

Project 42

Software Specific Detailed Design Document

Autonomous Car using Raspberry Pi 3

Team:	U1610041	Azizbek Kobilov
	U1610016	Akmal Karimov
	U1610047	Bekhzodkhon Makhkamov
	U1610052	Boburjon Bahodirov

Contents

1. Introduction.....	2
1.1 Purpose	2
1.2 Scope	2
1.3 References	2
1.4 Overview.....	2
2. General description	3
2.1 Product perspective.....	3
2.2 Product functions	4
2.3 Operating principle	4
2.4 Constraints	5
3. Specific requirements.....	6
3.1 Hardware Requirements	6
3.2 Smart car components:	6
3.3 Software requirements	9
3.3 Code design (example) - Python is used for code	9
Prioritization and Release Plan	14
Appendix I: Selection for Cost-Value Approach.....	14
Appendix II: Prioritization Result of 10 selected Requirements Using Cost-Value Approach	14
Appendix IV: Release Plan	16
Appendix V: I-star.....	16

1. Introduction

This section gives a scope description and overview of everything included in this document. Also, the purpose for this document is described and a list of abbreviations and definitions is provided.

1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for the project. It will illustrate the purpose and complete declaration for the development of system. It will also explain system constraints, interface and interactions with other external applications. This document is primarily intended to be proposed to a Professor for its approval and a reference for developing the first version of the system for the development team.

1.2 Scope

Autonomous car project is a smart car which uses different sensors in order to collect, process data and decide its path of movement. Avoids obstacles using IR sensors, ultrasonic sensor and camera. “Sees” road using camera and detects special lines on the road with IR sensors. Road signs and other objects can be detected and processed using camera.

This project gives us a brief introduction into an autonomous world. Lets us to meet the challenges of real autonomous cars. Project itself is for understanding and solving real world problems in a simulation. All this knowledge and experience can be used in real engineering later.

1.3 References

<https://www.raspberrypi.org/documentation/usage/gpio/python/README.md>

<https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up>

<https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/>

<https://gpiozero.readthedocs.io/en/stable/>

<https://learn.sparkfun.com/tutorials/raspberry-gpio/python-rpigpio-api>

Also, slides provided in lectures.

1.4 Overview

The remainder of this document includes three chapters and appendixes. The second one provides an overview of the system functionality and system interaction with other systems. This chapter also introduces different types of stakeholders and their interaction with the system. Further, the chapter also mentions the system constraints and assumptions about the product.

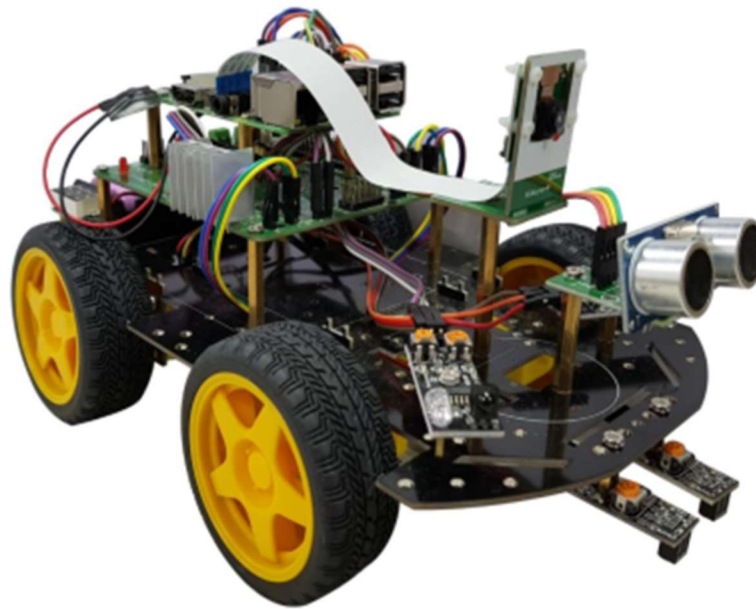
The third chapter provides the requirements specification in detailed terms and a description of the different system interfaces. Different specification techniques are used in order to specify the requirements more precisely for different audiences.

The fourth chapter deals with the prioritization of the requirements. It includes a motivation for the chosen prioritization methods and discusses why other alternatives were not chosen.

The Appendixes in the end of the document include the all results of the requirement prioritization and a release plan based on them.

2. General description

This section will give an overview of the entire system. System will be explained in context how the system interacts with real world, and present the basic functionality. Finally, limitations for the system will be presented.



2.1 Product perspective

There are four main so called components of our car

1. Orientation to the front/back/left/right is done by:
 - DC motor 'n L298N motor controller
2. Distance detection 'n obstacle detection done by:
 - Ultrasonic sensor
 - IR sensors
3. Line tracking
 - Raspberry Pi Camera
4. Wireless control using VNC application 'n Wi-Fi

2.2 Product functions

Ultrasonic sensor – is an instrument that measures the distance to an object using ultrasonic sound waves. An ultrasonic sensor uses a transducer to send and receive ultrasonic pulses that relay back information about an object proximity. High-frequency sound waves reflect from boundaries to produce distinct echo patterns. The ultrasonic sensor module has four pins (ports): VCC, Trig, Echo, GND.

2.3 Operating principle

Raspberry Pi inputs a 10 microsecond HIGH signal from the Trigger pin.

Sensor fires a frequency of 40kHz of 8 cycles.

Input HIGH signal with Echo pin and input LOW signal when reflected on the obstacle.

Echo pin.

If there is no return signal for 36 milliseconds, force the LOW signal.

An infrared (IR) sensor is an electronic device that measures and detects infrared radiation in its surrounding environment.

Anything that emits heat (everything that has a temperature above around -268.15° Celsius) gives off infrared radiation.

It is a device that converts the infrared light into electric that can be processed by signal processing.

IR sensor widely used for crime prevention and fire detection, medical thermography, ecological observation, robot obstacle detection, etc.

There are three ways to implement the line tracer function:

- Infrared tracer: Black and white classification according to reflection by emitting infrared ray
- Camera tracer: Read and analyze information by camera image
- Laser tracer: Black and white classification according to reflection by emitting laser

The path we have chosen is infrared tracer method

IR Line tracer sensor is for line tracking purpose.

- It differentiate white and black color
- Utilizing it enable robot to have intelligence such as line following, or anti collision, or anti-edge falling.

There is a variable resistor on-board for user to tune the threshold of white and black color. (The sensitivity can be changed)

The sensor has 3 pins:

-VCC

-GND

-OUT

2.4 Constraints

As every coin has its second side, so our project also has its own constraints due to number of sensors.

First, we can take an example of Ultrasonic sensor, as it is located almost in the nose of the car we had a lot of problems, we broke sensor two times because pins of the sensor are very small.

IR sensors and Line tracer, we had an issue in the map mission, because the border lines of the map are reflecting the light coming from light bulbs.

One issue should be mentioned, for the first time we were working on one computer, later another member of our team took the car to home, and while making a connection he changed the IP address, next day while connecting car to the first computer we had a problem.

Pins on Raspberry Pi are very weak so they can be broken easily.

Also, battery issue can be considered, Ultrasonic sensor could not be used after battery percentage were going lower 7V, when battery was going even lower the DC motors were not working properly.

3. Specific requirements

This section contains all of the functional and quality requirements of the system. It gives a detailed description of the system and all its features.

3.1 Hardware Requirements

User should have:

- Computer monitor
- Laptop
- Smartphone for wifi connection
- Smart car

3.2 Smart car components:

1. Raspberry Pi 3, Model B:

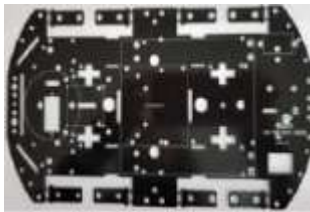







- 900MHz quad-core ARM Cortex-A7
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- HDMI, Ethernet port
- Camera interface (CSI)



2. Sensor Shield

- L298N interface
- IR interface
- INPUT/OUTPUT Pin interface
- Line sensor interface
- Ultrasonic sensor interface



<p>3. Base Frame:</p> 	<p>4. Ultrasonic Frame</p> 
<p>5. Camera frame</p> 	<p>6. 4 DC geared motors:</p> 
<p>7. 2 Line tracers</p> 	<p>8. 2 IR sensors</p> 
<p>9. Voltmeter</p> 	<p>10. Camera module</p> 
<p>11. Ultrasonic sensor</p> 	<p>12. Micro 5- Pin power supply cable</p> 
<p>13. Battery holder</p>	<p>14. 4 Tires</p>



15. 8 or 32 GB Micro SD Card, SD Card Reader



16. Assembly bolt & screws set



17. 2 3.7V batteries



18. 33 Female-Female cable



19. Screwdriver



3.3 Software requirements

User should install:

- VNC Viewer on OS (Windows recommended)
- Raspbian OS on Raspberry Pi

User should:

- update and upgrade Raspbian
- connect needed device (smartphone). Laptop should be connected to the same device (smartphone) through hotspot. PC and Raspberry should be connected to the same network
- enable VNC on Raspberry OS
- enter the IP address of Raspberry Pi to connect to the Raspberry on VNC Viewer

3.3 Code design (example) - Python is used for code

```
import RPi.GPIO as gpio
import time
import sys
import random
```

Here we import necessary libraries like RPi.GPIO (helps write code which deals with pins and the state of pins), time (helps use time-related functions), sys and random (to get random value).

```
def distance():
    gpio.output(38, True)
    time.sleep(0.00001)

    gpio.output(38, False)
    while gpio.input(40) == 0:
        nosig = time.time()

    while gpio.input(40) == 1:
        sig = time.time()

    t1 = sig - nosig
    print(t1 / 0.000058)
    return t1 / 0.000058
```

This distance() function uses ultrasonic sensor with pins for output 38 and for input 40. Pins are set with the help of RPi.GPIO library.

time.sleep() function is responsible for insignificant delay.

In distance() function the time of signal input is measured with time.time(). Then the distance between the sensor and object that was detected is calculated and returned.

```
def init():
    try:
        gpio.setmode(gpio.BOARD)
        gpio.setup(12, gpio.OUT)
        gpio.setup(16, gpio.OUT)
        gpio.setup(18, gpio.OUT)
        gpio.setup(22, gpio.OUT)
        gpio.setup(24, gpio.IN)
        gpio.setup(26, gpio.IN)

        gpio.setup(32, gpio.IN)
        gpio.setup(36, gpio.IN)

        gpio.setup(38, gpio.OUT)
        gpio.setup(40, gpio.IN)
    except Exception as e:
        gpio.cleanup()
        init()
```

The init() function sets up every channel used as an input or an output.

At the end we clean up GPIO channels that our script has used. GPIO.cleanup() also clears the pin numbering system in use.

```
def stop():
    p1.ChangeDutyCycle(0)
    p2.ChangeDutyCycle(0)
    p3.ChangeDutyCycle(0)
    p4.ChangeDutyCycle(0)
```

stop() function stops the car. Duty cycle is the percentage of the ratio of pulse duration – frequency.

```
def reverse(wt, x=100, y=100):
    p1.ChangeDutyCycle(0)
    p2.ChangeDutyCycle(x)
    p3.ChangeDutyCycle(0)
    p4.ChangeDutyCycle(y)
    time.sleep(wt)
```

reverse() function makes the car move in reverse direction. Here p2 and p4 are responsible for the speed in reverse motion.

```
def forward(wt, x=100, y=100):
    p1.ChangeDutyCycle(x)
    p2.ChangeDutyCycle(0)
    p3.ChangeDutyCycle(y)
    p4.ChangeDutyCycle(0)
    time.sleep(wt)
```

forward() function is for forward motion. Here p1 and p3 are responsible for the speed in forward motion.

```
def right(wt, x=100, y=100):
    p1.ChangeDutyCycle(x)
    p2.ChangeDutyCycle(0)
    p3.ChangeDutyCycle(0)
    p4.ChangeDutyCycle(y)
    time.sleep(wt)

def left(wt, x=100, y=100):
    p1.ChangeDutyCycle(0)
    p2.ChangeDutyCycle(x)
    p3.ChangeDutyCycle(y)
    p4.ChangeDutyCycle(0)
    time.sleep(wt)
```

right() and left() functions are for turning motion. Here p1, p4 and p2 and p3 are responsible for the speed in turning motion respectively

```
def ir_sensor():
    if r == 0:
        left(.1, 65, 65)
        print('LEFT')
    elif l == 0:
        right(.1, 65, 65)
        print('RIGHT')
```

ir_sensor() function is used when ir_sensors (tracer) are used. It calls the needed right() or left() functions and it depends which ir_sensor is triggered. The use of this function is in main code below.

```

def obgon():
    side = random.choice([True, False])

    if side:
        right(.5, 40, 40)
    else:
        left(.5, 40, 40)

    forward(.8, 55, 55)
    if not side:
        right(.5, 50, 50)
        forward(.8, 55, 55)
        right(.5, 50, 50)
        forward(1.1, 55, 55)
        left(.5, 50, 50)
    else:
        left(.5, 50, 50)
        forward(.8, 55, 55)
        left(.5, 50, 50)
        forward(1.1, 55, 55)
        right(.5, 50, 50)

    forward(1.3, 55, 55)

    reverse(0.3, 50, 50)
    right(1.02, 50, 50)
    reverse(1.8, 50, 50)

```

obgon() function is created to avoid obstacles in front of the car. This function uses the needed functions above. It also includes parking part.

Main code below:

Here the code starts with calling init() function.

Then we initialize PWM, after we start to set initial value.

Next we use gpio.input() to read value of ir_sensor pins and call ir_sensor() – defined above.

Then we use all other needed functions above to start the car.

The mission of the car from the given code is as follows:

- The car starts forward motion
- The car should detect lines of the map to move inside road lines.
- The car should stop on detecting first obstacle
- The car should avoid the second obstacle and park

To stop the car user can press CTRL+C.

```

init()

p1 = gpio.PWM(12, 100)
p2 = gpio.PWM(16, 100)
p3 = gpio.PWM(18, 100)
p4 = gpio.PWM(22, 100)

p1.start(0)
p2.start(0)
p3.start(0)
p4.start(0)

flag = True

try:
    while True:
        r = gpio.input(26)
        l = gpio.input(24)

        ir_sensor()

        dist = distance()

        if flag and dist < 30:
            while distance() < 30:
                stop()
                time.sleep(.5)
            flag = False
            continue

        if not flag and dist < 30:
            obgon()
            break

        forward(0.015, 40, 40)

        #time.sleep(0.05)
except KeyboardInterrupt as e:
    print(e)

stop()
gpio.cleanup()
print('cleaned')

```

Prioritization and Release Plan

Appendix I: Selection for Cost-Value Approach

Table 1 - Select of 3 most important requirements

Requirement	Aziz K.	Akmal K.	Boburjon B.	Bekhzodkhon M.	Total
Ride straight	10	10	10	10	40
Avoid obstacles	9	10	10	10	40
Turn	8	8	7	9	32
Reverse	5	5	5	5	20
Stop at the traffic light	6	7	5	7	25
Parking	7	7	8	7	29
Rotate smoothly	8	8	6	7	29

Appendix II: Prioritization Result of 10 selected Requirements Using Cost-Value Approach

Table 2 – 3 most important requirements

Requirements ID	Title	Requirement Type
FR1	Ride straight	Function
FR2	Avoid obstacles	Function
FR3	Turn	Function

Table 3 – Value

Value	FR1	FR2	FR3
FR1	1	3	5
FR2	1/3	1	5
FR3	1/5	1/5	1

Table 4 – Cost

Cost	FR1	FR2	FR3
FR1	1	1/3	1/5
FR2	3	1	1/5
FR3	5	5	1

Points scored

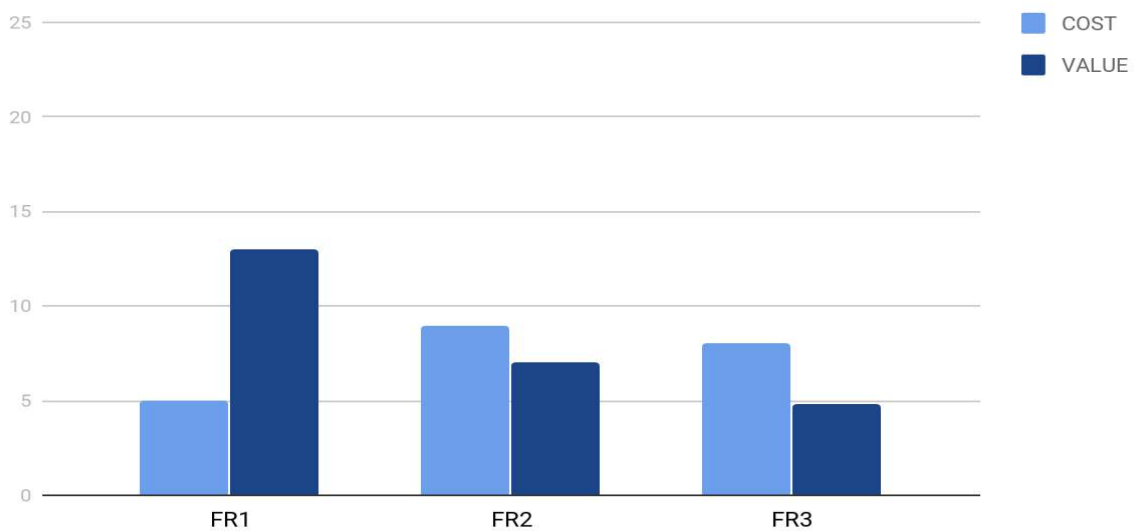


Fig. 1. The value distribution and estimated cost

Table 5 - The value distribution and estimated cost

	FR1	FR2	FR3
COST	5	9	8
VALUE	13	7	4.8

Appendix IV: Release Plan

RE:	Dependencies	Description	Motivation	Release	Duration
FR1	-	Ride straight	This function controls the smart car movement to forward	1	5
FR2	FR1	Avoid obstacles	This function helps to avoid obstacles	1	5
FR3	FR1	Turn	This function turns the car when it is in front of obstacle or when it needs to turn	1	5

Appendix V: I-star

