**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Hafsa Iqbal **Date:** 26/04/2024

**Semester:** 8th **Group:** GP-1

**Lab Engineer:** Munadi Ahmad Sial

**EE-381 Robotics**

Lab 12: C++ Programming in ROS

Open-ended Lab

**Group Members**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **PLO5 – CLO4** | | **PLO9 –**  **CLO4** |  |
| **Name** | **Reg. No** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Individual and Team Work** | **Total Marks** |
|  |  | **10 Marks** | **5 Marks** | **5 Marks** | **20 Marks** |
| Muhammad Abdullah Sohail | 343642 |  |  |  |  |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |
| Hassan Rizwan | 335753 |  |  |  |  |
| Muhammad Saad | 333414 |  |  |  |  |

**Table of Contents**

[2 C++ Programming in ROS 3](#_Toc166275562)

[2.1 Introduction 3](#_Toc166275563)

[2.2 Objectives 3](#_Toc166275564)

[2.3 Theory 3](#_Toc166275565)

[3 Open-ended Lab 4](#_Toc166275566)

[3.1 Lab Task 4](#_Toc166275567)

# C++ Programming in ROS

## Introduction

This laboratory exercise will focus on designing and simulation a robot simulation that performs a specified task while employing nodes driven by C++ programs.

## Objectives

The following are the main objectives of this lab:

* Write C++ program script files
* Implement C++ nodes in ROS

## Theory

In robotics, the two most popular languages for programming robots are C++ and Python. For prototype development and design concepts, python is used due to its user-friendly nature. For practical implementation, C++ is used as the code is compiled and thus runs much faster on the machine. A major issue with C++ programming is its’ high difficulty. In ROS, there is extensive support for both Python (rclpy) and C++ (rclcpp) as well as limited support for other programming languages.

The terminal commands are given as:

**cd <directory>**  change directory

**cd..**  go back to previous directory

**pwd**  print the current directory

**ls** list the contents of the current directory

**python <script.py>** execute python script

A brief summary of the terminal commands needed for working in ROS are provided below:

**colcon build**

build the workspace whenever a node is created or modified

**. install/setup.bash**

makes the terminal “aware” of the workspace (notice the dot and the space)

**ros2 pkg create --build-type ament\_python <package>**

create a new package (must be done in src directory)

**ros2 run <package> <node>**

execute a node

In the lab, you will download a number of image files which you must use in the tasks. Ensure that the terminal shows the names of the group members in the screenshots.

# Open-ended Lab

## Lab Task

Design and implement a ROS implementation that solves a specified problem with the constraint of using C++ programming in the nodes.

Note that the problem to be solved, as well as the solution to be designed, are both open-ended in nature. Students are expected to come up with their own creative ideas and implementations for the lab.

Specify the complete details of the problem and how it is to be solved.

### PROBLEM SPECIFICATION DETAILS START HERE ###

**Problem:** The objective is to design and implement a line-following robot using the Robot Operating System 2 (ROS2) framework and C++. The robot will be equipped with a camera sensor to capture images of its environment. Its primary task is to autonomously navigate a predefined track, characterized by a black line on a white surface.

**Constraints:** Implement the core logic in C++ within ROS nodes.

The proposed solution will involve the following key steps:

* Image Acquisition and Preprocessing
* Line Detection
* Line Feature Extraction and Center Calculation
* Control Algorithm and Robot Guidance
* System Integration and Testing

### PROBLEM SPECIFICATION DETAILS END HERE ###

Provide all relevant codes, screenshots/videos, and explanations of your work.

### CODES START HERE ###

**Linefollow.cpp**

#include <chrono>

#include <functional>

#include <memory>

#include <string>

#include "cv\_bridge/cv\_bridge.h"

#include "geometry\_msgs/msg/twist.hpp"

#include "opencv2/opencv.hpp"

#include "rclcpp/rclcpp.hpp"

#include "sensor\_msgs/msg/image.hpp"

*class* LineFollower : *public* rclcpp::Node {

*public:*

    LineFollower() : Node("task") {

        subscription\_ = *this*->create\_subscription<sensor\_msgs::msg::Image>(

            "camera/image\_raw", 10, std::bind(&LineFollower::callback, *this*, std::placeholders::\_1));

        publisher\_ = *this*->create\_publisher<geometry\_msgs::msg::Twist>("cmd\_vel", 10);

        move = geometry\_msgs::msg::Twist();

        br = std::make\_shared<cv\_bridge::CvImage>();

    }

*private:*

*void* callback(*const* sensor\_msgs::msg::Image::SharedPtr *msg*) {

        RCLCPP\_INFO(*this*->get\_logger(), "Receiving video frame");

        cv\_bridge::CvImagePtr cv\_ptr;

        cv\_ptr = cv\_bridge::toCvCopy(msg, sensor\_msgs::image\_encodings::BGR8);

        cv::Mat img = cv\_ptr->image;

        cv::resize(img, img, cv::Size(), 0.2, 0.2, cv::INTER\_CUBIC);

        cv::Mat img\_hsv;

        cv::cvtColor(img, img\_hsv, cv::COLOR\_BGR2HSV);

        geometry\_msgs::msg::Twist twist;

*float* linear\_x = 0.3;

*int* h\_min = 55;

*int* h\_max = 179;

*int* s\_min = 0;

*int* s\_max = 255;

*int* v\_min = 81;

*int* v\_max = 255;

*// making the mask*

        cv::Mat mask;

        cv::inRange(img\_hsv, cv::Scalar(h\_min, s\_min, v\_min),

cv::Scalar(h\_max, s\_max, v\_max), mask);

        cv::Mat result;

        cv::bitwise\_and(img, img, result, mask = mask);

*// region of interest (keep the lower 1/5 of the image)*

*int* height = mask.rows;

*int* width = mask.cols;

        cv::Mat roi = mask.clone();

        roi.rowRange(0, 4 \* height / 5) = cv::Scalar(0);

*// roi = mask.rowRange(4 \* height / 5, height);*

*// center of the region of interest*

        cv::Moments M = cv::moments(roi);

*int* cX, cY;

        if (M.m00 != 0) {

            cX = M.m10 / M.m00;

            cY = M.m01 / M.m00;

        } else {

            cX = 0;

            cY = 0;

        }

        cv::Point center\_roi(width / 2, (4 \* height / 5) / 2);

*float* distance\_center = std::sqrt(std::pow(cX - center\_roi.x, 2)

+ std::pow(cY - center\_roi.y, 2));

*// move the robot based on*

*// the distance from the center of the region of interest*

*// follow the line by turning left or right*

        if (distance\_center < 120) {

            twist.linear.x = linear\_x;

            twist.angular.z = 0;

        } else {

            twist.linear.x = 0;

            twist.angular.z = cX < center\_roi.x ? 0.3 : -0.3;

        }

        publisher\_->publish(twist);

    }

    rclcpp::Subscription<sensor\_msgs::msg::Image>::SharedPtr subscription\_;

    rclcpp::Publisher<geometry\_msgs::msg::Twist>::SharedPtr publisher\_;

    geometry\_msgs::msg::Twist move;

    cv\_bridge::CvImagePtr br;

};

*int* main(*int* *argc*, *char\** *argv*[]) {

    rclcpp::init(*argc*, *argv*);

    rclcpp::spin(std::make\_shared<LineFollower>());

    rclcpp::shutdown();

    return 0;

}

**package.xml**

<?xml *version*="1.0"?>

<?xml-model *href*="http://download.ros.org/schema/package\_format3.xsd" *schematypens*="http://www.w3.org/2001/XMLSchema"?>

<package *format*="3">

  <name>cpp\_linefollow</name>

  <version>0.0.0</version>

  <description>TODO: Package description</description>

  <maintainer *email*="thisismumer@gmail.com">root</maintainer>

  <license>TODO: License declaration</license>

  <buildtool\_depend>ament\_cmake</buildtool\_depend>

  <depend>rclcpp</depend>

  <depend>std\_msgs</depend>

  <depend>sensor\_msgs</depend>

  <depend>cv\_bridge</depend>

  <depend>image\_transport</depend>

  <depend>OpenCV</depend>

  <test\_depend>ament\_lint\_auto</test\_depend>

  <test\_depend>ament\_lint\_common</test\_depend>

  <export>

    <build\_type>ament\_cmake</build\_type>

  </export>

</package>

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.8)

project(cpp\_linefollow)

if(CMAKE\_COMPILER\_IS\_GNUCXX OR CMAKE\_CXX\_COMPILER\_ID MATCHES "Clang")

  add\_compile\_options(-Wall -Wextra -Wpedantic)

endif()

*# find dependencies*

find\_package(ament\_cmake REQUIRED)

find\_package(rclcpp REQUIRED)

find\_package(std\_msgs REQUIRED)

find\_package(sensor\_msgs REQUIRED)

find\_package(cv\_bridge REQUIRED)

find\_package(image\_transport REQUIRED)

find\_package(OpenCV REQUIRED)

if(BUILD\_TESTING)

  find\_package(ament\_lint\_auto REQUIRED)

*# the following line skips the linter which checks for copyrights*

*# comment the line when a copyright and license is added to all source files*

  set(ament\_cmake\_copyright\_FOUND TRUE)

*# the following line skips cpplint (only works in a git repo)*

*# comment the line when this package is in a git repo and when*

*# a copyright and license is added to all source files*

  set(ament\_cmake\_cpplint\_FOUND TRUE)

  ament\_lint\_auto\_find\_test\_dependencies()

endif()

add\_executable(cpp\_linefollow src/linefollow.cpp)

ament\_target\_dependencies(

  cpp\_linefollow

  "rclcpp"

  "image\_transport"

  "cv\_bridge"

  "sensor\_msgs"

  "geometry\_msgs"

  "std\_msgs"

  "OpenCV"

)

install(TARGETS

  cpp\_linefollow

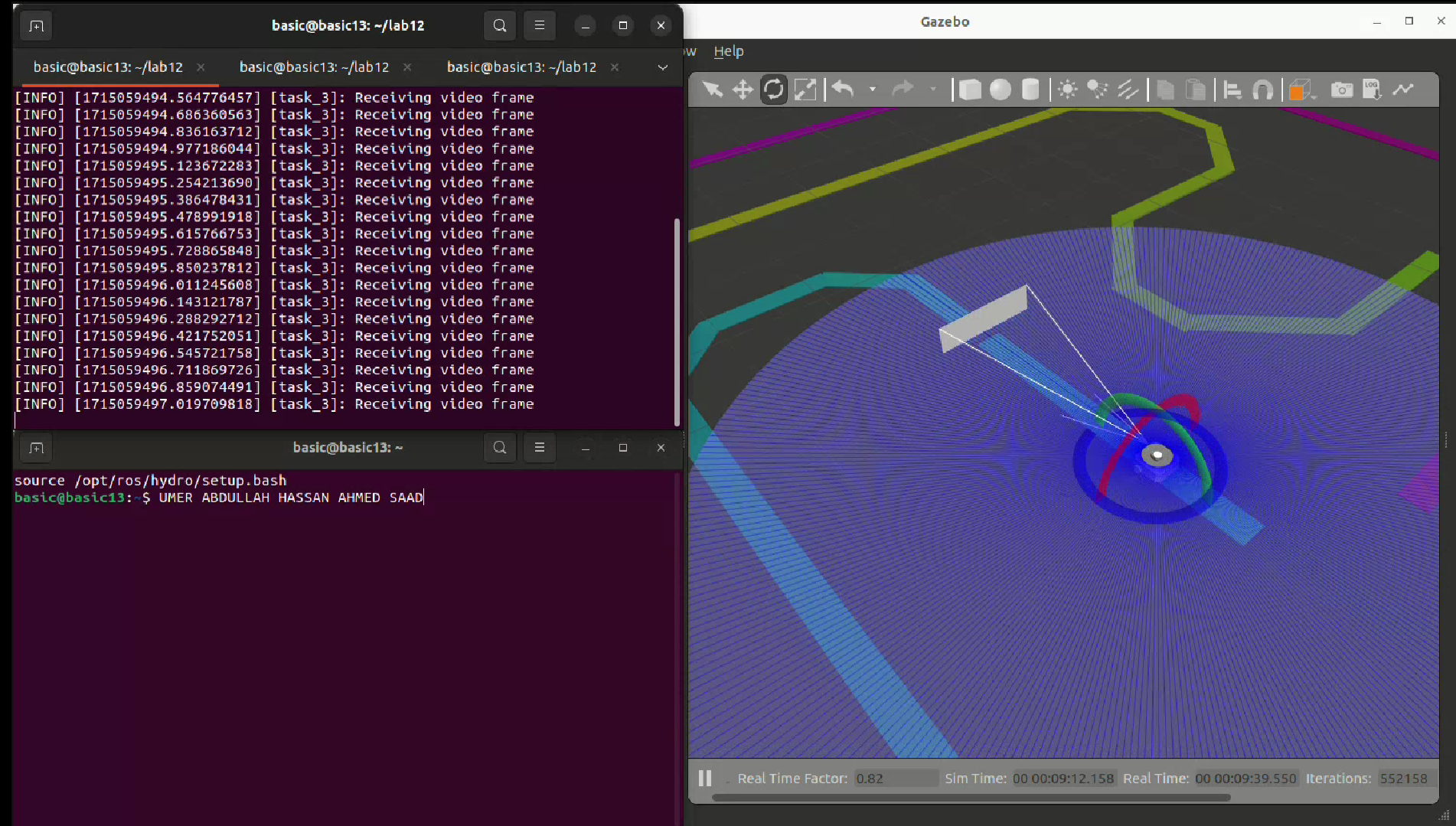
  DESTINATION lib/${PROJECT\_NAME}

)

ament\_package()

### CODES END HERE ###

### SCREENSHOTS START HERE ###



### SCREENSHOTS END HERE ###

### EXPLANATION-CONCLUSION START HERE ###

The provided C++ code serves as the brain of our line-following robot.

It begins by subscribing to the camera feed and setting up a way to send commands to control the robot's movement. When a new image arrives, the code analyzes it by first converting it to a specific color format that makes it easier to distinguish the black line from the white background. A mask is then applied to the image, isolating the black line pixels. As we are primarily interested in the line directly in front of the robot, the code focuses on a specific region in the lower part of the image. By calculating the center point of the detected line within this region, we can determine how much the robot needs to turn to stay on track.

Based on the position of the line relative to the center of the robot's view, the code generates commands to adjust the robot's speed and direction. If the line is centered, the robot moves forward. If the line deviates to one side, the robot turns accordingly to realign itself with the path. These commands are then sent to the robot, allowing it to navigate the line autonomously.

While the current code effectively demonstrates a basic line-following behavior, there are many potential enhancements. For instance, we could implement more advanced algorithms to improve the accuracy and robustness of line detection, especially under varying lighting conditions or with different line appearances. Additionally, incorporating feedback mechanisms like a PID controller would allow the robot to dynamically adjust its speed and turning, leading to smoother and more precise navigation. For a truly autonomous experience, integrating additional sensors like LiDAR could enable the robot to detect and avoid obstacles on its path.

### EXPLANATION-CONCLUSION END HERE ###