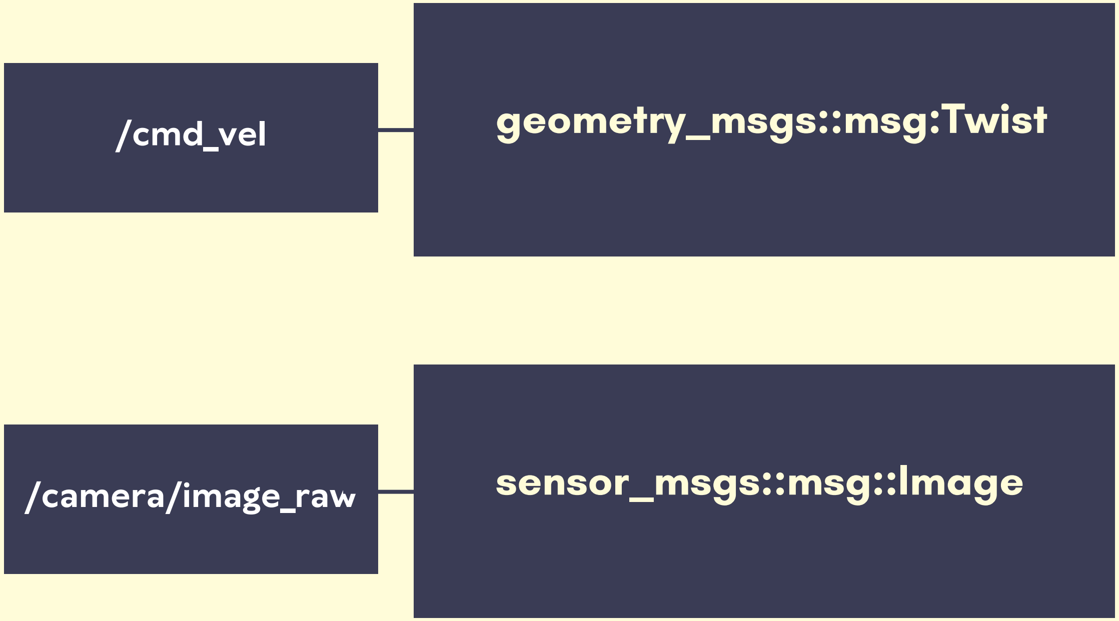
FROM POV CODER

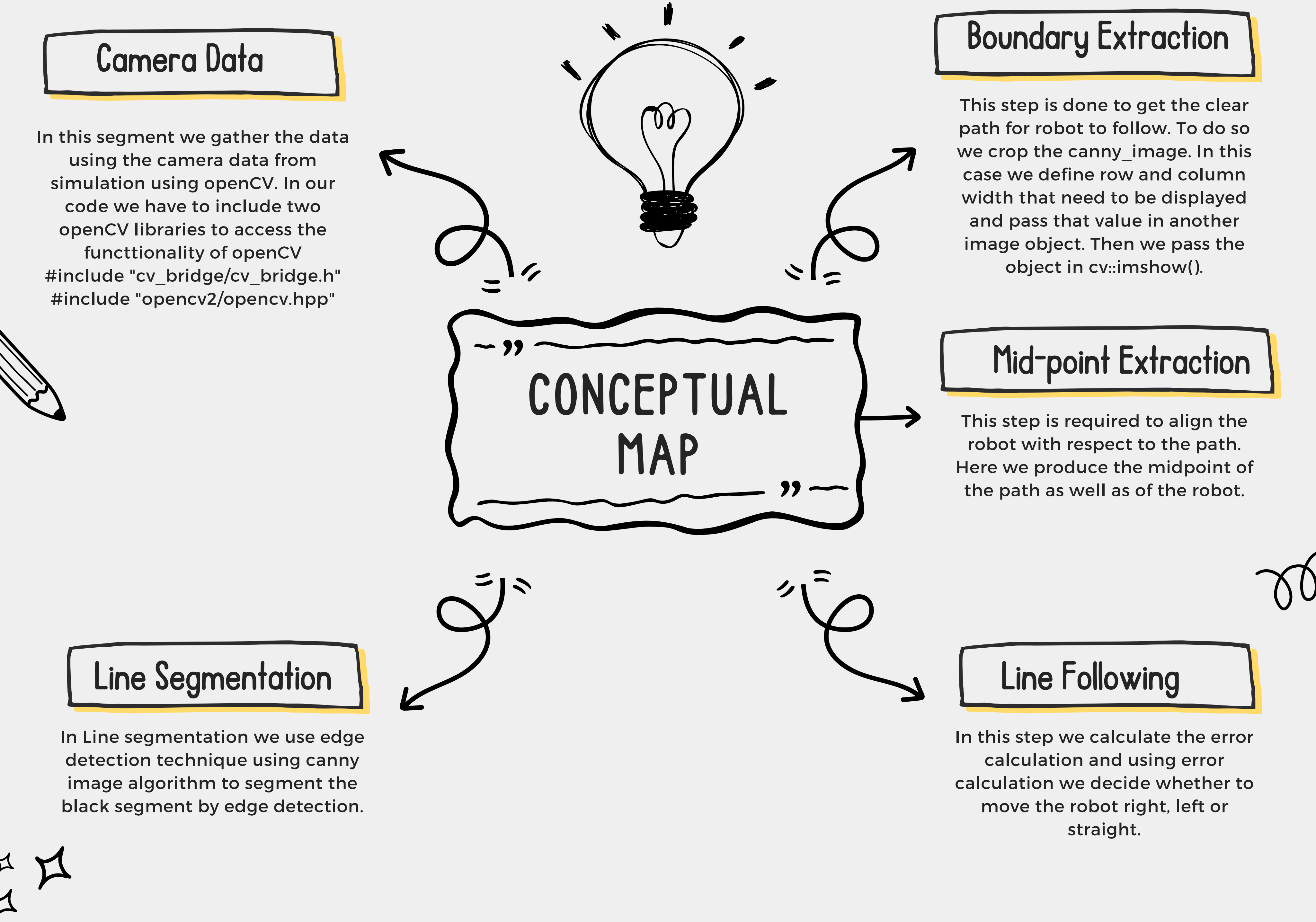
Name of Node we created:
“Line follower”
Publisher topic:
“/cmd_vel”
Subscriber topic:
“/camera/image_raw”

Name of Node we are using for simulation:
“turtlebot3_gazebo”
Publisher topic:
“/camera/image_raw”
Subscriber topic:
“/cmd_vel”



/cmd_vel — geometry_msgs::msg:Twist

/camera/image_raw — sensor_msgs::msg::Image



CONCEPTUAL MAP

Camera Data

In this segment we gather the data using the camera data from simulation using openCV. In our code we have to include two openCV libraries to access the functtionality of openCV

```
#include "cv_bridge/cv_bridge.h"
#include "opencv2/opencv.hpp"
```

Line Segmentation

In Line segmentation we use edge detection technique using canny image algorithm to segment the black segment by edge detection.

Boundary Extraction

This step is done to get the clear path for robot to follow. To do so we crop the canny_image. In this case we define row and column width that need to be displayed and pass that value in another image object. Then we pass the object in cv::imshow().

Mid-point Extraction

This step is required to align the robot with respect to the path. Here we produce the midpoint of the path as well as of the robot.

Line Following

In this step we calculate the error calculation and using error calculation we decide whether to move the robot right, left or straight.

CAMERA DATA

- step 1: Create a pointer to access the camera data
    - cv:bridge::CVImagePtr cv_ptr
    - cvr_ptr = cv_bridge :: toCvCopy(camera_msg, “bgr8”)
        - here camera_msg is sharedptr with message type
        - sensor_msgs::msg::Image
        - “bgr8” format of the image
- step 2: Change the image to grey color
    - cv::Mat gray_image
        - creating a object name gray_image
    - cv::cvtColor(cv_ptr->image, gray_image,cv::ColorBGR2GRAY)
        - in this syntax we are saying from pointer cv_ptr change image to Gray using ColorBGR2GRAY and save it to object name gray_image
- step 3: Displaying the image
    - cv::imshow(“Image”,gray_image)
        - this syntax indicate show the image named gray_image

LINE SEGMENTATION

- step 1: Calculation mid_point and robot_mid_point
  - for mid_point
  - int mid_area = edge[1] - egde[0]
    - here egde[1] and edge[0] repersent seperate edge on the image if there is any
  - now to find mid_point we add half of mid_area with edge[0]
  - int mid_point = edge[0] + mid_area/2
  - int robot_mid_point = 640/2
    - we are using the total frame length which is 640 and dividing by 2.
- step 2: creating circle to represent two dot on the image
  - cv::circle(roi, cv::Point(mid_point, 160),2, cv::Scalar(255,255,255)-1)
    - this means we want to create a cirlce with region of interest (roi), with x value supplied by x and y value as 160, the color of the point is white as all the value of RGB is 255. The -1 represent the outline of the circle
  - now we do same for robot mid point
    - cv::circle(roi, cv::Point(robot_mid_point, 160),5, cv::Scalar(255,255,255)-1)

BOUNDARY EXTRACTION

- step 1: Define the width of row and column
  - int row = 150, column =0
  - cv::Mat img = canny_image(cv::Range(row, row+240), cv::Range(column,column+640));
    - in this syntax we are taking canny_image and cropping the image where the row indicate the vertical height from 150 to 390, and in horizontal sectio we are keeping the whole width of the canny_image.
- step 2: displaying the image
  - cv::imshow("Image", img);

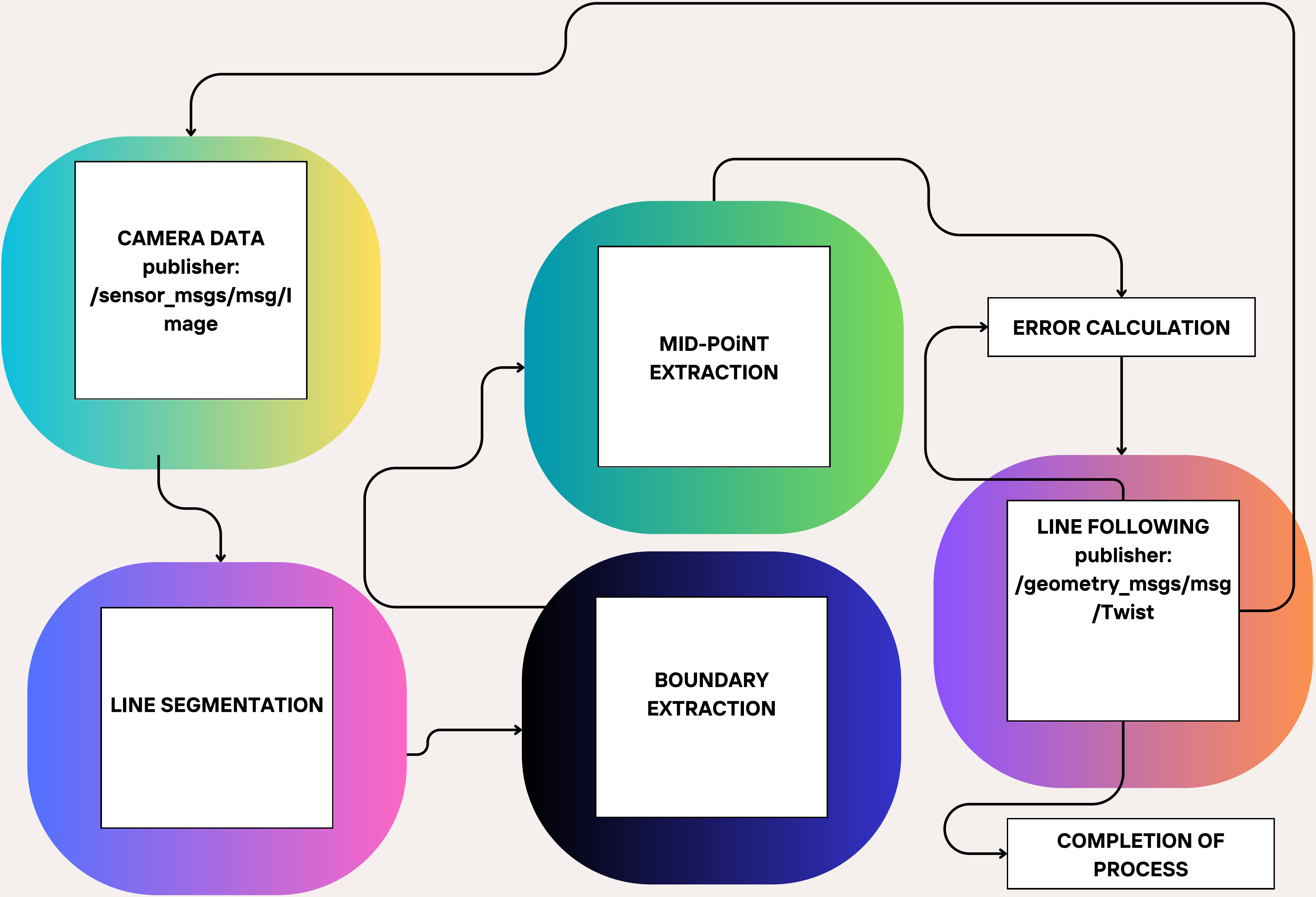
MID-POINT EXTRACTION

- step 1: Create a pointer to access the camera data
  - cv:bridge::CVImagePtr cv_ptr
  - cvr_ptr = cv_bridge :: toCvCopy(camera_msg, “bgr8”)
    - here camera_msg is sharedptr with message type
    - sensor_msgs::msg::Image
    - “bgr8” format of the image
- step 2: Change the image to grey color
  - cv::Mat gray_image
    - creating a object name gray_image
  - cv::cvtColor(cv_ptr->image, gray_image,cv::ColorBGR2GRAY)
    - in this syntax we are saying from pointer cv_ptr change image to Gray using ColorBGR2GRAY and save it to object name gray_image
- step 3: Displaying the image
  - cv::imshow(“Image”,gray_image)
    - this syntax indicate show the image named gray_image

LINE FOLLOWING

- step 1: Error Calculation
  - double error = robot_mid_point - mid_point
- step 2: Using the error we will steer the robot
  - if (error >0){
    - RCLCPP_INFO(this -> get_logger(),”TURN RIGHT”)
    - velocity_msg.angular.z = -0.8;}
  - elseif (error <0){
    - RCLCPP_INFO(this -> get_logger(),”TURN LEFT”)
    - velocity_msg.angular.z = 0.8;}
  - else{
    - RCLCPP_INFO(get_logger(), “GO STRAIGHT”)}
    - velocity_msg.linear.x= 1.0;}
      - in this case we are using if else statement to steer the robot
      - steering of robot is done using velocity_msg
      - here the velocity_msg is sharedptr for “geometry_msgs::msg::Twist”
      - we define the publisher at the beginning of the private function
      - publisher_->publish(velocity_msg)
      - velocity_msg.angular.z is used for rotational velocity
      - velocity_msg.linear.x is used for linear velocity.

Process diagram:



CAMERA DATA publisher: /sensor_msgs/msg/Image

LINE SEGMENTATION

BOUNDARY EXTRACTION

MID-POiNT EXTRACTION

ERROR CALCULATION

LINE FOLLOWING publisher: /geometry_msgs/msg/Twist

COMPLETION OF PROCESS