, km, in, ft
     , yd, mi
1183     taptic_start_distance = key_engine.get_key("
     TapticFeedbackStartDistance").get("value")
1184 
1185     distance_in_meter = []
1186 
1187     for o in range(len(distance)):
1188         pass
1189         # if distance[o].endswith("cm") and float
     (
1190         #     distance[o][:-2]) <
```

```
1190 taptic_start_distance * 100: distance_in_meter.
     append(float(distance[o][:-2]) / 100)
1191         distance_in_meter.append(
     distance_to_meter(distance[o]))
1192
1193     # min_distance = min(distance_in_meter)
1194     # if min_distance is None: return
1195
1196     min_distance = None
1197     for o in range(len(distance_in_meter)):
1198         pass
1199         if distance_in_meter[o] is None: continue
1200         min_distance = distance_in_meter[o]
1201         break
1202     # min_distance_index = 0
1203     for o in range(len(distance_in_meter)):
1204         pass
1205         if distance_in_meter[o] is None: continue
1206         if min_distance > distance_in_meter[o]:
     min_distance = distance_in_meter[o]
1207
1208     if min_distance is None or min_distance >
     taptic_start_distance:
1209         pass
1210         taptic_engine.left_taptic.change_amp(0.0)
1211         taptic_engine.right_taptic.change_amp(0.0
     )
1212         return
1213
1214     min_distance_index = distance_in_meter.index(
     min_distance)
1215     one_section = 320 / 3
1216     object_x_center = (boxes[min_distance_index][
     1] + boxes[min_distance_index][3]) / 2
1217     target_amp = 1.0 - min_distance /
     taptic_start_distance
1218     if target_amp < 0.0: target_amp = 0.0
1219     # change_amp
1220     if object_x_center < one_section:  # left
1221         pass
1222         # debug
```

```
1223            print("left taptic feedback" + str(
    target_amp))
1224            # debug end
1225            taptic_engine.left_taptic.change_amp(
    target_amp)
1226            taptic_engine.right_taptic.change_amp(0.0
    )
1227        elif object_x_center < one_section * 2:  #
    center
1228            pass
1229            # debug
1230            print("center taptic feedback" + str(
    target_amp))
1231            # debug end
1232            taptic_engine.left_taptic.change_amp(
    target_amp)
1233            taptic_engine.right_taptic.change_amp(
    target_amp)
1234        else:  # right
1235            pass
1236            # debug
1237            print("right taptic feedback" + str(
    target_amp))
1238            # debug end
1239            taptic_engine.left_taptic.change_amp(0.0)
1240            taptic_engine.right_taptic.change_amp(
    target_amp)
1241        return
1242
1243
1244 def taptic_feedback_loop():
1245        global taptic_command_thread_run
1246        while taptic_command_thread_run:
1247            pass
1248            distance_taptic_feedback()
1249            time.sleep(0.1)
1250        return
1251
1252
1253 print("defs defined.")
1254
```

```
1255 print("preparing RKernel...")
1256 tensor_engine.fps_engine = fps_engine
1257 tensor_engine.calculate_distance_function =
     calculate_distance
1258 notification_engine.tts_engine = tts_engine
1259 notification_engine.tts_enabled = key_engine.
     get_key("Notification TTS Enabled").get("value")
1260 if "boot.RBluetooth" in sys.modules:
1261     pass
1262     print("callback set.")
1263     bluetooth_engine.connected_callback =
     bluetooth_connected_callback
1264     bluetooth_engine.recv_callback =
     bluetooth_signal_callback
1265 camera = get_camera()
1266 sound_engine.overall_volume = key_engine.get_key(
     "SoundVolume").get("value")
1267 filter_engine.key_engine = key_engine
1268 if "boot.RGPIOD" in sys.modules:
1269     pass
1270     if "boot.RTaptic" in sys.modules:
     taptic_engine.gpio_engine = gpio_engine
1271     if "boot.RUSS" in sys.modules:
     ultrasonic_engine.gpio_engine = gpio_engine
1272
1273 if "boot.RTaptic" in sys.modules:
1274     pass
1275     taptic_engine.init()
1276
1277 if "boot.RUSS" in sys.modules:
1278     pass
1279     ultrasonic_engine.init()
1280
1281 if "boot.RTaptic" in sys.modules:
1282     pass
1283     taptic_command_thread = threading.Thread(
     target=taptic_feedback_loop).start()
1284
1285 print("RKernel prepared.")
1286
1287 # set_tensor_input()
```

```
1288 if key_engine.get_key("ROSBootChimeEnabled").get(
     "value"):
1289     pass
1290     sound_engine.play("boot/res/StartUp.mp3")
1291 
1292 started_time = time.time()
1293 splash_display_time = key_engine.get_key("
     ROSSplashScreenTime").get("value")
1294 while time.time() - started_time <
     splash_display_time:
1295     pass
1296     boot_logo(time.time() - started_time,
     splash_display_time)
1297     tick_screen()
1298     # debug
1299     # if hard_warning_icon is None:
1300     #     kernel_panicked = True
1301     kernel_panicked = kernel_panic_check()
1302     if cv.waitKey(1) & 0xFF == key_engine.get_key
     ("ROSOffKey").get("value"):
1303         pass
1304         shutdown()
1305 
1306 tts_engine.sound_engine = sound_engine
1307 notification_engine.sound_engine = sound_engine
1308 
```

```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.8 </> <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 <name> ColorFilterMode </> <type> str </> <value>
   sync mobile </> <comment>
   ColorFilterModeNoticeColor filter mode </>
54 <name> ColorFilterRedBoost </> <type> int </> <
   value> 0 </> <comment>
   ColorFilterRedBoostNoticeColor filter red boost </>
55 <name> ColorFilterGreenBoost </> <type> int </> <
   value> 0 </> <comment>
   ColorFilterGreenBoostNoticeColor filter green boost
    </>
56 <name> ColorFilterBlueBoost </> <type> int </> <
   value> 0 </> <comment>
   ColorFilterBlueBoostNoticeColor filter blue boost
    </>
57 <name> ColorFilterRedRatio </> <type> float </> <
   value> 1.0 </> <comment>
   ColorFilterRedRatioNoticeColor filter red ratio </>
58 <name> ColorFilterGreenRatio </> <type> float </> <
   value> 1.0 </> <comment>
   ColorFilterGreenRatioNoticeColor filter green ratio
    </>
59 <name> ColorFilterBlueRatio </> <type> float </> <
   value> 1.0 </> <comment>
   ColorFilterBlueRatioNoticeColor filter blue ratio
    </>
```

```
60 <name> BlueLightFilterEnabled </> <type> bool
    </> <value> True </> <comment>
   BlueLightFilterEnabledNoticeBlue light filter is
   enabled or not </>
61 <name> BlueLightFilterStrength </> <type> float
    </> <value> 0.2 </> <comment>
   BlueLightFilterStrengthNoticeBlue light filter
   strength </>
62
```

```
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
```