**GUI FOR ROBOTIC OPERATIONS**

By

Dinesh.R,

Rithik Magaraj.M,

Naveen.M

Department of EIE

PANIMALAR ENGINEERING COLLEGE

The Intern Report is submitted to

CVRDE DRDO, Avadi

upon completion of internship at Robotics Division

From 11/08/2021 to 10/09/2021 under the guidance of

Christina E, Sc‘E’

9th September, 2021

Chennai.

Acknowledgement

First and foremost we sincerely thank The Director, CVRDE for permitting us to undergo training in this organization.

We feel immense pleasure in expressing our gratitude to Mrs. Swarna Ramesh, Sc ‘G’, Addl. Dir (Robotics) and Mr. Radhakrishna KK, Sc’F’ Head(Robotics) for providing us this wonderful internship opportunity which was an extremely enriching learning experience. We are extremely fortunate and thankful to have Mrs. Christina Mary Catherine, Sc ’E’ as our intern mentor and Mrs. Prabhavathy TO ‘B’ who have guided us throughout the internship. We are very appreciative of the commitment shown in providing us all the help and assistance prior to us asking for them.

We have learnt the application of electronics & instrumentation in unmanned technologies in the defense environment.

We have studied the QT based software and written basic codes to design a GUI and also to interface with hardware.

Finally, We extend our sincere thanks to all the Robotics division staff for their help and support during the tenure of our training.

**Table of Contents**

1. Unmanned Ground Vehicle

1.1 Introduction……………………………………………………………………………………………………. 5

1.2 Control System………………………………………………………………………………………………… 6

1.3 UGV Locomotion……………………………………………………………………………………………… 7

1.4 Small UGV’s…………………………………………………………………………………………………….. 7

2. Unmanned Vehicle Communication

2.1 Introduction…………………………………………………………………………………………………….... 7

2.2 Mobility…………………………………………………………………………………………………………….. 8

2.3 Networking……………………………………………………………………………………………………….... 8

2.4 Weaponry…………………………………………………………………………………………………………… 8

3. Commercial UGV

3.1 Self Control Mode…………………………………………………………………………………………..… 9

3.2 Result……………………………………………………………………………………………………………….. 13

3.3 Conclusion and Future Works………………………………………………………………………….... 13

4. ARDUINO

4.1 Introduction…………………………………………………………………………………………………….… 14

4.2 RGB with ARDUINO……………………………………………………………………………………………. 16

4.2.1 Interfacing Arduino with RGB LED……………………………………………………………… 18

4.2.2 Reference code …………………….…………………………………………………………………… 20

4.2.3 Working……………………………………………………………………………………………………… 22

4.3 DS18B20 Digital Temperature Sensor………………………………………………………....……. 23

4.3.1 About DS18B20……………………………………………………………………………….………… 23

4.3.2 Source Code and Output…………………………………………………………………….…….. 24 5. QT Creator

5.1 Introduction…………………………………………………………………………………..………………… 26

5.2 Editor and Tools………………………………………………………………………………………….…….. 26

5.3 QT – RGB LED (GUI)………………………………………………………………………………………….. 26

5.4 QT Temperature Sensor (GUI)………………………………………………………………..………….. 34

5.5 QT Notepad Application Using PYTHON………………………………………………….…....... 39

6. Conclusion…………………………………………………………………………………………………………………………. 42

7. Reference…………………………………………………………………………………………………………….……………. 42

**List of Figures**

3.1 Block Diagram of Self Control Mode……………………………………………………………………………………… 10

3.2 Flow Chart of Unmanned Control Vehicle…………………………………………………………………………….. 11

3.3 Obstacle Detection Algorithm Codes For LEFT SIDE ‘0’………………………………………………………….. 12

3.4 Obstacle Detection Algorithm Codes For LEFT SIDE ‘1’ ……………………………………………………….. 13

4.1 RGB LED with ARDUINO……………………………………………………………………………………………………….. 17

4.2 Interfacing RGB LED with Arduino……………………………………………………………………………………… 18

4.3 Circuits Connections of Arduino RGB LED Interface……………… ……………………………………………. 19

4.4 Overview of DS18B20S………………………………………………………………………………………………………….. 24

4.5 Serial Monitor……………………………………………………………………………………………………………………….. 25

5.1 Qt Creator Window……………………………………………………………………………………………………………….. 26

1. **Unmanned Ground Vehicles**

**1.1 INTRODUCTION**

In the present scenario, the recent increase in terrorism has carved the way to develop self controlled robots tremendously. This motivation has challenged the research scholars to invent efficient robots for the war fields. These self-controlled robots counter the emergency problems that present day people and soldiers are facing. For example, self-robots work in handy parallel with the soldiers in tracking the terrorist activities and can also work with police to help them find illegal activities occurring in the city and to find the culprits indulged in thefts and other antisocial activities. Many countries are now aware of the serious treat issues and are investing in this development of the self-controlled robots to safeguard citizens from terrorism. To be more precise, unmanned ground vehicles operate without human interference, i.e. artificial intelligence comes into the need for taking self decisions and navigate according to these decisions. This self decision of the robot can be achieved with the help of GPS coordinates, magnetic compass to detect the obstacles in the mid journey and to self-plan the new route by the robot itself Unmanned ground vehicle is based on two basic robot technologies combined into one unit and make it work effectively. The first one is the DRDO Daksh robot, which is controlled remotely in locating hazardous objects safely and destroying them. This is achieved by the X-ray machine to locate metals and LED’s. It is also used for climbing staircases and scan vehicles for explosives. The second one is a Foster-miller TALON robot used to navigate through sand, water and even to climb trees. So all these features are added up along with the self-controlled operations to create a multitask robot which works effectively using path planning and obstacle detection. If it works as planned, the world can see a new era of robots with its new defense system working for the betterment of society and making earth a better place to live in.

**Abstract**

Unmanned ground vehicle (UGV) is also called as self-controlled robot which plays an important role in military purposes, however the demand for manual robots is on rise. UGV work in more efficient way to counter terrorism and in far fields. Unmanned ground vehicle support and augment soldiers' lives in battle fields. The capability of this robot mainly depends on protecting the soldiers or at least reducing the death tolls in wars. Many researches are extensively carried out in major parts of the world to develop this prototype which work more efficiently in military activities and in counter terrorism. This Unmanned ground vehicle effectively works in the fields of border patrol, surveillance and in war combats. This UGV mainly works in two different modes, Automatic or self mode and Manual or in coordination with human’s. In self mode , UGV is tasked to survey the border between the two areas without human instructions. This feature can be attained by implementing GPS, Magnetic compass to measure the turning angles and adjust the route avoiding the obstacles such as mountains, trees and drivers. This path planning strategy is the key program basic to work the robot in the self-controlled mode. The complete working and setup are described in the paper.

**Keywords- GPS - Geographical positioning system, Magnetic Compass, Self-controlled mode, unmanned ground vehicle.**

Unmanned ground vehicles (UGVs) are robotic systems that operate on land without an onboard human operator. They are used for a wide variety of both civilian and military applications,00 particularly in environments that are hazardous or unpleasant to humans and for tasks that are difficult, dull or pose unacceptable risks.

UGV platforms may be designed from scratch or based on an existing manned land vehicle such as a Land Rover. Vehicle kits are also available for converting manned vehicles into UGVs, and may provide optionally manned capabilities that allow users to select crewedor unscrewed operation.

### **1.2 Control systems**

Unmanned ground vehicles are generally considered Remote-Operated and Autonomous, although Supervisory Control is also used to refer to situations where there is a combination of decision making from internal UGV systems and the remote human operator.

### **1.3 UGV Locomotion**

The three main UGV locomotion methods are wheels, tracks and legs. Wheels are power-efficient and allow the highest speeds on flat ground, but are not good for traversing off-road and uneven terrain, as they can get stuck or sink into the ground due to low contact surface area and thus higher pressure. Tracks are the best option for rugged terrain, but are slower, less efficient, involve more mechanical complexity and cause more vibration. Legged ground robots can cope with a wide variety of terrain, but are limited in speed and require complex control and stability hardware.

### **1.4 Small UGVs**

Small UGVs can be powered by electric batteries. Larger ones may use a petrol or diesel engine, or a hybrid system that uses a combustion engine to drive an electric generator rather than directly connecting to the locomotion system. Hydrogen fuel cell power systems for UGVs are also under development.

# 2.Unmanned Vehicle Communication

# 2.1 Introduction

Wireless communications are required to operate UGVs remotely, as well as to relay video footage and other sensor data. This will typically be done by RF (radio frequency) communications, satellite links, or wireless fiber optic. UGVs will often require non-line of sight (NLOS) communications due to operating in urban and cluttered environments. Mobile ad hoc network (MANET) technology is often utilized in order to help UGVs maintain links even in adverse conditions.

UGVs can be equipped with a variety of sensors and payloads. Due to operating in indoor and other GNSS-denied environments, UGVs may rely on LiDAR sensors, combined with inertial navigation systems and vehicle odometry, for accurate navigation. Mission-specific sensors and payloads include RGB and thermal cameras, manipulator arms, chemical and explosives sensors, and weapons systems.

### **2.2 Mobility**

Tracked combat vehicles are suited for heavy combat and rough terrain. Wheeled combat vehicles offer improved logistical mobility and optimized speeds on smooth terrain.

Silent watch is becoming an increasingly important combat vehicle application. It is a role that requires that all mission requirements be met while keeping acoustic and infrared signature levels to a minimum. For this reason, silent watches often require the vehicle to operate without use of the main engine and sometimes even auxiliary engines. Many modern combat vehicles often have electronic equipment that cannot be supported solely with auxiliary batteries alone. Auxiliary fuel cells are a potential solution for covert operations.

### **2.3 Networking**

Force trackers are not as prevalent as in the [air force](https://en.wikipedia.org/wiki/Air_force) but are still essential components of combat vehicles.

In the mid-'90s, U.S. weapon developers envisioned a sophisticated communication network where positions of enemy and friendly forces could be relayed to [command vehicles](https://en.wikipedia.org/w/index.php?title=Command_vehicle&action=edit&redlink=1) and other friendly vehicles. Friendly vehicles could transmit enemy positions to friendly combat vehicles in combat range for efficient annihilation of the enemy. Logistics support could also monitor front-line combat vehicle fuel and ammunition statuses and move in to resupply depleted vehicles.

### **2.4 Weaponry**

Combat vehicles weapons are primarily designed for engaging other vehicles at long distances and are not optimized for destroying non-traditional targets like [car bombs](https://en.wikipedia.org/wiki/Car_bomb).

# 3.Commercial UGVs

UGVs are used for a wide range of civilian applications such as urban search and rescue, firefighting, nuclear plant operations, crowd control and agricultural spraying and harvesting.

### **UGV Military**

### Military UGV uses include EOD (explosive ordnance disposal), equipment carrying, forward reconnaissance, mobile weapons platforms and manned-unmanned teaming. Many UGV platforms can be quickly reconfigured to perform multiple roles.

**3.1 SELF CONTROL MODE**

An autonomous robot can work without human intervention. It gains information about the climatic conditions and operates according to it using GPS coordinates. It learns new knowledge frequently about the obstacle detection and how to avoid them by planning a new path using the magnetic compass to measure the angles. It parallely adapts to the changing surroundings for accomplishing the new tasks. The block diagram for the self-controlled

**1. A). BLOCK DIAGRAM OF SELF CONTROL MODE**

The specific function of each block in the figure are explained in detail.

**1). COMMAND CONTROL STATION:** The UGV is located away from the command control station and is controlled by keyboard and mouse for movement of the vehicles, monitoring the live video feedback and switching between the autonomous and manual modes.

**2). INTERNET:** This acts as a medium for the flow of commands from control station to the UGV to make it work wireless.

**3). ON-BOARD SYSTEM:** The computer in the UGV receives the commands from the control systems and operates accordingly.

**4). CAMERA:** The camera records the video of the path travelled by the UGV and sends it to the control station for the users to access it.

**5). CONTROL UNIT:** It is usually the micro controller that receives the signals from the user and other sensors and operates according to the signals such as UGV movement and angle, obstacle detections. 6). GPS UNIT: Used for the autonomous mode for obtaining the source and destination places for the UGV to travel between these places and to plan the required path accordingly.

**7). MAGNETIC COMPASS:** It is used to obtain the angle for the UGV to take a turn towards the destination coordinates.

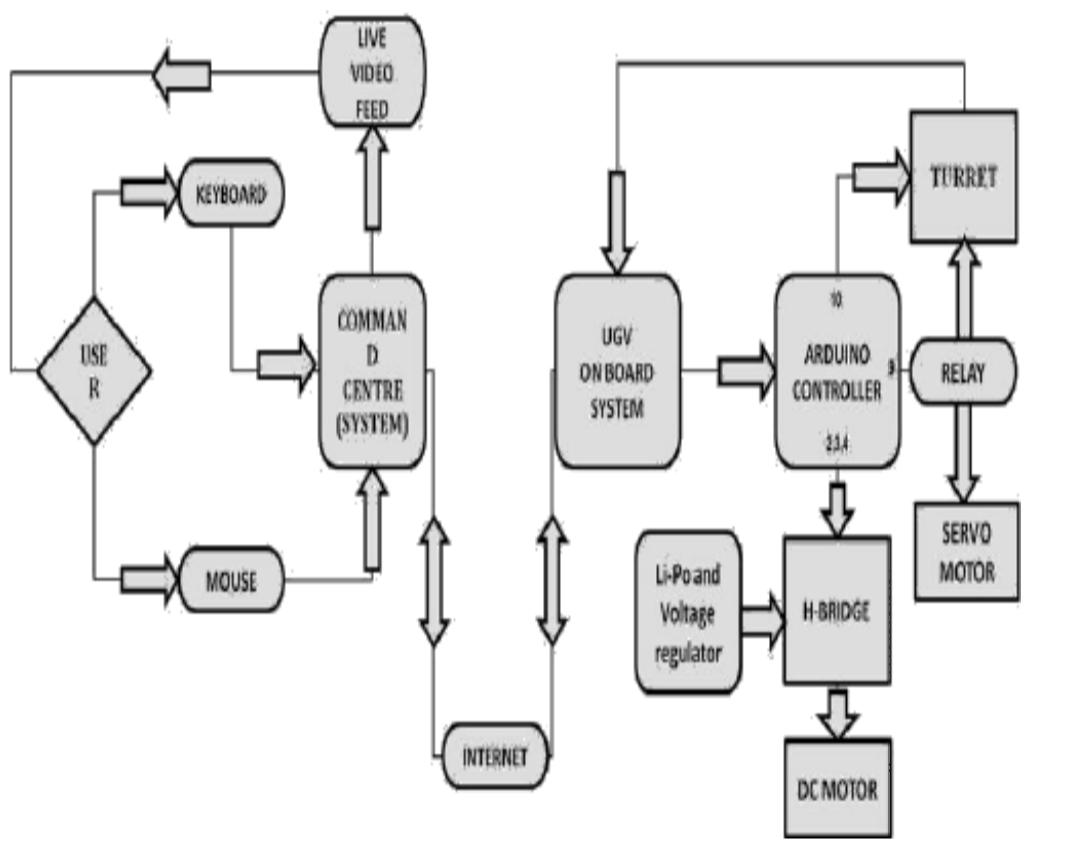
**8). IR SENSORS:** Infrared sensors are used for detecting metal objects like bombs, mines in the autonomous mode.

**9). SERVO MOTOR:** This is used to control the direction of the Unmanned control vehicle.

**10). DC MOTOR:** It is mainly used for UGV movement from one place to other place.

**11). WIRELESS MODEM:** Raspberry pi or zigbee module provides wireless data transfer to the control station from the on-board system.

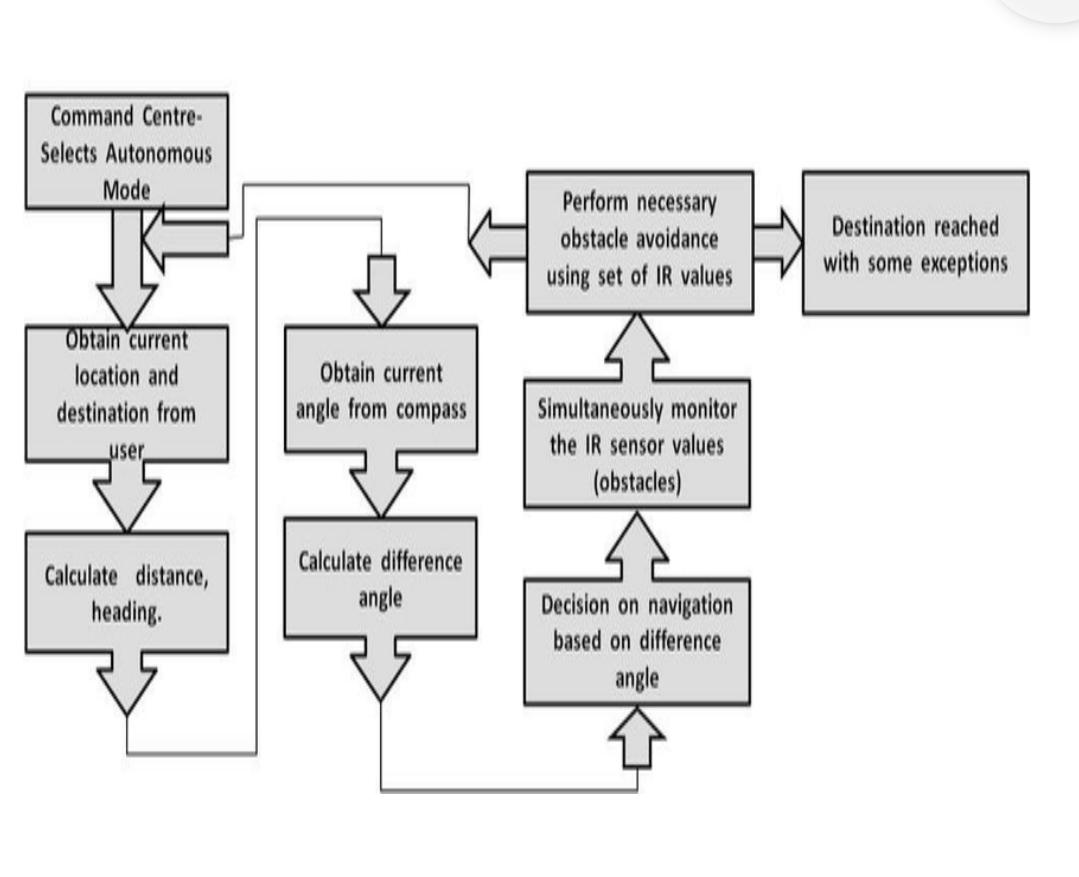
**12). BATTERY:** Used for powering up the control unit and the zigbee module.



**3.1 BLOCK DIAGRAM OF SELF CONTRO MODE**

**HARDWARE COMPONENTS REQUIRED FOR UGV**

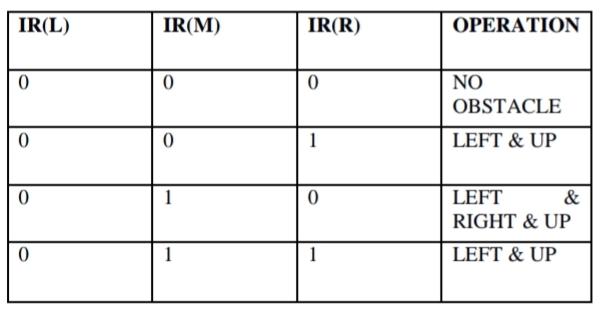
* Arduino microcontroller
* Servo motor
* DC motor
* ZigBee radio modem
* Magnetic compass
* GPS system
* Battery
* Obstacle detector
* Water detector
* Human sensor
* Camera
* IR Sensor



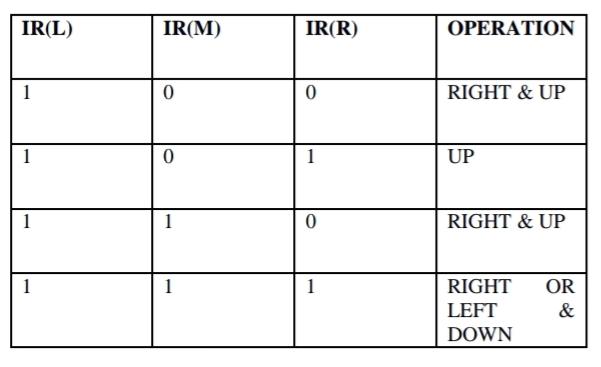
**3.2 FLOW CHART OF UNMANNED CONTROL VEHICLE**

**B). ALGORITHM DESIGN FOR SELF CONTROL MODE:**

The algorithm design for self-control mode is mainly achieved by two important factors called path planning and obstacle detection algorithms for the UGV to navigate accordingly. The user from the command station obtains the current GPS coordinates of the UGV and also the destination coordinates to where UGV has to navigate. The directions are obtained by the magnetic compass by trigonometric functions by calculating the specific angle required by the UGV to move accordingly. Thus the path planning algorithm decides the path taken by the UGV taking into account all these measurements. Now, obstacle detection is done by the IR sensors that detect the obstacles like mountains, rivers and thus in parallel update the path planning measurements accordingly. It also detects metals like bombs, metal objects those are buried in the ground and send this information to the command control station. Thus in self-control mode, path planning and obstacle detection algorithms have to work in parallel in the most efficient manner based on all the measurements. On the other side command control station obtains all the measurements and signals from the UGV and coordinates the robot continuously. Based on the path planning and obstacle detection algorithms the UGV navigates to the destination given by the user itself automatically. The obstacle detection algorithm works according to following data shown in figure 3.3 and figure 3.4



**FIG 3.3 : OBSTACLE DETECTION ALGORITHM CODES FOR LEFT SIDE ‘0’**



**FIG 3.4: OBSTACLE DETECTION ALGORITHM CODES FOR LEFT SIDE ‘1’**

**3.2 RESULTS**

We successfully built an unmanned ground vehicle which functions automatically using magnetic compass, GPS coordinates, path planning and obstacle detection algorithms. Technologies like zigbee, arduino, water sensor, human sensor, metal sensor are being used and implemented effectively.

**3.3 CONCLUSION AND FUTURE WORKS**

We successfully built a prototype that works automatically in self-control mode using GPS antenna, magnetic compass, path planning and obstacle detection algorithms. In this self-control mode, UGV is capable of travelling automatically from one place to another without human interference and navigation commands. It adjusts its strategies and measurements asked on path planning, obstacle detection, GPS antenna and magnetic compass. We strongly feel, self control robots are very helpful for military purposes that are operated outdoors. This UGV can undertake operations like surveillance, border patrol and in active combat both as a standalone unit or automatic mode as well as in coordination with human soldiers or in manual mode. Our future work is on developing arm controlled mode or gesture controlled mode along with the command control mode and automatic mode.

# 4.ARDUINO

**4.1 INTRODUCTION**

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

**WHY ARDUINO?**

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Net media’s BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than 500 Rs

Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.

Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

**4.2 RGB LED with Arduino**

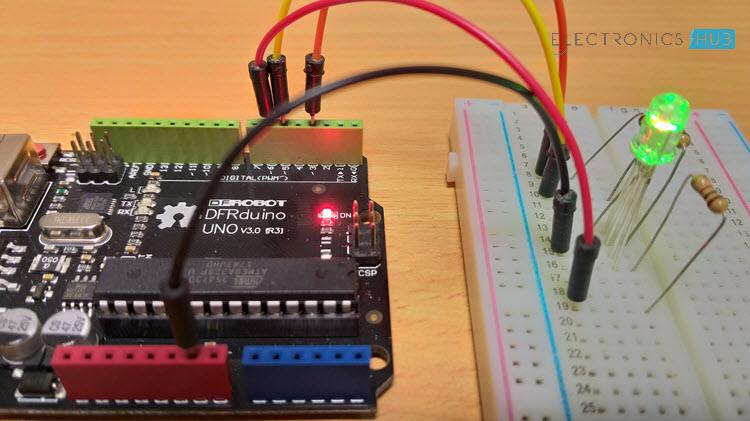
In this project, we will discuss what an RGB LED is, how to interface an RGB LED with Arduino and finally, how to drive an RGB LED using Arduino UNO

**Introduction**

In this program, we will Blink an LED on and off at a certain delay.

We will begin working around LEDs. Coming to an LED, it is short for Light Emitting Diode, which is a semiconductor device that emits light when current passes through it.

There are several types of LEDs available today and RGB LED is one such fascinating variant of LED.



**4.1 RGB LED with ARDUINO**

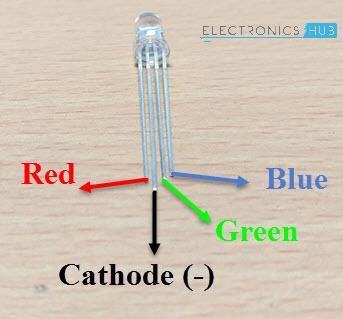
### **What is an RGB LED?**

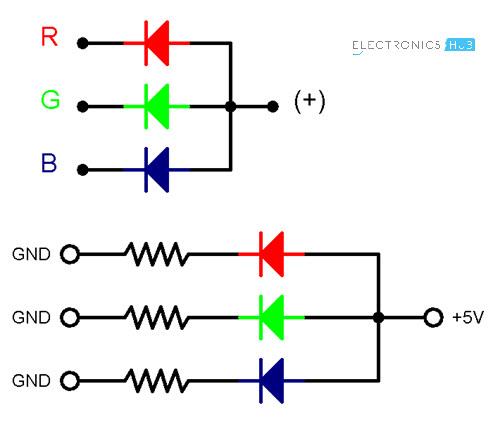
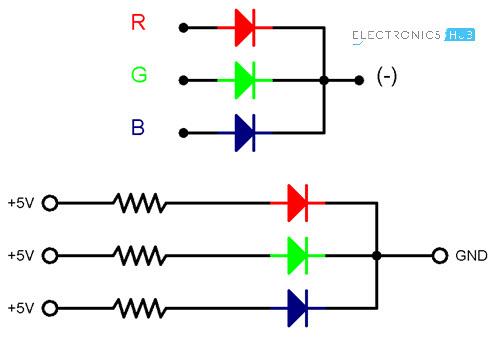
RGB LED is a type LED which emit multiple colors i.e. Red, Green and Blue to be specific. Hence, it is called RGB LED (RGB stands for Red, Green and Blue). Appearance wise, an RGB LED looks very similar to a regular LED except that an RGB LED has three LEDs, each for Red, Green and Blue lights and all these are housed in a single package.

Since an RGB LED consists of three individual LEDs in a single package, the number of leads is different from that of a regular LED, which has two leads (one for cathode and the other for anode).

The through-hole variant of an RGB LED has 4 leads: one lead for each individual color (Red, Green and Blue) and the fourth one is the common lead (which can be either a cathode or anode).

Generally, the common cathode variant is found more frequently than the common anode variant. In this project, I will use a Common Cathode type RGB LED. I will show you how to drive both the variants.





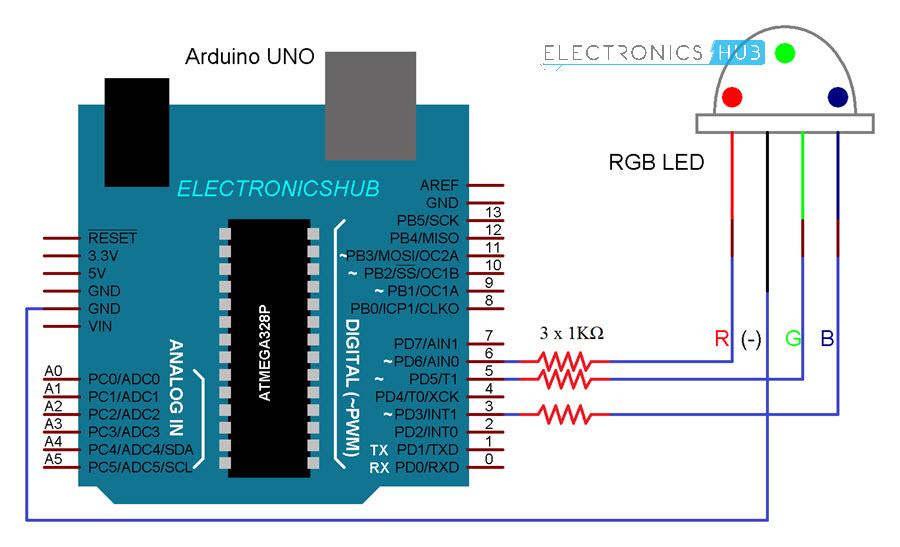
### **4.2 Interfacing RGB LED with Arduino**

### **4.2.1 Interfacing RGB LED with Arduino**

Now, let me take you through the steps involved in interfacing an RGB LED with Arduino UNO.

First, an important point is that, since we will be using the PWM technique to control the brightness of the LEDs (Red, Green and Blue separately), make sure that you connect the Red, Green and Blue Anodes of the RGB LED to three PWM supported pins of Arduino (assuming you have a common cathode RGB LED).

The following image shows the circuit connections of Arduino RGB LED Interface. A Common Cathode type RGB LED is used here.



# 4.3 Circuit Connections of Arduino RGB LED Interface

#### Components

* Arduino UNO
* RGB LED
* 1KΩ Resistor x 3
* Mini Breadboard
* Connecting Wires

PWM Pins 6, 5 and 3 are used in the project and they are connected to Red, Green and Blue anode terminals of the RGB LED through individual current limiting resistors (1KΩ).

The common terminal i.e. the cathode is connected to the GND pin of Arduino.

**4.2.2 Reference Code**

The code for interfacing RGB LEDs with Arduino is given below.

| constintredPin = 6; |
| --- |
| constintgreenPin = 5; |
| constintbluePin = 3; |
|  |
| voidsetup() |
| { |
| pinMode(redPin,OUTPUT); |
| pinMode(greenPin,OUTPUT); |
| pinMode(bluePin,OUTPUT); |
| } |
|  |
| voidloop() |
| { |
| delay(1000); |
| primaryColors(1,0,0); // Red |
| delay(2000); |
| primaryColors(0,1,0); // Green |
| delay(2000); |
| primaryColors(0,0,1); // Blue |
| delay(2000); |
| primaryColors(1,1,0); // Yellow |
| delay(2000); |
| primaryColors(1,0,1); // Magenta |
| delay(2000); |
| primaryColors(0,1,1); // Cyan |
| delay(2000); |
| primaryColors(1,1,1); // White |
| delay(2000); |
|  |
| RGBFading(); |
| } |
| voidprimaryColors(intredValue, intgreenValue, intblueValue) |
| { |
| digitalWrite(redPin, redValue); |
| digitalWrite(greenPin, greenValue); |
| digitalWrite(bluePin, blueValue); |
| } |
|  |
| voidRGBFading() |
| { |
| int x; |
| intredBrightness; |
| intgreenBrightness; |
| intblueBrightness; |
|  |
| for (x = 0; x <768; x++) |
| { |
| if (x <= 255) |
| { |
| redBrightness = 255 - x; |
| greenBrightness = x; |
| blueBrightness = 0; |
| } |
| elseif (x <= 511) |
| { |
| redBrightness = 0; |
| greenBrightness = 255 - (x - 256); |
| blueBrightness = (x - 256); |
| } |
| Else |
| { |
| redBrightness = (x - 512); |
| greenBrightness = 0; |
| blueBrightness = 255 - (x - 512); |
| } |
|  |
| analogWrite(redPin, redBrightness); |
| analogWrite(bluePin, blueBrightness); |
| analogWrite(greenPin, greenBrightness); |
|  |
| delay(10); |
| } |
| } |

### **4.2.3 Working:**

After making the connections as per the circuit diagram, copy the code and upload it to Arduino UNO. Initially, the RGB LED display individual colors Red, Green and Blue then followed by combination colors Yellow, Magenta and Cyan. White is also displayed.

Following this, a continuous display of the color spectrum possible with RGB Primary colors is implemented using PWM technique.

# 4.3 DS18B20 digital temperature sensors with Arduino:

## **4.3.1 About the DS18B20 1-Wire temperature sensor**

The DS18B20 is a digital temperature sensor manufactured by [Maxim Integrated](https://www.maximintegrated.com/en.html) (formerly Dallas Semiconductor). It is one of the most popular temperature sensors on the market and provides fairly high accuracy (±0.5 °C) over a large temperature range (-55 °C to + 125 °C). Because the operating voltage of the sensor is 3.0 to 5.5 V, you can use it with both the Arduino (which operates at 5 V), as well as with devices like the ESP32 and Raspberry Pi which have 3.3 V GPIO pins.

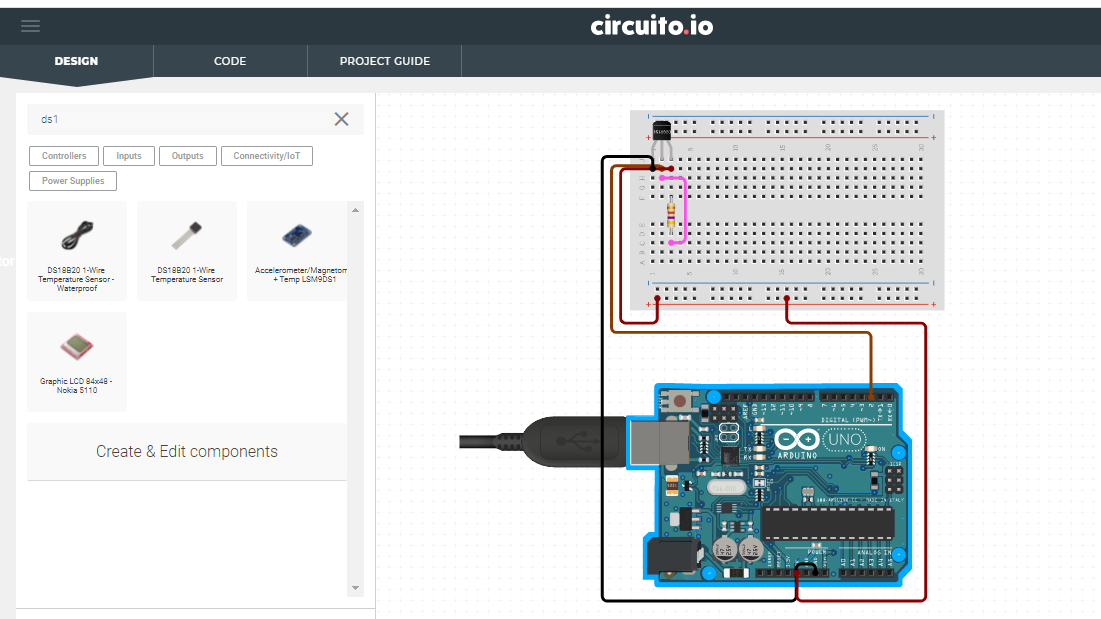
Advantage is that each DS18B20 sensor has a unique 64-bit serial code, which allows multiple sensors to function on the same 1-Wire bus. So you can read data from multiple sensors that are connected together with just one Arduino pin (see code examples below).

The resolution of the sensor can be set programmatically to 9, 10, 11, or 12 bits. This corresponds to temperature increments of 0.5 °C, 0.25 °C, 0.125 °C, and 0.0635 °C, respectively. The default resolution at power-up is 12-bit.

## **DS18B20 temperature sensor with Arduino example code**

With the following example code, you can read the temperature from a DS18B20 sensor and display it in the Serial Monitor.

You can upload the example code to your Arduino using the [Arduino IDE](https://www.arduino.cc/en/main/software).

****

**4.4 Overview DS18B20S**

**4.3.2 SOURCE CODE**

**#include <OneWire.h>**

**#include <DallasTemperature.h>**

**#define ONE\_WIRE\_BUS 2**

OneWireoneWire(ONE\_WIRE\_BUS);

DallasTemperaturesensors(&oneWire);

**void**setup(){

Serial.begin(9600);

sensors.begin();

}

**void**loop(){

sensors.requestTemperatures();

**float**tempC = sensors.getTempCByIndex(0);

**float**tempF = sensors.getTempFByIndex(0);

Serial.print(“Temperature: “);

Serial.print(tempC);

Serial.print(“ \xC2\xB0”);Serial.print(“C | “);

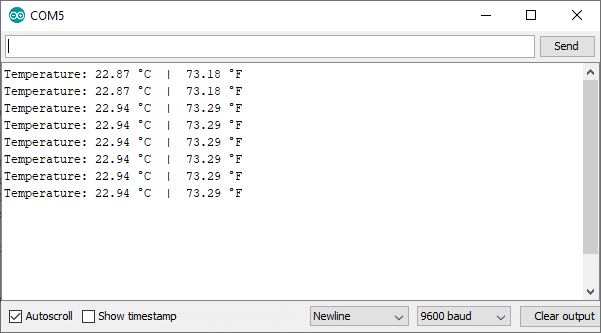
Serial.print(tempF);

Serial.print(“ \xC2\xB0”);Serial.println(“F”);

delay(1000);

}

**OUTPUT**



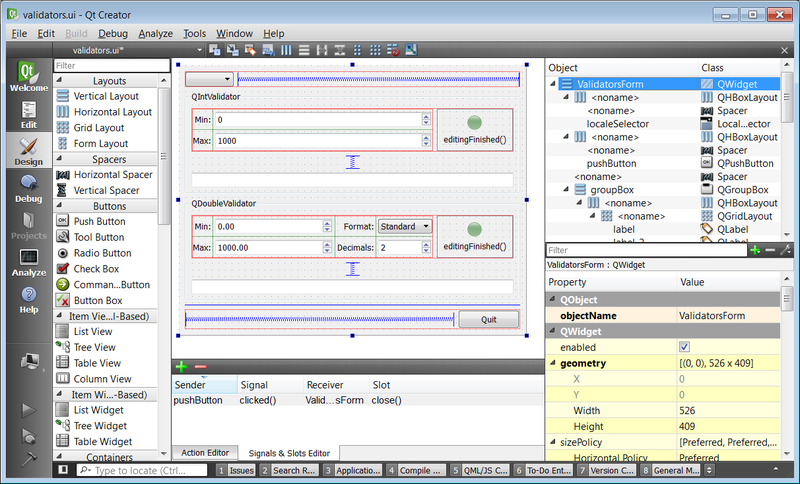
**4.5 Serial Monitor**

Qt Creator logo.png

**5. Qt Creator**

**5.1 Introduction**

Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which simplifies GUI application development. It is part of the SDK for the Qt GUI application development framework and uses the Qt API, which encapsulates host OS GUI function calls.



**5.1 Qt Creator Window**

**5.2 EDITORS**

Qt Creator includes a code editor and integrates Qt Designer for designing and building graphical user interfaces (GUIs) from Qt widgets.

The code editor in Qt Creator supports syntax highlighting for various languages. In addition to that, the code editor can parse code in C++ and [QML](https://en.wikipedia.org/wiki/Qt_Meta_Language) languages and as a result code completion, context-sensitive help, semantic navigation are provided.

Qt Designer is a tool for designing and building graphical user interfaces (GUIs) from Qt widgets. It is possible to compose and customize the widgets or dialogs and test them using different styles and resolutions directly in the editor. Widgets and forms created with Qt Designer are integrated with programmed code, using the Qt signals and slots mechanism.

Qt Quick Designer is a tool for developing animations by using a declarative programming language

**TOOLS**

Qt Creator is integrated with a set of tools, such as version control systems and Qt Simulator.

**PROJECTS**

Qt Creator includes a project manager that can use a variety of project formats such as [.pro](https://en.wikipedia.org/wiki/Qmake), [CMake](https://en.wikipedia.org/wiki/CMake), [Autotools](https://en.wikipedia.org/wiki/GNU_Build_System) and others. A project file can contain information such as what files are included into the project, custom build steps and settings for running the applications.

**5.3 Qt RGB-LED**

**MAIN FILE**

#define red\_led 9

#define green\_led 10

#define blue\_led 11

void setup()

{

pinMode(red\_led, OUTPUT);

pinMode(green\_led, OUTPUT);

pinMode(blue\_led, OUTPUT);

analogWrite(red\_led, 255);

analogWrite(green\_led, 255);

analogWrite(blue\_led, 255);

Serial.begin(9600);

}

void loop()

{

if (Serial.available())

{

charled\_specifier = Serial.read();

intled\_brightness = Serial.parseInt();

write\_leds(led\_specifier, led\_brightness);

}

}

voidwrite\_leds(char led, int brightness)

{

if (led == 'r'){

analogWrite(red\_led, 255 - brightness);

return;

}

if (led == 'g')

{

analogWrite(green\_led, 255 - brightness);

return;

}

if (led == 'b')

{

analogWrite(blue\_led, 255 - brightness);

return;

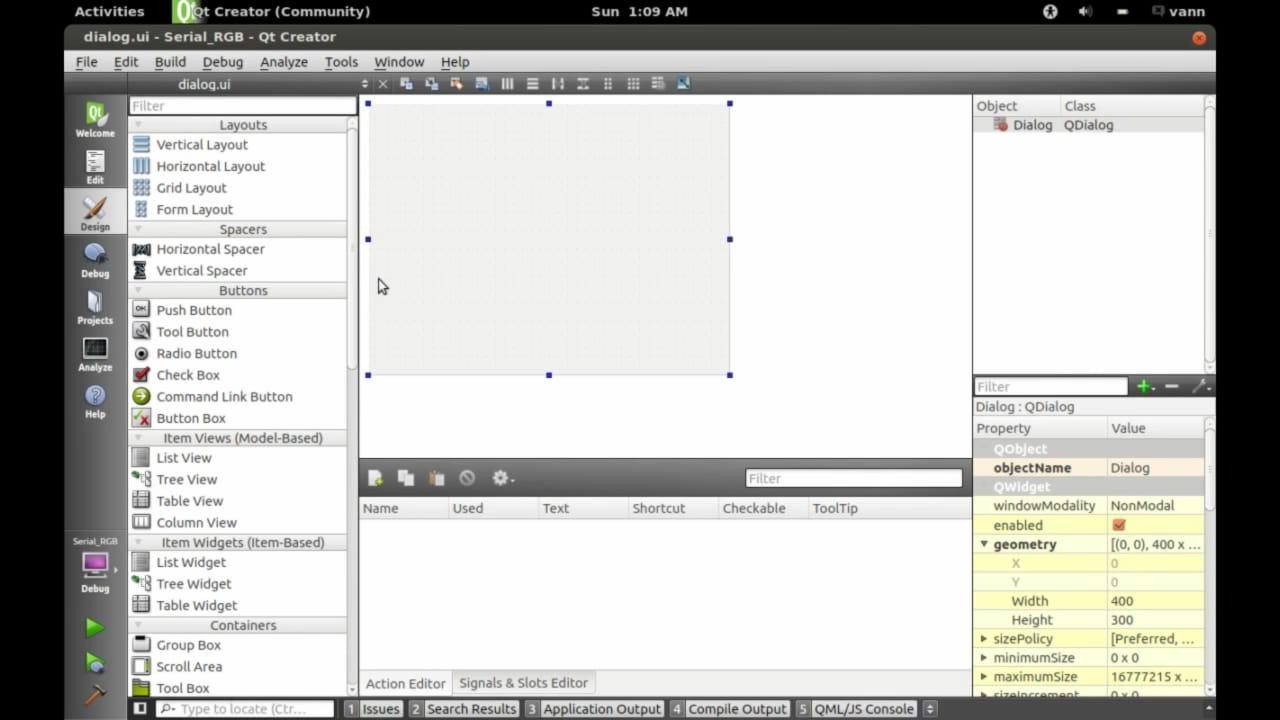
}

return;

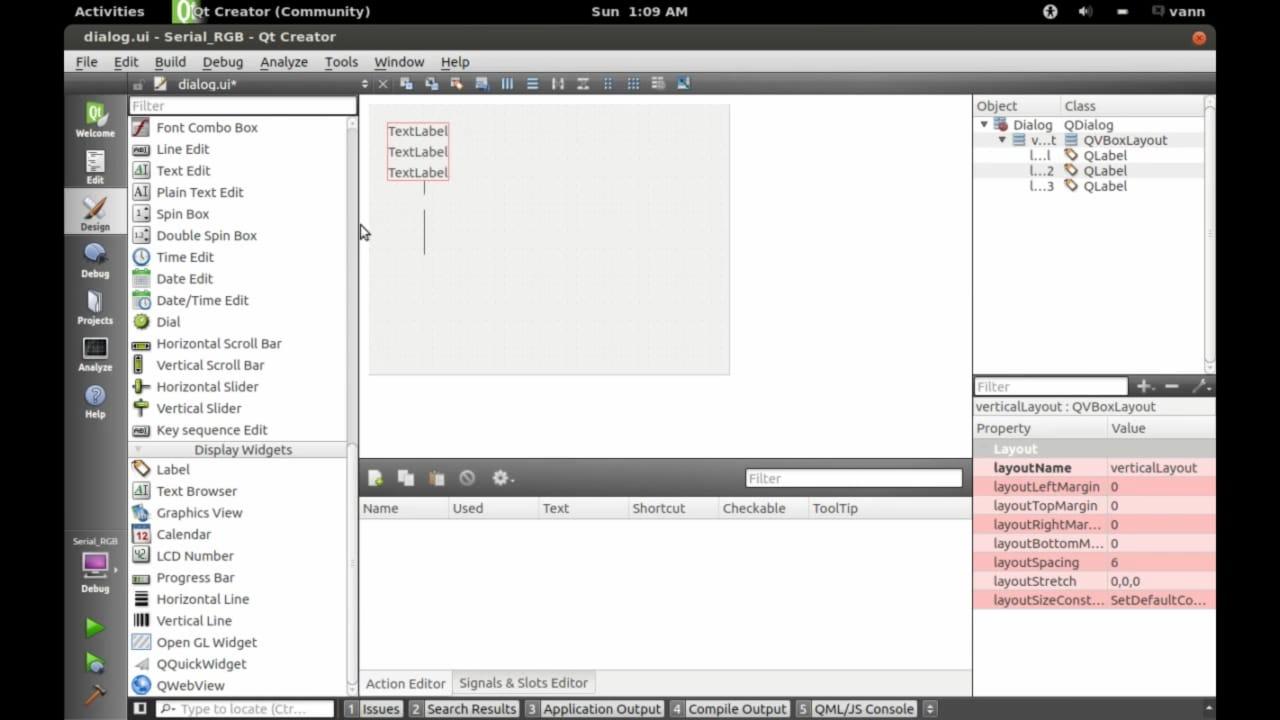
}

**PROCEDURE:**

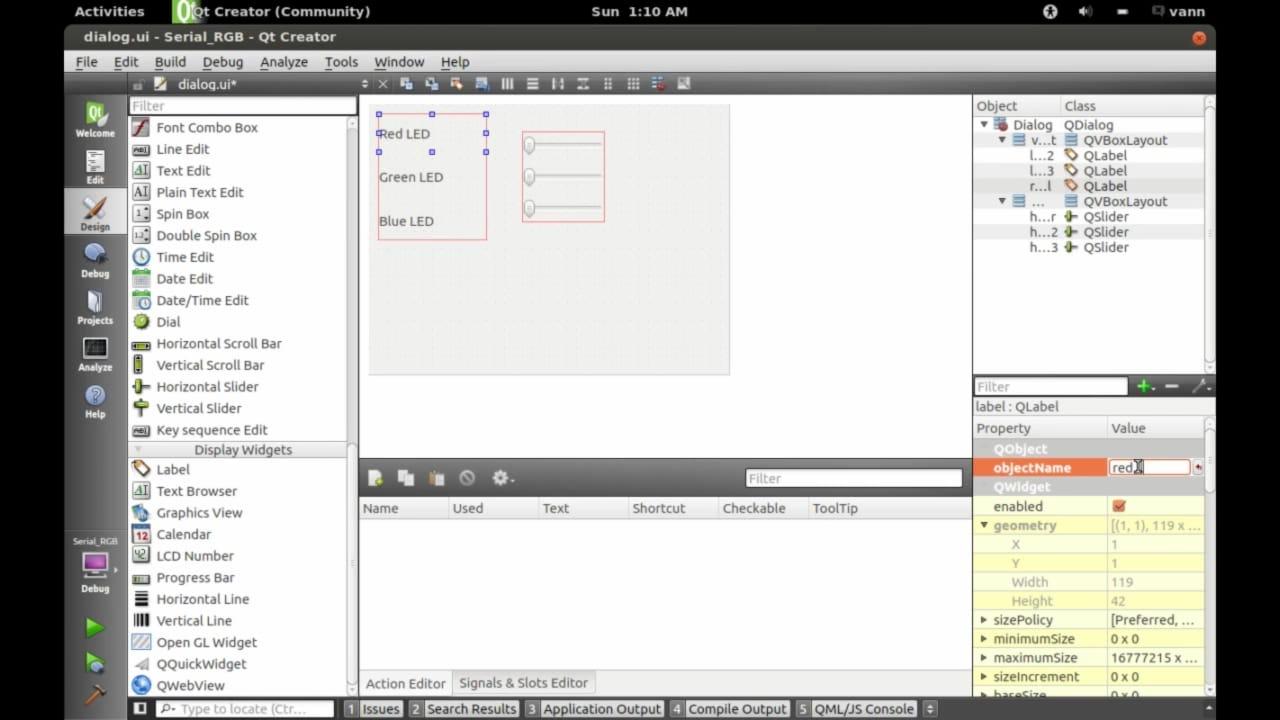
**STEP 1 CREATING UI INTERFACE**



**STEP 2**



**STEP 3**



**DIALOG.UI**

#include "dialog.h"

#include "ui\_dialog.h"

#include <QSerialPort>

#include <QSerialPortInfo>

#include <QDebug>

#include <QtWidgets>

Dialog::Dialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Dialog)

{

ui->setupUi(this);

arduino\_is\_available = false;

arduino\_port\_name = "";

arduino = new QSerialPort;

/\*

qDebug() << "Number of available ports: " <<QSerialPortInfo::availablePorts().length();

foreach(constQSerialPortInfo&serialPortInfo, QSerialPortInfo::availablePorts()){

qDebug() << "Has vendor ID: " <<serialPortInfo.hasVendorIdentifier();

if(serialPortInfo.hasVendorIdentifier()){

qDebug() << "Vendor ID: " <<serialPortInfo.vendorIdentifier();

}

qDebug() << "Has Product ID: " <<serialPortInfo.hasProductIdentifier();

if(serialPortInfo.hasProductIdentifier()){

qDebug() << "Product ID: " <<serialPortInfo.productIdentifier();

}

}

\*/

foreach(constQSerialPortInfo&serialPortInfo, QSerialPortInfo::availablePorts()){

if(serialPortInfo.hasVendorIdentifier() &&serialPortInfo.hasProductIdentifier()){

if(serialPortInfo.vendorIdentifier() == arduino\_uno\_vendor\_id){

if(serialPortInfo.productIdentifier() == arduino\_uno\_product\_id){

arduino\_port\_name = serialPortInfo.portName();

arduino\_is\_available = true;

}

}

}

}

if(arduino\_is\_available){

// open and configure the serialport

arduino->setPortName(arduino\_port\_name);

arduino->open(QSerialPort::WriteOnly);

arduino->setBaudRate(QSerialPort::Baud9600);

arduino->setDataBits(QSerialPort::Data8);

arduino->setParity(QSerialPort::NoParity);

arduino->setStopBits(QSerialPort::OneStop);

arduino->setFlowControl(QSerialPort::NoFlowControl);

}else{

// give error message if not available

QMessageBox::warning(this, "Port error", "Couldn't find the Arduino!");

}

}

Dialog::~Dialog()

{

if(arduino->isOpen()){

arduino->close();

}

deleteui;

}

void Dialog::on\_red\_slider\_valueChanged(int value)

{

ui->red\_value\_label->setText(QString("<span style=\" font-size:18pt; font-weight:600;\">%1</span>").arg(value));

Dialog::updateRGB(QString("r%1").arg(value));

qDebug() << value;

}

void Dialog::on\_green\_slider\_valueChanged(int value)

{

ui->green\_value\_label->setText(QString("<span style=\" font-size:18pt; font-weight:600;\">%1</span>").arg(value));

Dialog::updateRGB(QString("g%1").arg(value));

qDebug() << value;

}

void Dialog::on\_blue\_slider\_valueChanged(int value)

{

ui->blue\_value\_label->setText(QString("<span style=\" font-size:18pt; font-weight:600;\">%1</span>").arg(value));

Dialog::updateRGB(QString("b%1").arg(value));

qDebug() << value;

}

void Dialog::updateRGB(QString command)

{

if(arduino->isWritable()){

arduino->write(command.toStdString().c\_str());

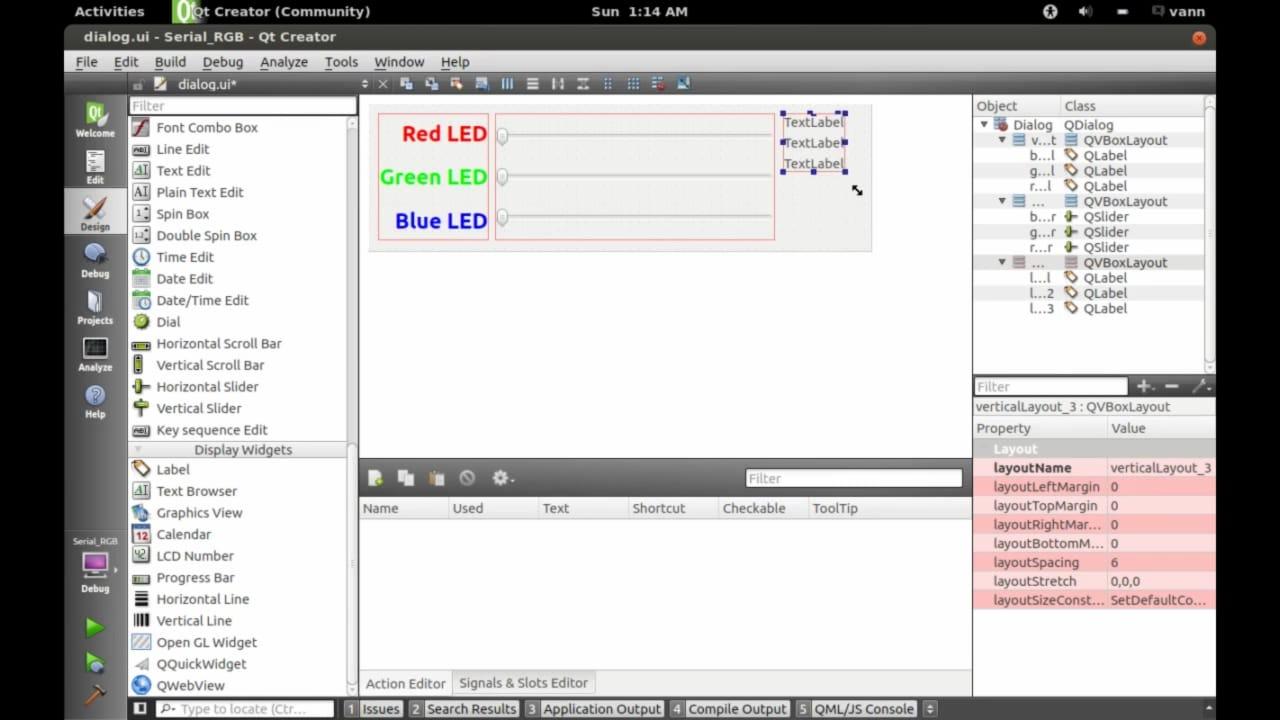
}else{

qDebug() << "Couldn't write to serial!";

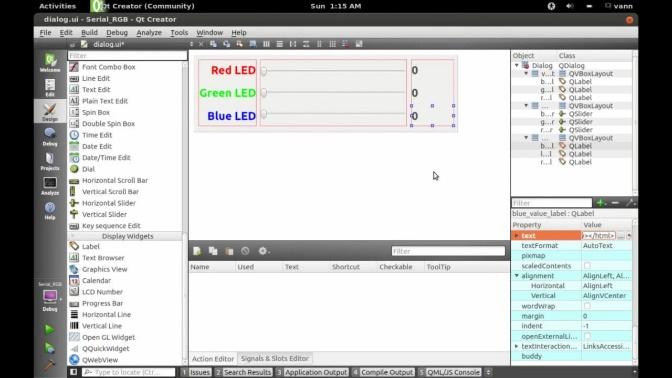
}

}

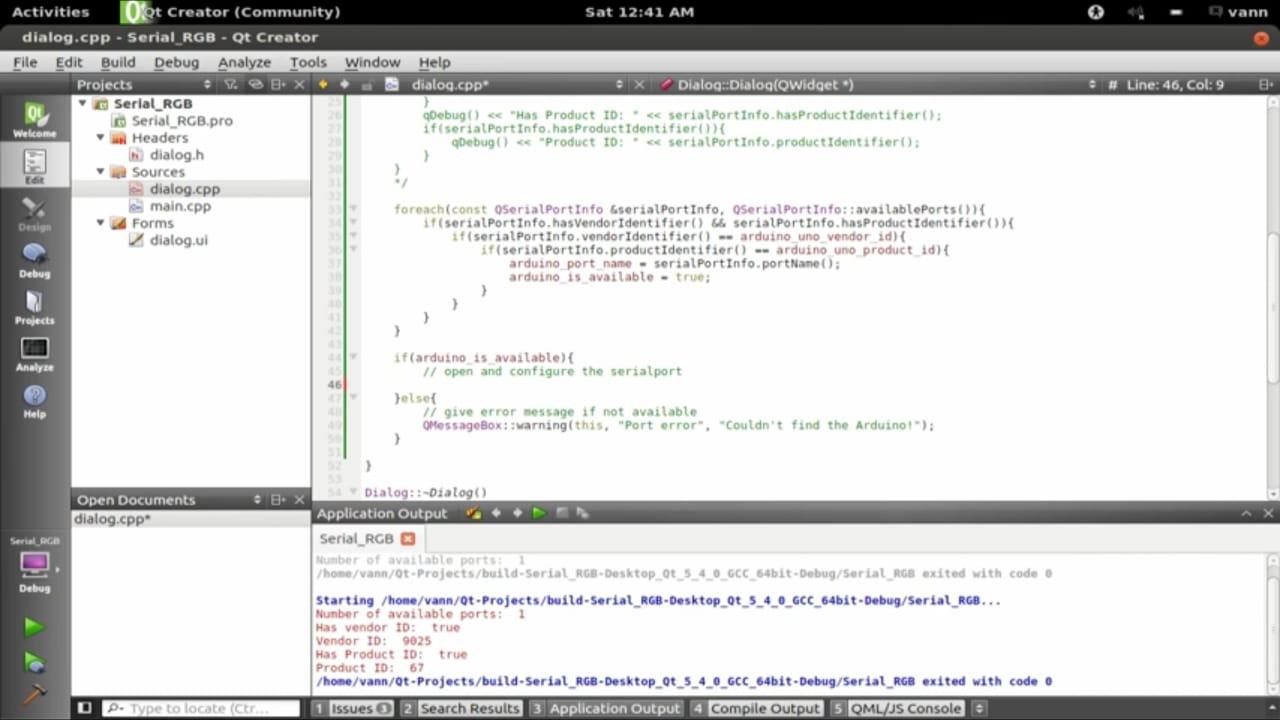
**STEP 4**



**STEP 5**

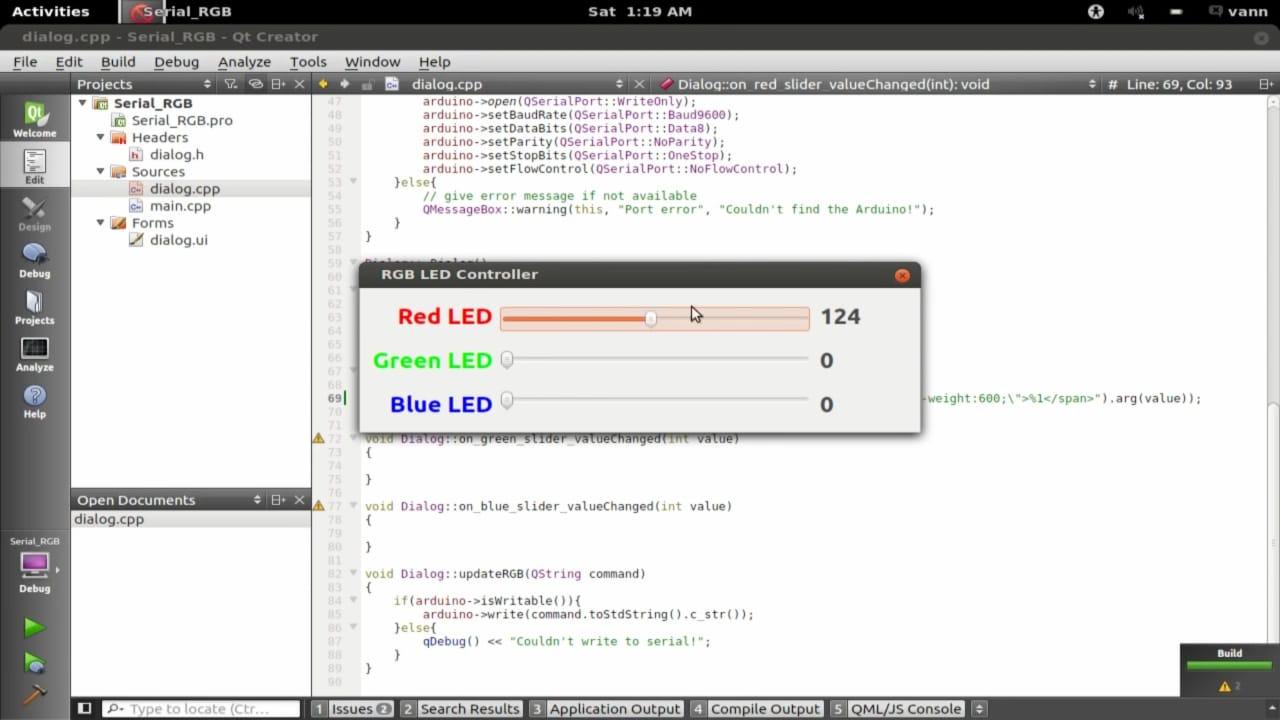


**STEP 6**



**STEP 7**

**FINAL OUTPUT**



**5.4 QT -TEMPERATURE SENSOR**

**MAIN FILE**

#include <DallasTemperature.h>

#include <OneWire.h>

OneWireow(9);

DallasTemperaturetemp\_sensor(&ow);

void setup()

{

temp\_sensor.begin();

Serial.begin(9600);

}

void loop()

{

temp\_sensor.requestTemperatures();

double temp = temp\_sensor.getTempCByIndex(0);

Serial.print(temp);

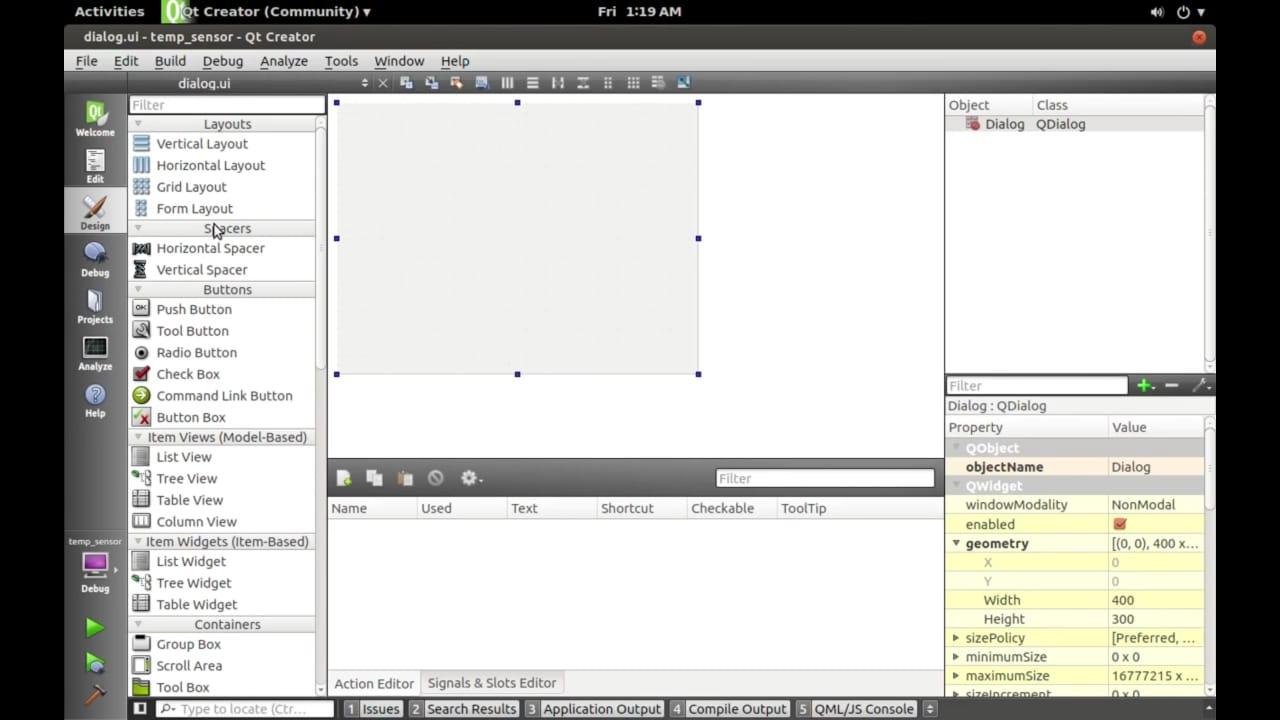
Serial.print(",");

Serial.flush();

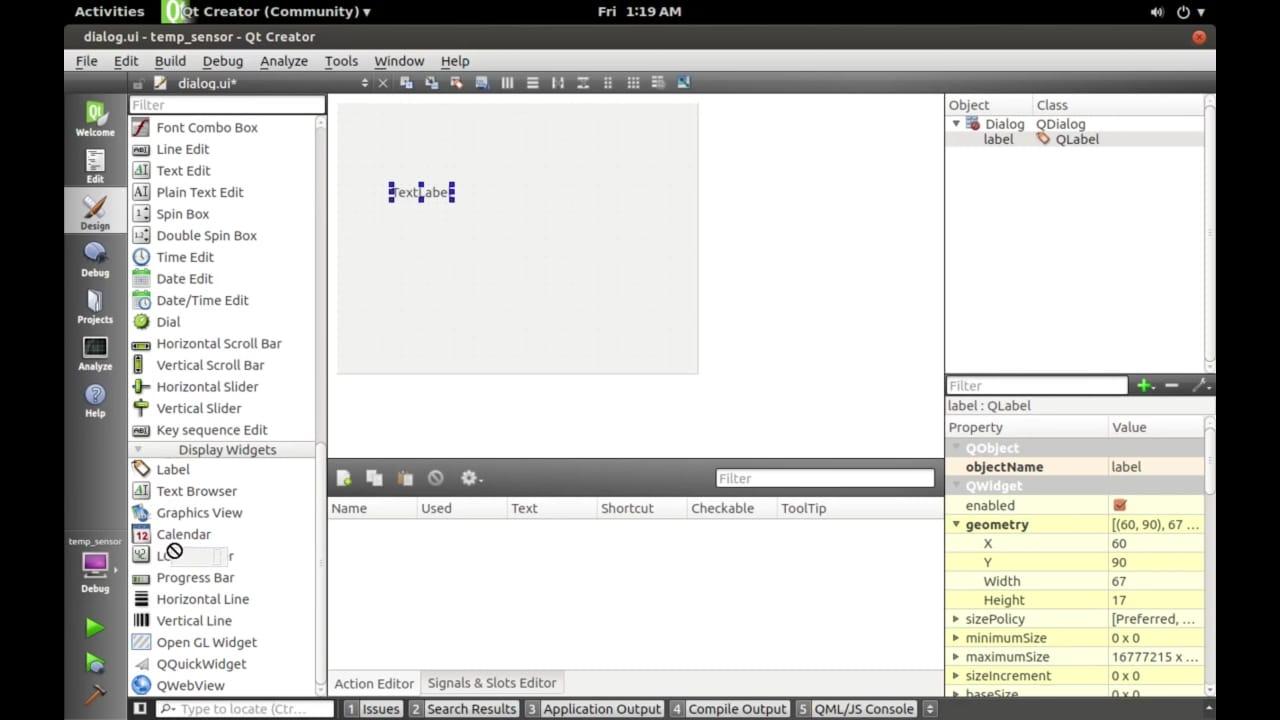
delay(1000);

}

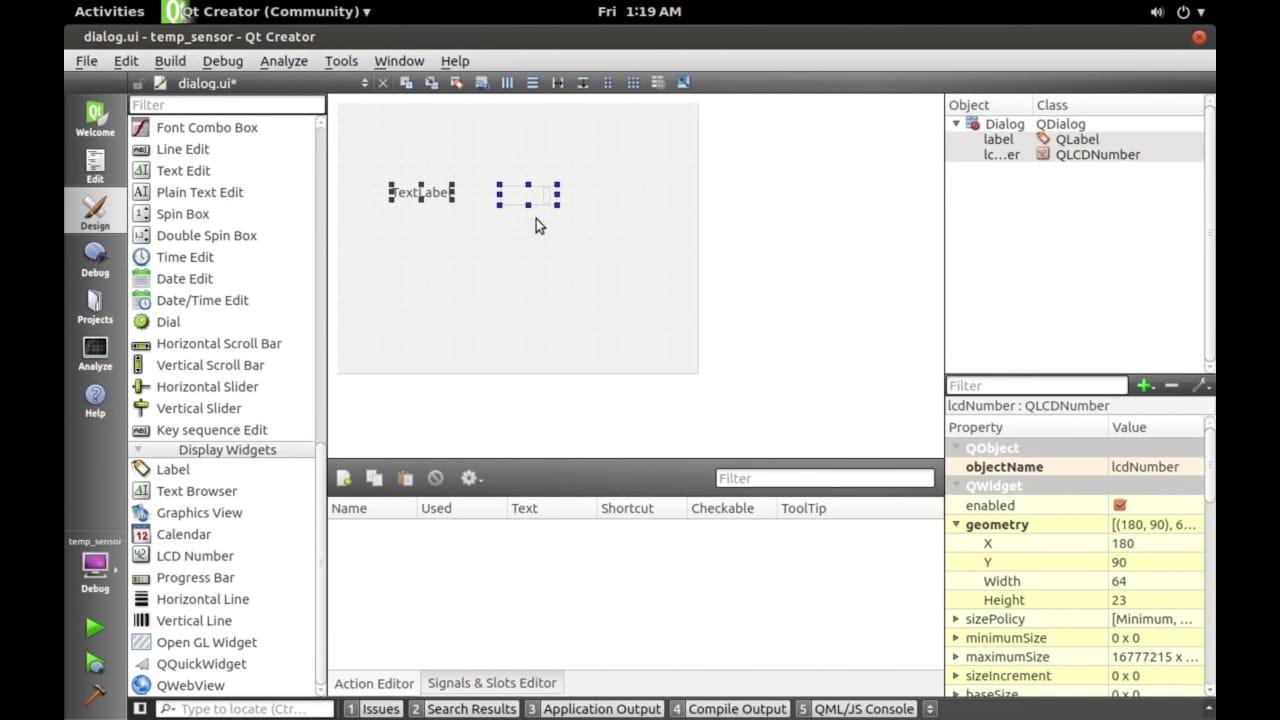
**STEP 1**



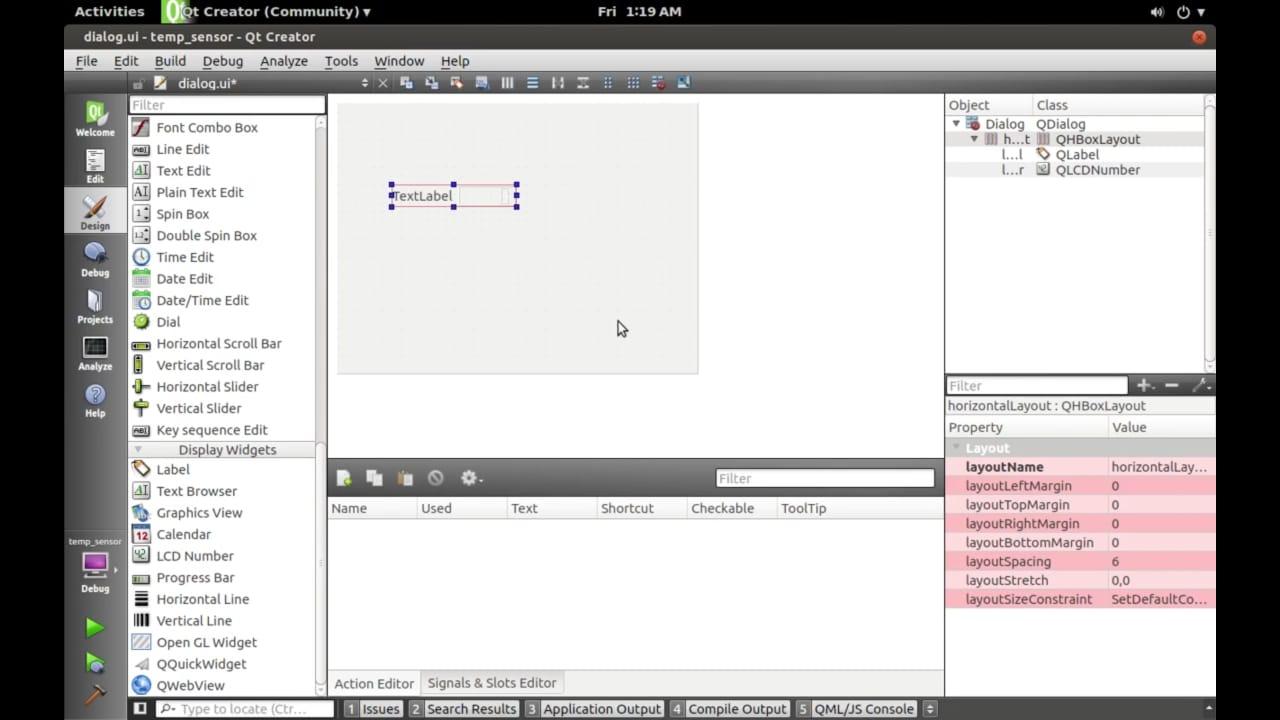
**STEP 2**



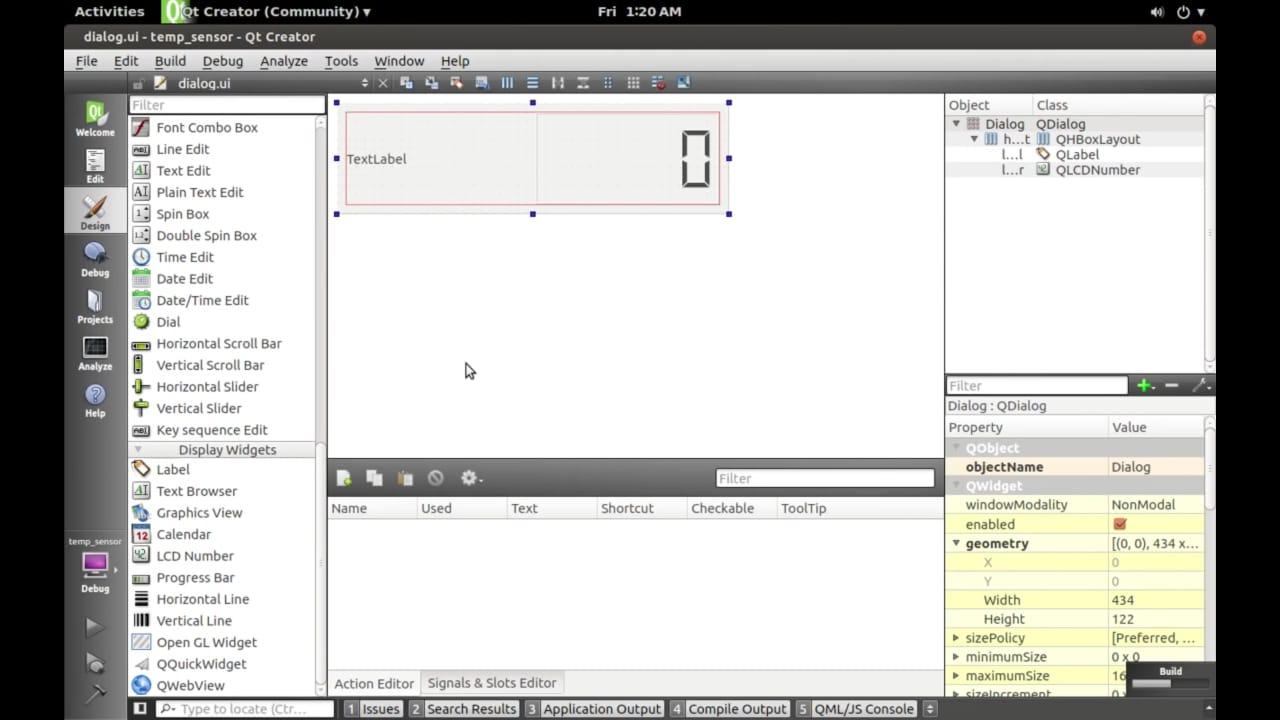
**STEP 3**



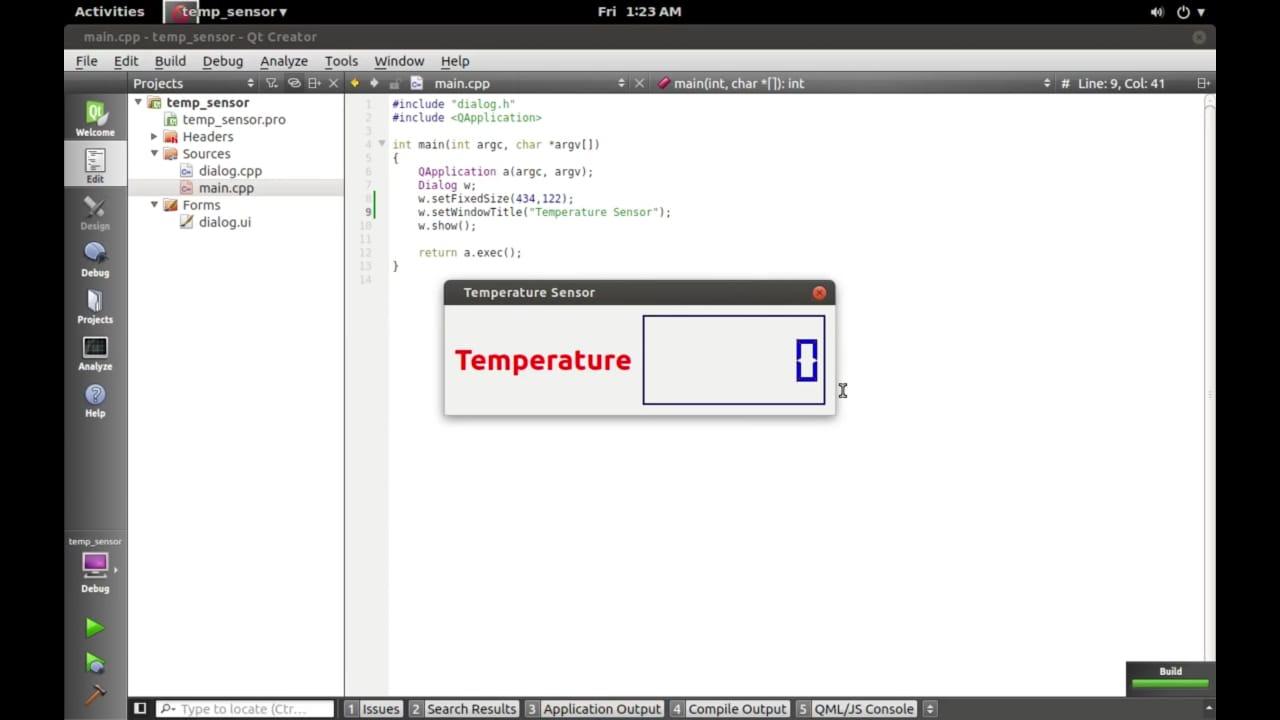
**STEP 4**

****

**STEP 5**

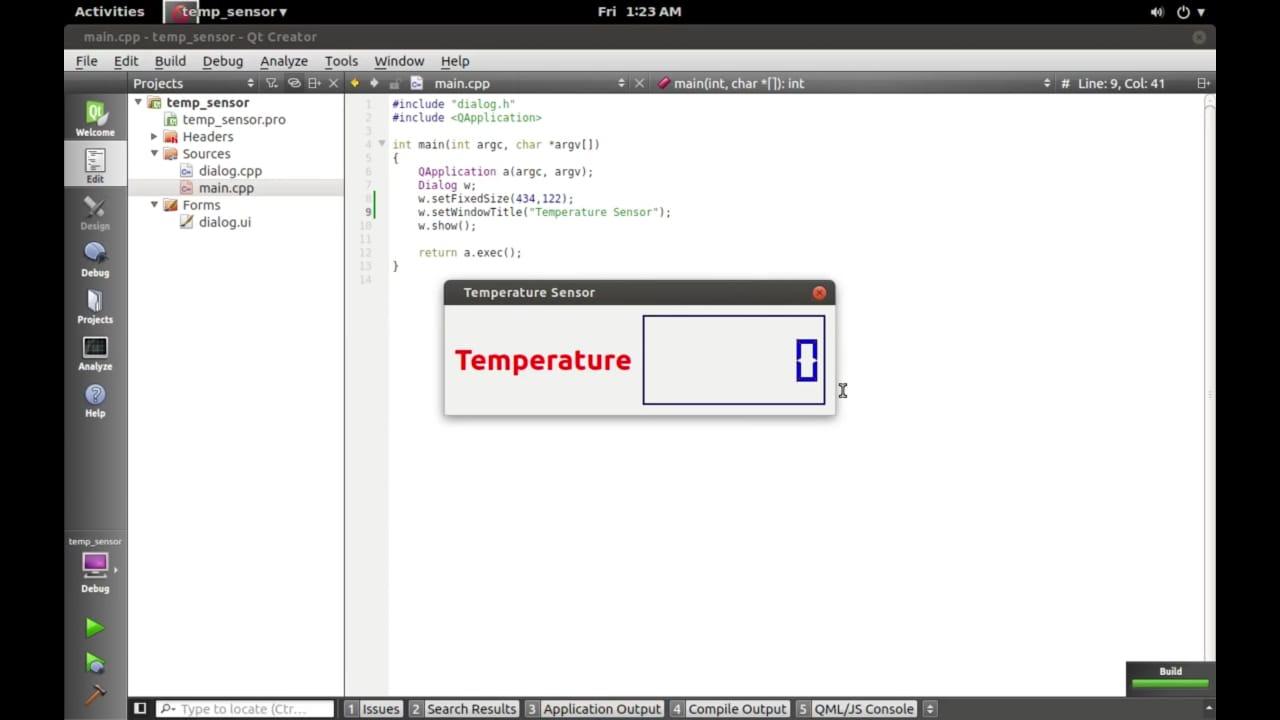


**STEP 6**



**STEP 7**

**FINAL OUTPUT**



**5.5 QT NOTEPAD APPLICATION USING PYTHON**

importtkinter as ck

fromtkinter.filedialog import askopenfilename, asksaveanfilename

def open file():

\*\*Open a file for editing.\*\*

filetypes-("Text Files", ".txt"), ("All Files", \*.\*)]

filepathankopenfilename (

if not filepath:

return

txtedit.delete (1.0, tk.END)

with open (filepath, "") as input file:

text input file.read()

txt\_edit.insert (tk. END, text)

window.title (f"PYTHONAPPLICATIONNOTEPAD (filepath)")

def save file():

\*\*Save the current file as a new file.\*\*

filepathasksaveasfilename (

defaultextension="txt",

filetypes-[("Text Files", "\*.txt"), ("All Files", ".")],

if not filepath:

with open (filepath, "u") as output\_file::

text-txt\_edit.get (1.0, tk.END)

output\_file.write (text)

window.title (f"PYTHONAPPLICATIONNOTEPAD (filepath)"}

window - tk.Tk()

window.title("PYTHONAPPLICATIONNOTEPAI)

vindow.rowconfigure (0, minsize-800, weight=1)

window.columnconfigure (1, minsize-800, weight-1)

txt\_edittk.Text (window)

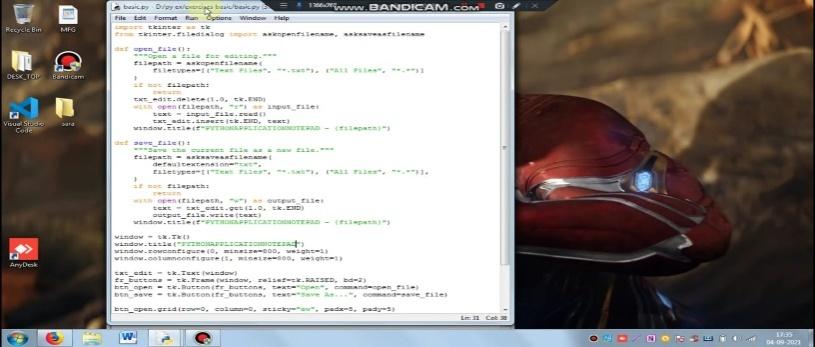
fr buttons tk. Frame (window, relief-tk.RAISED, bd-2)

btn\_open = tk.Button (fr\_buttons, text="Open", command-open\_file)

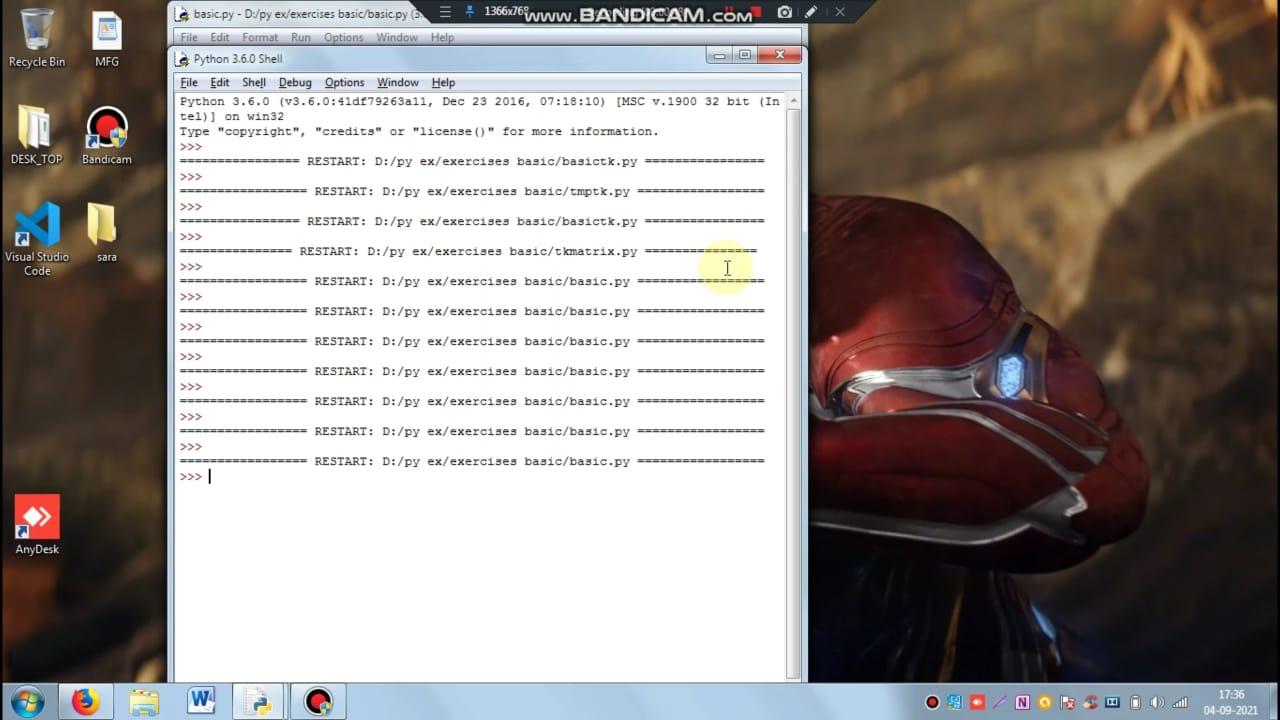
btn\_savetk.Button (fr\_buttons, text-"Save As...", command save\_file)

btn\_open.grid (zow-0, column-0, sticky", padx-5, pady-5)

**STEP 1**

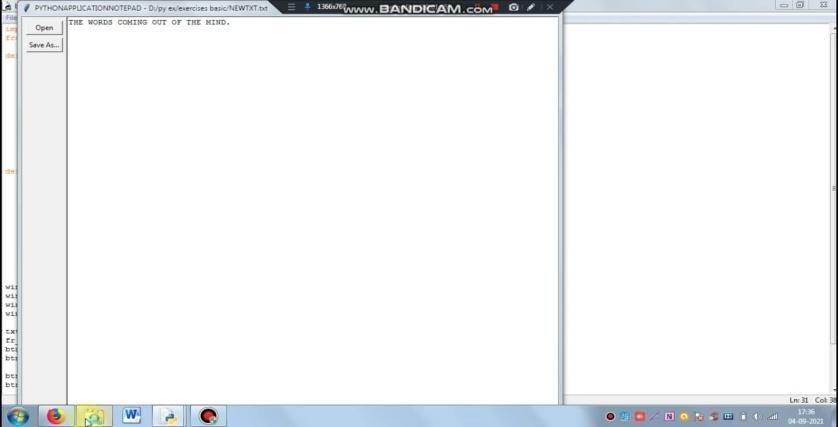


**STEP 2**

****

**STEP 3**

**FINAL OUTPUT**



**6. CONCLUSION**

During the intern period, we learnt about QTcreator IDE and C ++ and created GUI to interact with hardware applications. We also learnt about the application of electronic instrumentation and software in unmanned systems for ground vehicles used for surveillance. It was a great opportunity for us to learn innovative techniques for creating interface using GUI and compatibility for unmanned system.

**7. REFERENCES**

**YouTube**

* <https://www.youtube.com/watch?v=AX-HhBXBzGg>
* <https://www.youtube.com/watch?v=iAbJy4weFK8>
* <https://www.youtube.com/watch?v=aOCLUpk6QAU>
* <https://www.youtube.com/watch?v=9oQZPqi47Y8>
* http://en.wikipedia.org/wiki/Unmanned ground vehicle.
* http://www.armyguide.com/eng/product1795.html,2002.

1. Custom Video Streaming Player using LibVLC and Qt
2. REQUIREMENTS FOR THE CONTROL OF GUNNERS CONTROL AND DISPLAY INTERFERENCE CHARACTERISTICS OF SUBSYSTEMS AND EQUIPMENT