

Page 1 | 3

TECHNICAL REPORT FOR AQUAPHOTN’S

MEGATRAINING PROJECT

# 1-Introduction

# This Mega Project is a simulation of the main activities needed to be ready for a Robot The main objective is to create a flexible and effective remotely operated vehicle that can be operated via a comprehensive GUI interface and switch between manual and autonomous modes The project brings together hardware design, firmware programming, and advanced software techniques, particularly in image processing.

# 1. Hardware and Firmware

# In the hardware aspect, you will design and fabricate a PCB that forms the core of the ROV’s control system. This PCB will house a bare microcontroller circuit, motor drivers for controlling two DC motors, and feedback sensors to monitor the car’s voltage and current. The vehicle will be powered by a battery, making it fully mobile.

# In firmware, the car will operate in two modes:

# • Manual Mode: This mode enables the user to move the car in any direction using a GUI and wireless control. • Autonomous Mode: Using PID control to guarantee it travels parallel to the wall, the vehicle will keep a certain distance from it.

# 2. Graphical User Interface (GUI)

# The GUI serves as the ROV's command center, giving users access to real-time video feeds, sensor readings, and vehicle control. With several windows for distinct tasks, the interface will be flexible, modular, and easy to use. The graphical user interface will integrate:

# • Car Control: Move the vehicle in either manual or automated mode.

# • Live Camera Feed: See and take pictures with the vehicle's cameras.

# •Sensor Readings: Display real-time voltage, current, and ultrasonic sensor data.

# •Video Stitching and Stereo Vision: Advanced image processing tasks to merge video feeds and extract 3D information from 2D images.

# 3. Computer Vision Tasks

# These tasks involve using image processing techniques to achieve specific goals:

# Video Stitching: Combine two video streams (e.g., from left and right cameras) into a single, larger video, useful for enhanced visual coverage.

# Stereo Vision: Extract 3D information from the environment, helping in navigation and object measurement underwater.

# 2-Discussion

# Hardware

# ❖ Software tasks’ Algorithms:

# Arduino Code Algorithm

# Initialization:

# Initialize the microcontroller, sensors, motors, and user interface components.

# Configure the variables required for PID control, ultrasonic sensor reading, and motor control.

# Inputs from user:

# Ask the user to choose the car's speed (low, medium, or high).

# Ask the user to select the mode of operation (manual or autonomous).

# In manual mode

# Give the driver the ability to stop, turn the car left or right move forward or backward. Execute the corresponding motor control commands based on user input. Press '0' to allow the user to go out of manual mode.

# In autonomous mode

# The code measures the separations between obstacles in the autonomous mode by analyzing data from ultrasonic sensors. It computes errors and applies a PID controller to keep the car moving at a safe distance from obstacles.

# PID Control

# The PID function determines the errors for the left and right ultrasonic, adjusts the car's movements based on the errors, and makes corrections by turning left or right as needed to keep moving in a path parallel to walls.

# Ultrasonic Sensor Reading

# To calculate the distance between the car and the walls, the read\_ultrasonic function reads data from the ultrasonic sensors located at the sides of the car.

# RGB LED

# LED has 3 colors depending on the speed of the motor

# low: red color

# medium: blue color

# high: green color

# Flow Chart

# 

# ❖ GUI Features:

# Our GUI has the following features:

# Manual Mode

# In the manual mode, the GUI allows users to control the car's movements directly. When this mode is selected, the car's color changes to indicate that it is in manual mode. A new menu appears with directional buttons, enabling the user to choose the direction the car should move. This mode gives the user total control over the car's movements, making it perfect for hands-on control.

# Autonomous Mode

# In Auto Mode, the car operates autonomously based on user inputs. Selecting this mode changes the car's color to signify the switch to autonomous control. A new window opens where the user can input the desired distance the car should travel.

# Speed

# The user can choose from three speed levels: Low, Medium, and High. The speed indicator reflects the selected speed, adjusting the color to match the selected level.

# Main Window Features

# A screenshot of a video game Description automatically generated

# A screenshot of a video game Description automatically generated

# A screenshot of a video game Description automatically generated

# A screenshot of a computer game Description automatically generated

# Sub Windows Features

### **Integration in the GUI**

1. **Button Actions**:
   * **Manual Mode and Auto Mode**:
     + Buttons for switching between Manual and Auto modes **(manual\_pushButton and auto\_pushButton**) change the appearance of the car icon and open the respective dialogs. The **change\_car\_icon\_color** method updates the car icon when Auto mode is selected, and **return\_car\_icon\_color** reverts the icon color when Manual mode is selected.
   * **Speed Selection**:
     + Buttons for speed selection (**low\_pushButton, medium\_pushButton, high\_pushButton**) are connected to the **changeSpeed** method. Each button triggers a signal that sends the corresponding speed command to the Arduino.
2. **Dialogs**:
   * **Manual Control Dialog**:
     + Opens a dialog window with directional buttons (**up\_pushButton,** **down\_pushButton, left\_pushButton, right\_pushButton**). Each button is connected to the **sendCommand** method, which sends the respective command to the Arduino via the serial thread.
   * **Auto Mode Dialog**:
     + Opens a dialog window where users can input a distance and select a speed level. The **sendSafeDistance** method reads the input, validates it, and sends it to the Arduino. The speed level can be adjusted using the UI, and corresponding speed commands are sent to the Arduino.
3. **Serial Communication**:
   * **SerialThread**:
     + Manages serial communication in a separate thread to avoid blocking the GUI. It reads data from the Arduino and emits signals (**data\_received**) that update sensor values in the UI.
   * **Sending Commands**:
     + Commands for speed and mode are sent to the Arduino using the **sendCommand** method. This method is called from various UI actions, including speed button clicks and dialog button actions.
4. **Sensor Updates**:
   * **updateSensorValues**:
     + This method updates the UI with data received from the Arduino, such as current and voltage readings. It processes incoming serial data and updates the corresponding QLabel widgets in the main window.
5. **Event Handling**:
   * **close event**:
     + Ensures the serial thread is stopped and the serial port is closed when the application is closed.