Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет»
Электротехнический факультет
Кафедра: Информационные технологии и автоматизированные системы

Дисциплина: «Методы и средства визуализации аналитических данных»
Лабораторная работа
на тему: «Создание приборной панели автомобиля с использованием Matplotlib и Blender 3D»

Выполнил: студент группы АСУ8-23-1м
Шеретов Марк Алексеевич
Проверил: д.э.н., профессор кафедры ИТАС
Долгова Елена Владимировна

Пермь 2024

Постановка задачи

Создать на Matplotlib динамическую приборную панель автомобиля с графиками и диаграммами. Отренедерить полученную панель в картинку. Создать 3D модель приборной панели в Blender 3D с использованием отрендеренной картинки.

Ход работы

Создание динамической приборной панели с использованием Matplotlib

Будем отражать на панели стандартные элементы, такие как спидометр, тахометр, одометр, уровень бензина, номер передачи и температуру двигателя. Обычно спидометр градуируется от 0 до 180 км/ч с интервалом разбиения равным 20 км/ч. Тахометр от 0 до 8000 оборотов с интервалом в 1000 оборотов. Для отображения данных элементов будем использовать круговые диаграммы (polar) с указателями (стрелками). В листинге 1 представлен код программы, визуализирующий приборную панель автомобиля.

Листинг 1 – Код программы

```
>>4  import sys
>>5  from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton,
QVBoxLayout, QWidget, QLabel, QLCDNumber, QHBoxLayout
>>6  from PyQt5.QtCore import Qt, QTimer
>>7  from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
>>8  from matplotlib.figure import Figure
>>9  import numpy as np
>10
>11  class Dashboard(QMainWindow):
>12      def __init__(self):
>13          super().__init__()
>14          self.setWindowTitle("Car Dashboard")
>15          self.setGeometry(100, 100, 800, 800)
>16
>17          # Initialize dashboard data
>18          self.speed = 0
>19          self.target_speed = 0
>20          self.rpm = 0
>21          self.target_rpm = 0
>22          self.fuel_level = 100
>23          self.temperature = 90
>24          self.odometer = 0
>25          self.gear = "N"
>26          self.gear_ratios = {"R": 250, "N": 0, "1": 250, "2": 200, "3":
120, "4": 60, "5": 20}
>27          self.speed_history = []
>28          self.rpm_history = []
```

```
>29          self.history_update_count = 0
>30
>31          self.central_widget = QWidget()
>32          self.setCentralWidget(self.central_widget)
>33          self.layout = QVBoxLayout()
>34          self.central_widget.setLayout(self.layout)
>35
>36          self.upper_layout = QHBoxLayout()
>37          self.layout.addLayout(self.upper_layout)
>38
>39          self.speedometer_figure = Figure(facecolor='black')
>40          self.speedometer_canvas = FigureCanvas(self.speedometer_figure)
>41          self.speedometer_ax = self.speedometer_figure.add_subplot(111,
polar=False)
>42          self.upper_layout.addWidget(self.speedometer_canvas)
>43
>44          self.tachometer_figure = Figure(facecolor='black')
>45          self.tachometer_canvas = FigureCanvas(self.tachometer_figure)
>46          self.tachometer_ax = self.tachometer_figure.add_subplot(111,
polar=False)
>47          self.upper_layout.addWidget(self.tachometer_canvas)
>48
>49          self.digital_layout = QHBoxLayout()
>50          self.layout.addLayout(self.digital_layout)
>51
>52          self.speed_label = QLabel("Speed (km/h):")
>53          self.speed_label.setStyleSheet("color: green; background-color:
black;")
>54          self.digital_layout.addWidget(self.speed_label)
>55          self.speed_display = QLCDNumber()
>56          self.speed_display.setStyleSheet("color: green; background-
color: black;")
>57          self.digital_layout.addWidget(self.speed_display)
>58
>59          self.rpm_label = QLabel("RPM (x1000):")
>60          self.rpm_label.setStyleSheet("color: green; background-color:
black;")
>61          self.digital_layout.addWidget(self.rpm_label)
>62          self.rpm_display = QLCDNumber()
>63          self.rpm_display.setStyleSheet("color: green; background-color:
black;")
>64          self.digital_layout.addWidget(self.rpm_display)
>65
>66          self.fuel_label = QLabel("Fuel Level (%):")
>67          self.fuel_label.setStyleSheet("color: green; background-color:
black;")
>68          self.digital_layout.addWidget(self.fuel_label)
```

```
>69          self.fuel_display = QLCDNumber()
>70          self.fuel_display.setStyleSheet("color: green; background-
color: black;")
>71          self.digital_layout.addWidget(self.fuel_display)
>72
>73          self.temp_label = QLabel("Temperature (°C):")
>74          self.temp_label.setStyleSheet("color: green; background-color:
black;")
>75          self.digital_layout.addWidget(self.temp_label)
>76          self.temp_display = QLCDNumber()
>77          self.temp_display.setStyleSheet("color: green; background-
color: black;")
>78          self.digital_layout.addWidget(self.temp_display)
>79
>80          self.odometer_label = QLabel("Odometer (km):")
>81          self.odometer_label.setStyleSheet("color: green; background-
color: black;")
>82          self.digital_layout.addWidget(self.odometer_label)
>83          self.odometer_display = QLCDNumber()
>84          self.odometer_display.setStyleSheet("color: green; background-
color: black;")
>85          self.digital_layout.addWidget(self.odometer_display)
>86
>87          self.gear_label = QLabel("Gear:")
>88          self.gear_label.setStyleSheet("color: green; background-color:
black;")
>89          self.digital_layout.addWidget(self.gear_label)
>90          self.gear_display = QLabel(self.gear)
>91          self.gear_display.setStyleSheet("color: green; background-
color: black;")
>92          self.gear_display.setAlignment(Qt.AlignCenter)
>93          self.digital_layout.addWidget(self.gear_display)
>94
>95          self.history_figure = Figure(facecolor='black')
>96          self.history_canvas = FigureCanvas(self.history_figure)
>97          self.history_ax_speed = self.history_figure.add_subplot(211)
>98          self.history_ax_rpm = self.history_figure.add_subplot(212)
>99          self.layout.addWidget(self.history_canvas)
100
101         self.buttons_layout = QHBoxLayout()
102         self.layout.addLayout(self.buttons_layout)
103
104         self.gas_button = QPushButton("Gas")
105         self.gas_button.clicked.connect(self.increase_speed)
106         self.buttons_layout.addWidget(self.gas_button)
107
108         self.brake_button = QPushButton("Brake")
```

```
109             self.brake_button.clicked.connect(self.decrease_speed)
110             self.buttons_layout.addWidget(self.brake_button)
111
112             self.gear_up_button = QPushButton("Gear Up")
113             self.gear_up_button.clicked.connect(self.gear_up)
114             self.buttons_layout.addWidget(self.gear_up_button)
115
116             self.gear_down_button = QPushButton("Gear Down")
117             self.gear_down_button.clicked.connect(self.gear_down)
118             self.buttons_layout.addWidget(self.gear_down_button)
119
120             self.timer = QTimer()
121             self.timer.timeout.connect(self.update_dashboard)
122             self.timer.start(10)  # Approximately 60 FPS
123
124         def increase_speed(self):
125             self.target_rpm = min(self.target_rpm + 500, 8000)
126             if self.gear != "N":
127                 self.target_speed = min((self.target_rpm /
self.gear_ratios[self.gear]) if self.gear_ratios[self.gear] > 0 else 0, 180)
128
129         def decrease_speed(self):
130             if self.target_speed > 0:
131                 self.target_speed = max(self.target_speed - 10, 0)
132             if self.target_rpm > 0:
133                 self.target_rpm = max(self.target_rpm - 500, 0)
134
135         def gear_up(self):
136             gears = ["R", "N", "1", "2", "3", "4", "5"]
137             current_index = gears.index(self.gear)
138             if current_index < len(gears) - 1:
139                 self.gear = gears[current_index + 1]
140                 self.update_target_rpm_and_speed()
141
142         def gear_down(self):
143             gears = ["R", "N", "1", "2", "3", "4", "5"]
144             current_index = gears.index(self.gear)
145             if current_index > 0:
146                 self.gear = gears[current_index - 1]
147                 self.update_target_rpm_and_speed()
148
149         def update_target_rpm_and_speed(self):
150             if self.gear != "N":
151                 ratio = self.gear_ratios[self.gear]
152                 self.target_rpm = min(self.speed * ratio, 8000)
153                 self.target_speed = min(self.rpm / ratio, 180)
154             else:
```

```
155             self.target_rpm = 0
156
157     def update_dashboard(self):
158         self.speed += (self.target_speed - self.speed) * 0.1
159         self.rpm += (self.target_rpm - self.rpm) * 0.1
160
161         self.fuel_level -= self.rpm / 100000
162
163         if self.speed >= 180:
164             self.rpm = min(self.rpm, 8000)
165         if self.rpm >= 8000:
166             self.speed = min(self.speed, 180)
167
168         self.speed_display.display(round(self.speed))
169         self.rpm_display.display(round(self.rpm / 1000, 1))
170         self.fuel_display.display(self.fuel_level)
171         self.temp_display.display(self.temperature)
172         self.odometer += self.speed / 3600
173         self.odometer_display.display(round(self.odometer, 2))
174         self.gear_display.setText(self.gear)
175         self.update_dials()
176
177         if self.history_update_count < 20:
178             self.history_update_count += 1
179         else:
180             self.update_history()
181             self.history_update_count = 0
182
183
184
185     def update_dials(self):
186         self.speedometer_ax.clear()
187         self.speedometer_ax.set_xlim(-1.5, 1.5)
188         self.speedometer_ax.set_ylim(-1.5, 1.5)
189         self.speedometer_ax.set_xticks([])
190         self.speedometer_ax.set_yticks([])
191
192         theta = np.linspace(-np.pi / 4, 5 * np.pi / 4, 100)
193         self.speedometer_ax.plot(np.cos(theta), np.sin(theta),
color="gray")
194         for i in range(0, 181, 20):
195             angle = -np.pi / 4 + i * (3 * np.pi / 2) / 180
196             x, y = 1.2 * np.cos(angle), 1.2 * np.sin(angle)
197             self.speedometer_ax.text(x, y, str(i), fontsize=8,
color="black", ha="center", va="center")
198
199         speed_angle = -np.pi / 4 + self.speed * (3 * np.pi / 2) / 180
```

```
200         self.speedometer_ax.arrow(0, 0, 0.9 * np.cos(speed_angle), 0.9
* np.sin(speed_angle),
201                                   head_width=0.1, head_length=0.1,
fc="cyan", ec="cyan")
202
203         self.tachometer_ax.clear()
204         self.tachometer_ax.set_xlim(-1.5, 1.5)
205         self.tachometer_ax.set_ylim(-1.5, 1.5)
206         self.tachometer_ax.set_xticks([])
207         self.tachometer_ax.set_yticks([])
208
209         theta = np.linspace(-np.pi / 4, 5 * np.pi / 4, 100)
210         self.tachometer_ax.plot(np.cos(theta), np.sin(theta),
color="gray")
211         for i in range(0, 9):
212             angle = -np.pi / 4 + i * (3 * np.pi / 2) / 8
213             x, y = 1.2 * np.cos(angle), 1.2 * np.sin(angle)
214             self.tachometer_ax.text(x, y, str(i), fontsize=8,
color="black", ha="center", va="center")
215
216         rpm_angle = -np.pi / 4 + (self.rpm / 1000) * (3 * np.pi / 2) /
8
217         self.tachometer_ax.arrow(0, 0, 0.9 * np.cos(rpm_angle), 0.9 *
np.sin(rpm_angle),
218                                  head_width=0.1, head_length=0.1,
fc="red", ec="red")
219
220         self.speedometer_canvas.draw()
221         self.tachometer_canvas.draw()
222
223     def update_history(self):
224         self.speed_history.append(self.speed)
225         self.rpm_history.append(self.rpm)
226         if len(self.speed_history) > 100:
227             self.speed_history.pop(0)
228             self.rpm_history.pop(0)
229
230         self.history_ax_speed.clear()
231         self.history_ax_speed.plot(self.speed_history, color="cyan")
232         self.history_ax_speed.set_title("Speed History", color="white")
233         self.history_ax_speed.tick_params(colors="white")
234         self.history_ax_speed.set_facecolor("black")
235
236         self.history_ax_rpm.clear()
237         self.history_ax_rpm.plot(self.rpm_history, color="red")
238         self.history_ax_rpm.set_title("RPM History", color="white")
239         self.history_ax_rpm.tick_params(colors="white")
```

```
240             self.history_ax_rpm.set_facecolor("black")
241
242             self.history_canvas.draw()
243
244     if __name__ == "__main__":
245         app = QApplication(sys.argv)
246         dashboard = Dashboard()
247         dashboard.show()
248         sys.exit(app.exec_())
```

Особенности спидометра (speedometer_ax) (рис. 1):

- Была использована полярная проекция: add_subplot(111, polar=False)
- Начальный угол необходимо было сместить `theta = np.linspace(-np.pi / 4, 5 * np.pi / 4, 100)`, чтобы стрелка спидометра указывала на 0, как в реальных панелях (в левом нижнем углу)
- Радиальные метки были скрыты: `self.speedometer_ax.set_xticks([])`
- `self.speedometer_ax.set_yticks([])`
- Для красоты был добавлен декоративный фон: `set_facecolor('gray')`
- Добавлена стрелка скорости для реалистичности: speedometer_ax.arrow(0, 0, 0.9 * np.cos(speed_angle), 0.9 * np.sin(speed_angle), head_width=0.1, head_length=0.1, fc="cyan", ec="cyan")

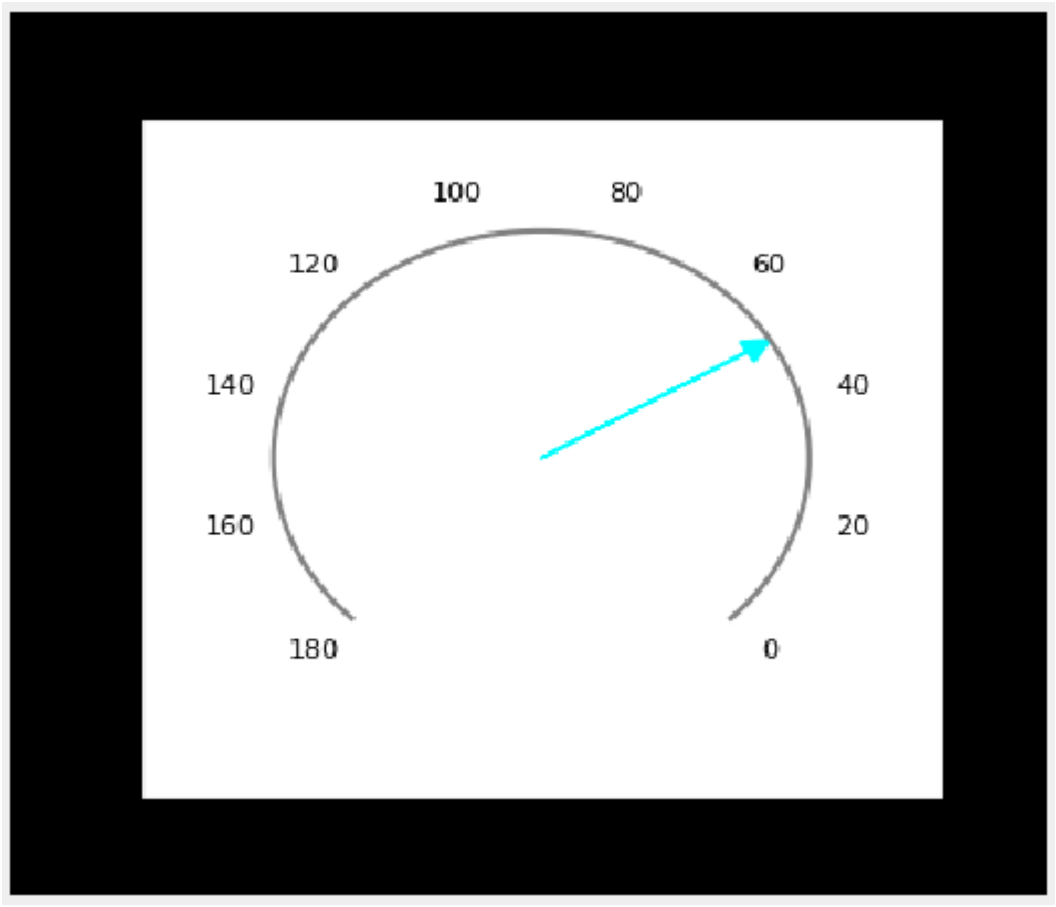

Рисунок 1 – Элемент «Спидометр»

Тахометр выполнен по аналогии (tachometer_ax) (рис. 2):

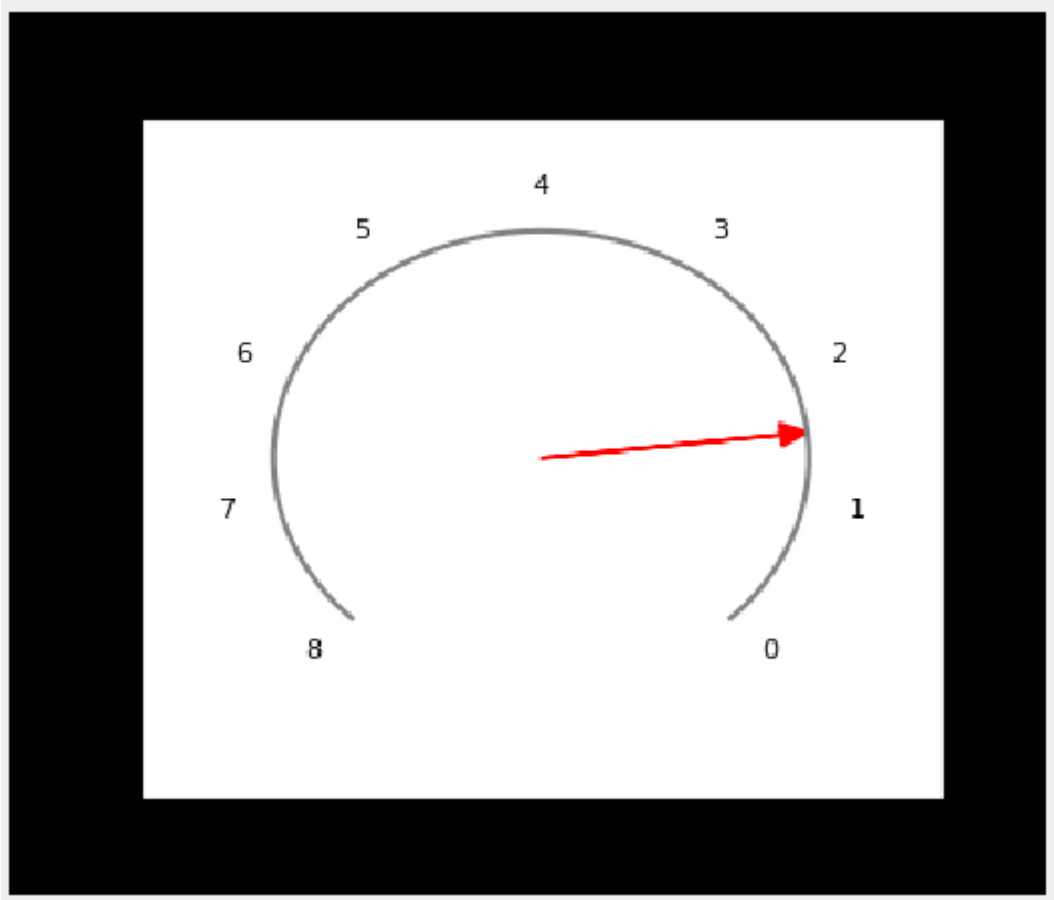

Рисунок 2 – Элемент «Тахометр»

Также информация о скорости, оборотах двигателя в минуту, уровне топлива, температуре двигателя, пробег, вкл. передача, выводятся на дополнительной цифровой панели, выполненной при помощи совокупности QLabel, и QLCDNumber-ов. Вид цифровой панели показан на рисунке 3.

Рисунок 3 – Элемент цифровых показателей

Также, ниже цифровых показателей выводится график истории изменения скорости и оборотов двигателя, выполненные в виде линейного графика зависимости скорости от времени, и оборотов в минуту от времени. Панель истории показана на рисунке 4.

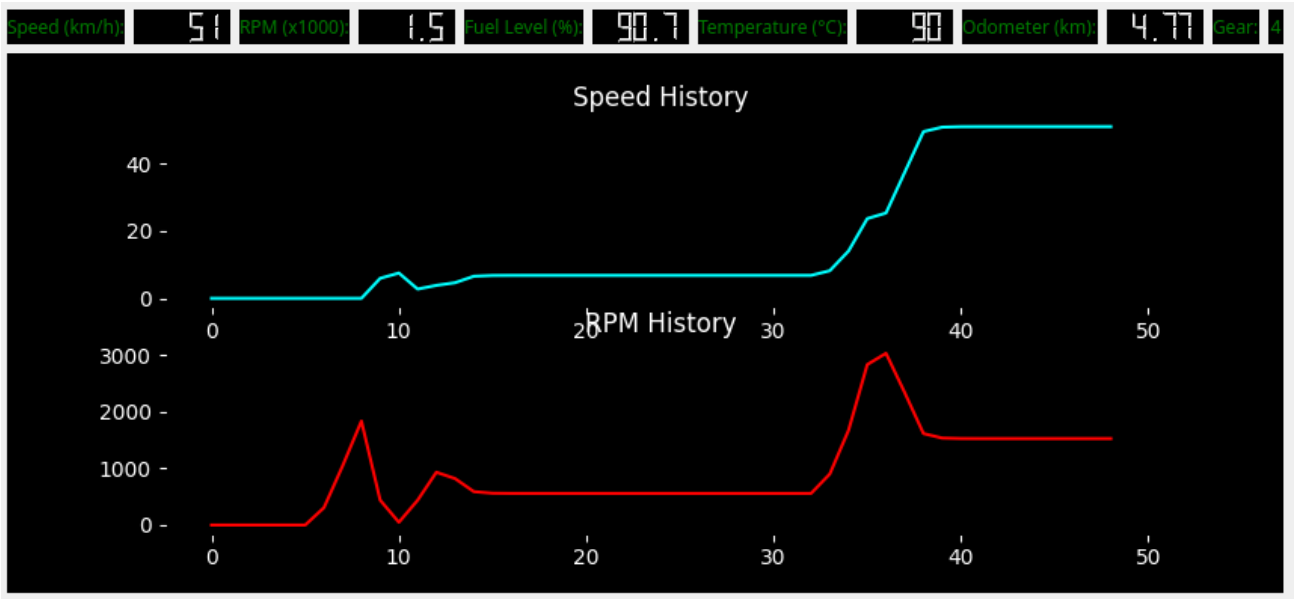

Рисунок 4 — график истории изменения скорости

Итоговая приборная панель автомобиля представлена на рисунке 5.

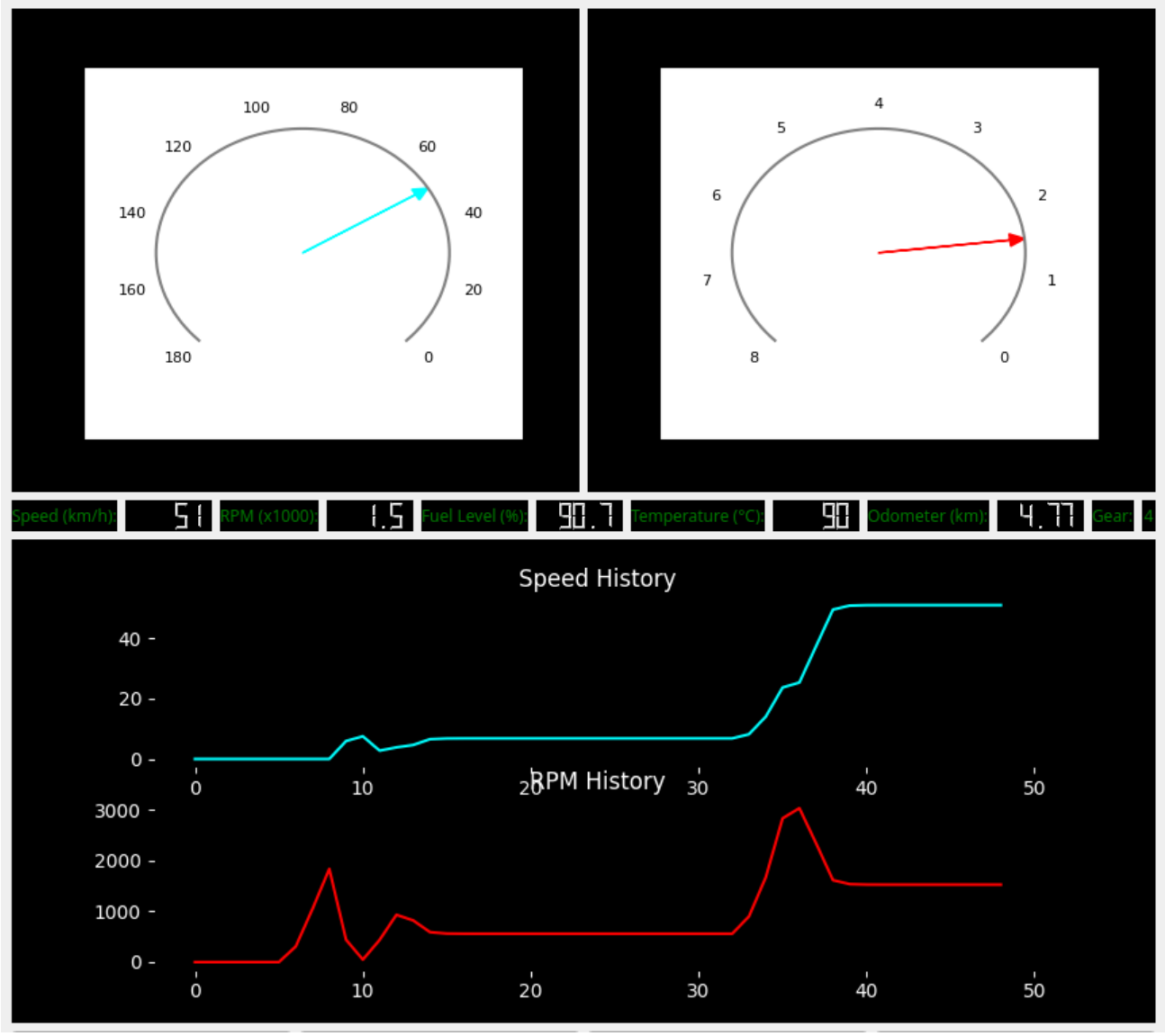

Рисунок 5 – Итоговый вид приборной панели автомобиля

Для динамического изменения параметров приборной панели были добавлены 4 кнопки с помощью PyQt5 (рис. 6):

- «Gas» - активирует ускорение;
- «Brake» - снижает скорость и обороты двигателя;
- «Gear Up» - Повысить передачу;
- «Gear Down» - Понизить передачу.

Рисунок 6 – Интерфейс управления приборной панелью

Функция update_dashboard обновляет значения приборов каждые 10 мс:

- Изменяет скорость, обороты двигателя, температуру и уровень топлива в зависимости от состояния (ускорение, торможение)
- Обновляет стрелки на спидометре и тахометре
- Обновляет график истории.

Создание 3D модели в Blender 3D

В Blender 3D был добавлен объект «Куб», отсечен пополам, выдавлен. Далее у нее была добавлена внутренняя часть и выдавлена внутрь.

В шейдинге была добавлена нода «Текстура изображения». В нее импортирована картинка приборной панели, нода соединена с выводом на материал, также для эффекта остекленения добавлен шейдер Principled BSDF, Для придания эффекта объёмности текстуре, на основе чёрно-белой версии изображение создана карта нормалей. Вид редактора нод шейдеров показан на рисунке 7.

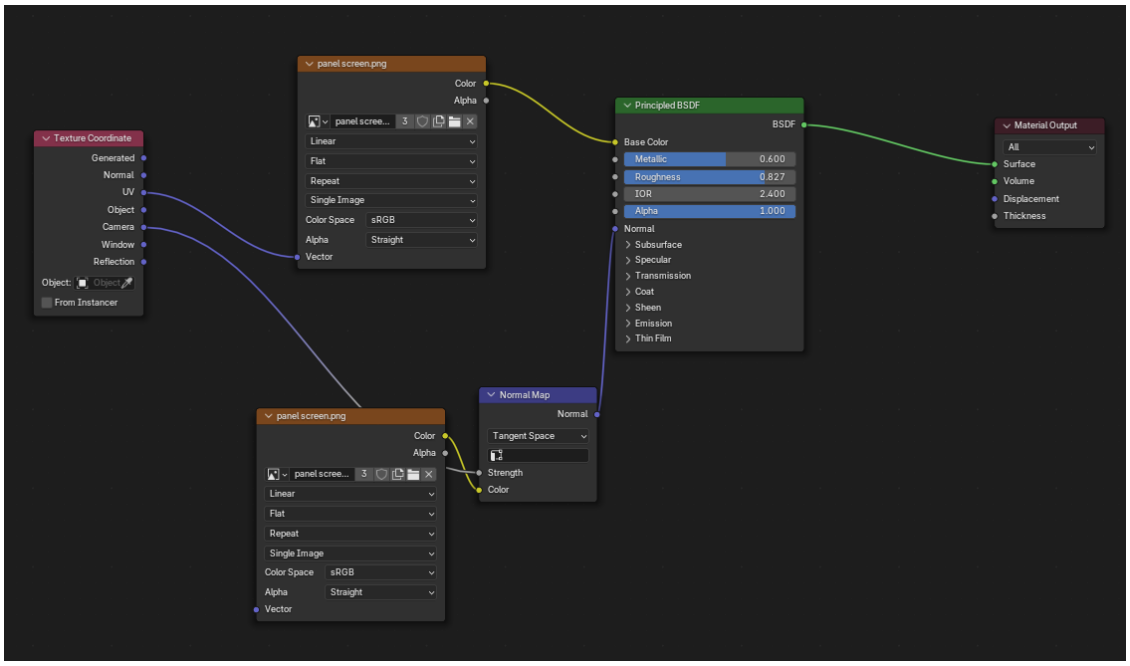

Рисунок 7 – Добавление картинги приборной панели

Получившаяся модель представлена на рисунке 9.

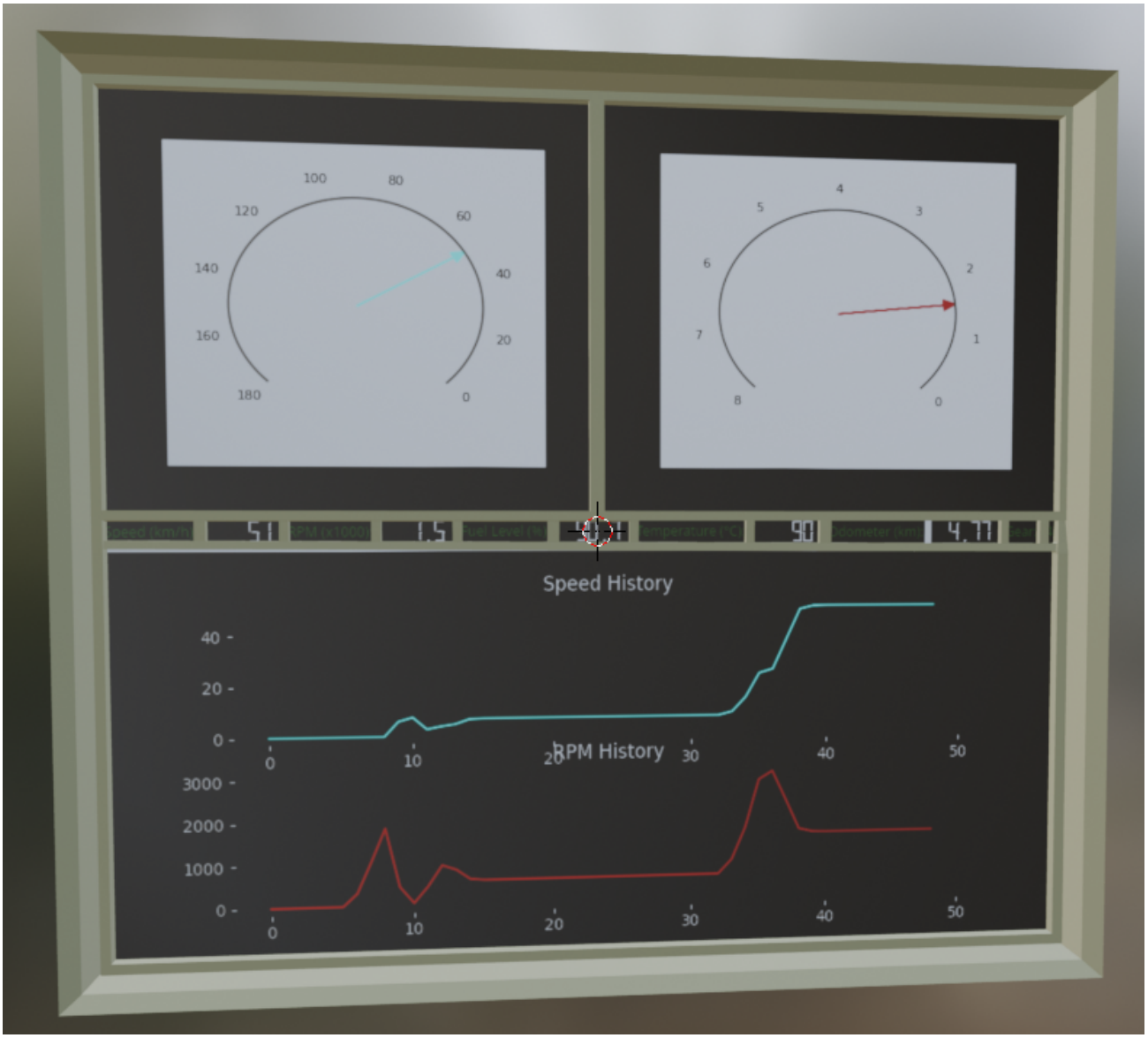

Рисунок 9 – 3D модель приборной панели автомобиля