Contents

1 Introduction

Autonomous navigation is an ever growing domain of research and practice in today's technological frontier, ranging from applications in autonomous driving, mobile robotics and agricultural machinery to name a few. With the advent of performance computing, and cross-industry progress in sensors, camera technology and development frameworks, autonomous navigation is soon to secure its place in humanity's day to day livelihood [1].

Autonomous mobile robots are composed of a supervisory controller, speed and steering controller, along with obstacle avoidance and emergency controls, with computer vision system at the core, to capture the environment and adapt the robot to it [2]. Such information can be of diverse nature such as lane information for autonomous driving, localization and mapping for mobile robotics and terrain mapping for Unmanned Aerial Vehicles for instance. Raw images captured are processed, filtered and various algorithms are run on them for feature detection. Moreover, these processes need to run in real time environments, outmaneuver obstacles and ensure safe task completion, making them complex systems.

The complexity of autonomous mobile systems comes from different areas such as (a) number of interconnected devices, (b) complexity in operating scenario (such as obstacles and uneven terrain for instance) and (c) the physical devices used. Due to the high number of computations that need to be performed in such applications, autonomous mobile systems often have subsystems integrated to a bigger system, with the subsystems processing their specific domain of application. As the number of subsystems increase, the system provides more functionality, but is prone to the pitfalls of complexity management. An example of an autonomous mobile robot, with various integrated sensors and cameras and a payload working in logistics field, is depicted in Figure 1.

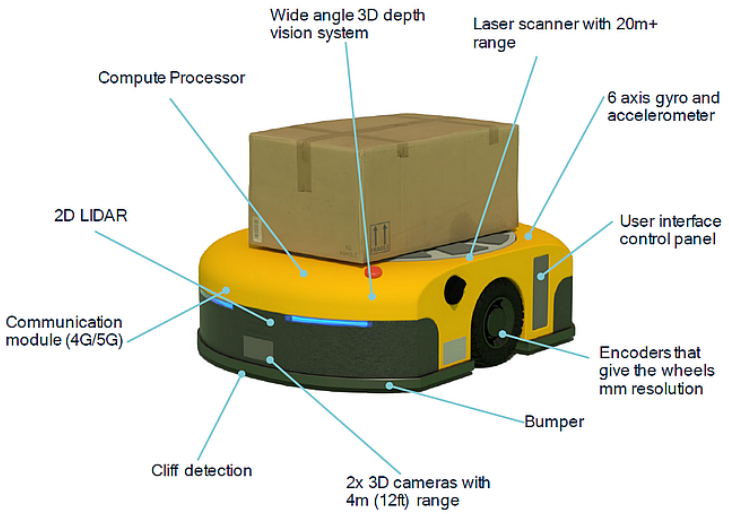

Figure 1: An example of a autonomous mobile robots with integrated sensors carrying a payload in action [3].

Although it can be argued that autonomous systems are meant to operate in difficult scenarios, it nevertheless opens the door for robust control. As newer methods such as Model Predictive control (MPC) for improving lateral and movement control of the robot [4] and Machine and deep learning techniques for efficient decision-making processes [5], have advanced the field, physical nature of devices such as cameras need to supply processed information. Cameras for instance can take images upto few dozen meters, capturing numerous information that the autonomous robot can use. However cameras have limitations in the range of view, and the effects of perspective distortion in the images they take. These images need to be transformed in order for the mobile vehicles to make accurate decisions. An example of such transformation is obtaining a Bird's Eye View of a scene, which is beneficial in applications such as lane detection as it ensures that lanes remain parallel, helping accurate decision making in curved path-ways for instance.

Autonomous mobile robots are not always necessarily large, smaller robots exist as well, with smaller computation platforms. In small-scale embedded applications, the problems of robustness and image processing becomes a noteworthy direction for research and development given the limited resources available for the robot to be controlled with. Meanwhile, rapid prototyping prospects such as 3D printing, open source tools and frameworks such as Python, OpenCV, OpenMV, Linux, etc. have made it possible in using such small embedded devices for hobbyists to researchers alike. These small scale rapidly prototype able systems are useful in research as they use common library and framework instead of propreitery ones, and are cost efficient [6].

Due to the possibilities, varied application areas and the need for such autonomous mobile systems, an exploration in this domain is beneficial. This project intends to work with an integrated mobile robot with two subsystems namely computer vision and mobile robot to achieve a degree of autonomity in self driving by following a lane.

1.1 Motivation

The motivation of this work is primarily to implement computer vision and mobile robots into an integrated system to investigate autonomous driving particularly in the context of small, prototypable systems, and to estabilish an understanding of how the small embedded systems behave. Although tools and frameworks might get complex, the techniques in image processing and control theory remain consistent whether small scale device or larger systems. Hence, there is a motivation to do a research and implementation in autonomous mobile robotics.

1.2 Objectives

The robot system in this project can be divided into two subsystems; an image processing subsystem, and a robot control subsystem. These two subsystems are interconnected to each other.

The objective of this work is threefold namely,

- Capture, extract features and process scenario in simple lane following mobile robot via computer vision. Image processing and transformation are to be utilised for provision of useable data for controlling the robot, as well as increase accuracy of the data, by means of perspective correction.
- Transfer the information from the vision subsystem to the robot subsystem. The communication works between the interfaces between the two subsystems.
- With the information from the image processing part, robot control is to be implemented using system theory and control techniques to enable a mobile robot that can follow a lane.

2 Background information and System Overview

2.1 Review of related works

Autonomous vision based systems range from mobile robots to drones and cars. Such autonomous systems require detection, extraction and identificaiton of relevant features to provide a control for the system. Lane detection is a primary step in autonomous robot mobility. Researchers have designed and experimented mobile robots using conventional techniques such as definition of a region of interest on the image frame captured, evaluation of color segmented areas, edge detection using Canny's method, and line detection using Hough transformation to provide lane information for the mobile robots [7, 8]. However, these conventional methods have some limitations regarding illumination and complexity of curved lanes identification, requiring more advanced image processing techniques [9]. A different method to gather lane information has been performed by researchers by obtaining a top-down view of the environment using Inverse perspective mapping, and determining lane points and positions. These points are fitted as lines, and a Kalman filter is applied to optimise and improve the detection robustness [10]. Illumination issues have been shown to improve by adaptive thresholding in the acquired images [11].

Using similar conventional image processing approaches and prediction of lane points, Rossi et al. implemented a vision based lane following mobile robot on Raspberry pi for image processing and Arduino for motor controls. They implement lane prediction for smooth functioning of the robot due to lags in image processing, and suggest an intermediary controller for divison of processing tasks [12]. Moreover, image processing methods require a relatively good computational platform and efficient algorithms, especially in dynamic environments and for real-time applications. Kwabena [13] has implemented an image detection technique by approximating a line through the points seperated as lanes via image thresholding and conversion of the thresholded image to binary image. This approximation serves as lane itself, and since it uses the thresholded image directly without using edge detection and line transformations from these edges, it reduces the computational load on the controller.

With the provision of lane information from the image processing, the robot utilises it to move and control its motor speed. A simple control method of the PID controller is used by researchers. Although PID controllers are simple and generally applicable, it has been shown that optimisation of the control parameters yield stable output on the robot movement [14]. In tuning the PID parameters, conventional Ziegler-Nichols method [15] is available, as well as more robust methods such as particle swarm optimisation technique is used for a better performance in more complex applications [16].

2.2 System Overview

This section overviews the system and subsystems used in the project. The two subsystems for computer vision and robot controls respectively are seperated and described. Meanwhile relevant background information as well as important information about the devices are provided, and finally the integration of the two subsystems and the communication between their interfaces is presented. Figure 2 provides a glimpse of the integrated system with the Nicla vision attached onto the Alvik Robot via 3D printed parts. The following sections provide details on the individual subsystems.

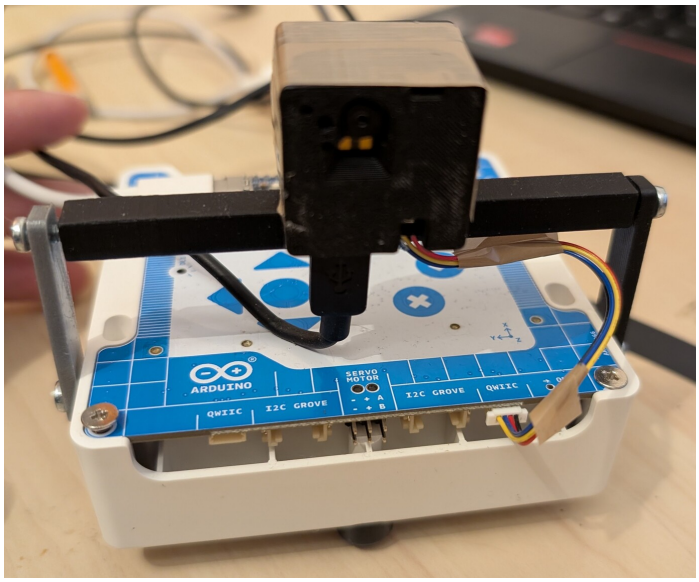

Figure 2: System Overview

Micropython, a subset of Python, optimised for use in microcontroller applications, is used as the programming language in this project. The library used for image processing is the OpenMV library [17]. the OpenMV project is intended for extensible, machine vision application empowered with python for low cost, small devices.

2.2.1 Image Processing system: Arduino Nicla Vision

The image acquisition and processing is done on the Arduino Nicla Vision controller. The microcontroller comes with its embedded camera and processor, meaning that it can serve as a local processor for image processing, and effectively be used as an independent subsystem. The framework used to process images in the Nicla vision is based on Micropython using the OpenMV library. Although similar to OpenCV, this library has limited functions, and is intended for small applications.

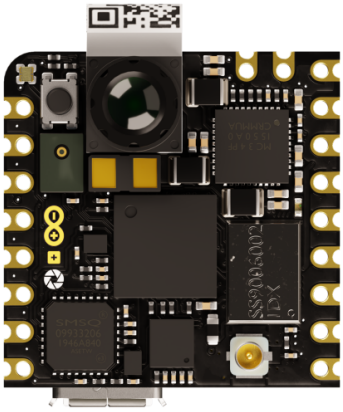

Figure 3: Nicla Vision controller [18]

Microcontroller	STM32H757AII6 Dual Arm®, Cortex® M7/M4
USB connector	Micro USB (USB-B)
Communication	UART, I2C, SPI
Clock speed	Processor (M7) : 480MHz, (M4) : 240MHz
Memory	2MB Flash, 1MB RAM, 16MB QSPI flash
Dimensions WxL	22.86mmx22.86mm

Table 1: Specifications of Nicla Vision

The pin hole camera (located towards the top of the board), is the core component of this device. A general mathematical and geometrical model for a pinhole camera is given in Figure 4.

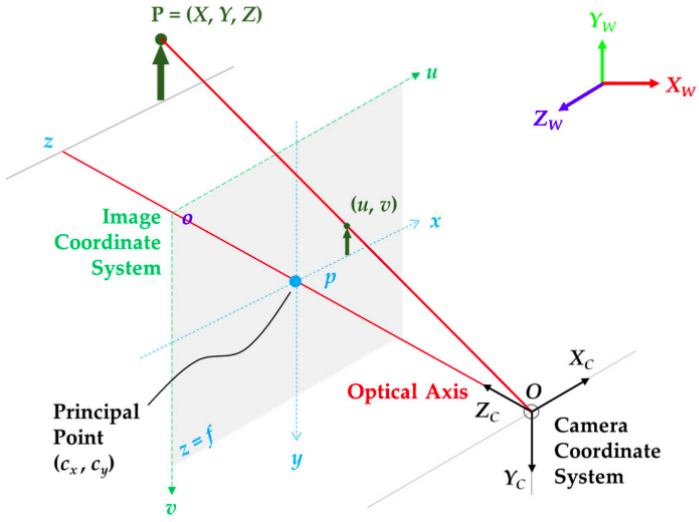

Figure 4: Mathematical model of the pinhole camera [19].

In Figure 4 above, the world is denoted by the coordinate system $q_w = (x_w, y_w, z_w)$, the camera coordinate system is denoted by $q_c = (x_c, y_c, z_c)$, and the image coordinate system is denoted by (u, v). The object point represented by $P(X, Y, Z)$ in 3D casts its image onto image plane, (represented by the two green arrows).

With the focal lengths of the camera in x and y directions given by f_x and f_y , and offset to the coordinate of principal points c_x and c_y, the camera matrix is given by;

$$C_m = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The extrinsic matrix $\begin{bmatrix} R & t \end{bmatrix}$ is a 3x4 matrix. In practice (particularly in this project), the optical axis can be aligned with the z axis, which means the system does not have rotation with respect to the z axis, and hence the matrix can be written as 3x3 matrix. This 3x3 matrix represents a planar homography. In other words, a scene in 2D is projected to a 2D image frame, a 3x3 matrix determines the relationship between the camera system to the world system. This relationship is the (Planar) Homography between two images [20]. This concept can be summarised in Equation 1.

$$q_{image} = C_m \cdot \begin{bmatrix} R & t \end{bmatrix} \cdot \begin{bmatrix} q_w \\ 1 \end{bmatrix} \tag{1}$$

In summary, equation 1 describes the image formed in q_{image} according to the intrinsic parameters like focal lengths of camera characterised by the matrix C_m, its extrinsic parameters Rotation and translation, with respect to the world coordinate system q_w.

2.2.2 Robot Control system: Arduino Alvik with ESP32 controller

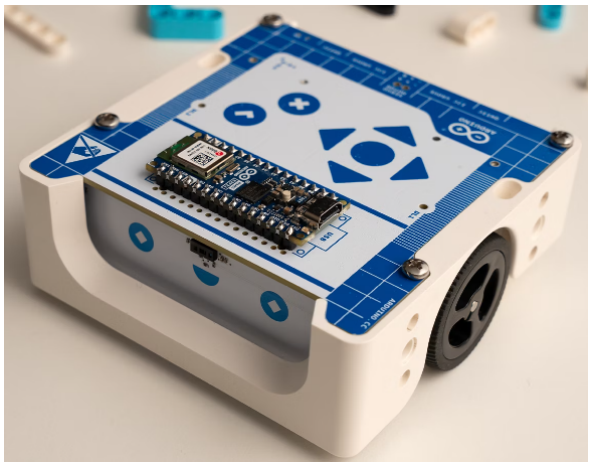

Figure 5: Alvik Robot with ESP32 microcontroller

The Alvik Robot controlled using Arduino Nano ESP32 microcontroller is the mobile subsystem of the project. The controllor as well as the individual motors, are powered by a (rechargeable) battery enclosed in the case. The robot has a roller wheel at the front, and a differential drive (consisting of two individual motors), to move. Furthermore, the Alvik robot has integrated set of sensors and buttons, along with plug in ports for external device communication and control, most noteably the communication ports (QWIIC and I2C Grove). The QWIIC port is

used in this project to communicate with the camera system. The programming for this robot is done on Micropython. The mobile subsystem Alvik is depicted in Figure 5.

Control theory is used to model and control the robot during its movement. A control system akin to Figure 6 requires a reference setpoint which the robot aims to be at, and the deviations to reach the setpoint is the error. The sensor block on the feedback loop is the camera as it updates the error in every time step. In the context of the setup in Figure 5, the ESP32 controller on the robot requires an error for it to process, which corresponds to steering angle in the physical sense.

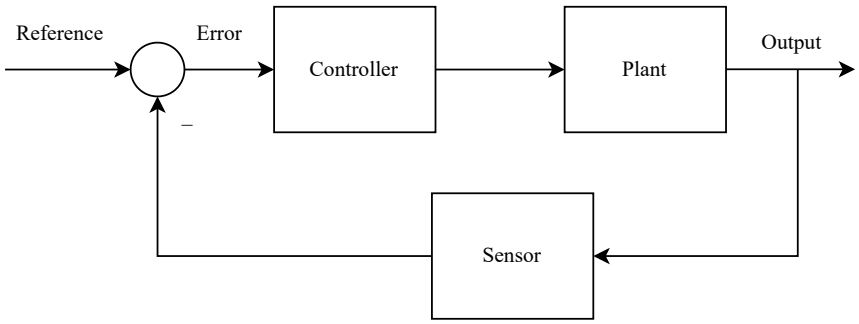

Figure 6: A general control system model.

3 Methodology

This chapter presents the method used to identify lane lines using image processing, determine error parameters to be minimised, and use of this errors to steer and move the robot. The image processing part is done by the Nicla Vision module, which sends the errors as an output to the ESP32 controller on the Alvik robot, that need to be minimised using controls. Both the devices are programmed using Micropython. The Nicla Vision uses the library OpenMV for image processing tasks, other libraries used are provided by the Micropython firmware. Figure 7 presents the overall pipeline.

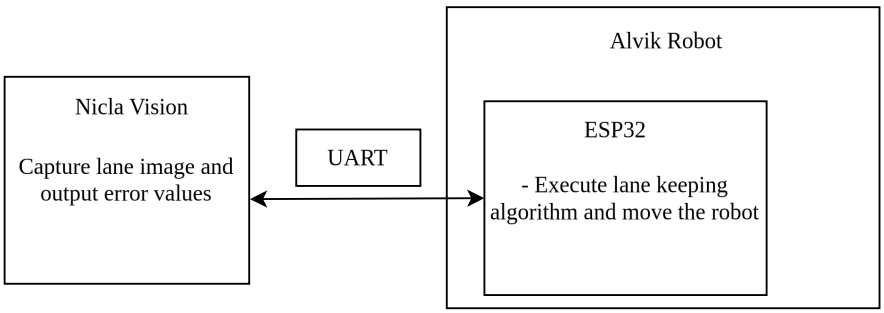

Figure 7: Overall pipeline of the implementation

As seen in chapter System overview, the setup can be modularised into computer vision part wherein the lane detection tasks is performed, and error setpoints are generated, and the other part is the robot movement, where control algorithm to steer and move the robot is implemented to minimise the error from the detected lane. These subsystems are elaborated in the sections below.

3.1 Image Acquision and Processing

The idea of this project is to move the robot to follow a line detected by image processing step. The problem, here to solve is to ensure that at the end, there is an output error that can be minimized. The errors are the angular and distance representative of the drift of the robot away from a desired line (lanes). The pipeline for this implementation is depicted by figure 8.

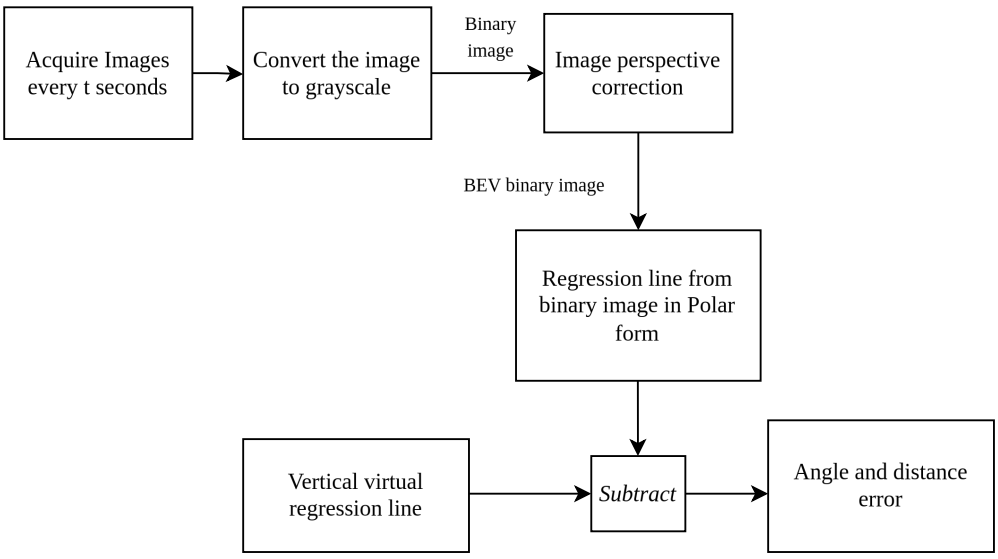

Figure 8: Pipeline for lane image processing

3.1.1 Image processing

The image processing steps on the Nicla Vision consumes considerable computational resource. This needs to be optimised throughout the implementation. To do this, the image frame size is reduced to 320x240 pixels, (QQVGA format). Firstly the camera takes images continuously (every t seconds) in a gray scale format because it was observed that the processing is quicker in gray scale format, without losing features in the image. Initially, as the camera (sensor) starts, it is given 2000 milliseconds to stabalise. The image frame rate is also set (in all cases) to 80 frames per second. This is done so that, whenever there is low computation load on the Nicla device, it can achieve this frame rate, and therefore a smoother image processing. However, during operation the frame rate can reduce.

From the grayscale image, a histogram equalisation is performed, giving an improved separation of thresholded binary image. A histogram equalisation improves the contrast in the captured image. As the camera has low resolution and it is easily affected by shadows, histogram equalisation is essential. Due to shadow, some random noise is visible. Image erosion is further applied on the binary image, which removes pixels from boundaries between binary areas in the image, so that noisy isolated pixels are removed. In other words, image clarity is improved by reduction of effects of shadows.

With these processing techniques applied to the image, the line detection works better, and therefore provides with better regression line for the robot to follow.

3.1.2 Image Rotation correction

Furthermore, an image transformation is applied where the image is rotation corrected. As the field of view of the system at hand spans rather a small area, an inverse perspective transforma-

tion is obtained to generate a bird's eye view. The idea for the transform is largely simplified in that the complete image is rotated such that it matches the rotation of the camera, which is obtained using the IMU (Inertial Measurement Unit) of the Nicla Vision module. The angle obtained from the IMU however, needs calibration for orientation.

In this transformation, the camera also translates by the offset it has from its centre. Figure 9 depicts how the image rotation correction is performed. The openMV library provides a function for rotation correction where it uses rotation of a camera or an image x, y or z axis, along with the translation in x and y axis to rotate the image accordingly. The rotation is obtained from the IMU sensor, and the translation is calculated for x and y using the offset of the camera with 1 mm.

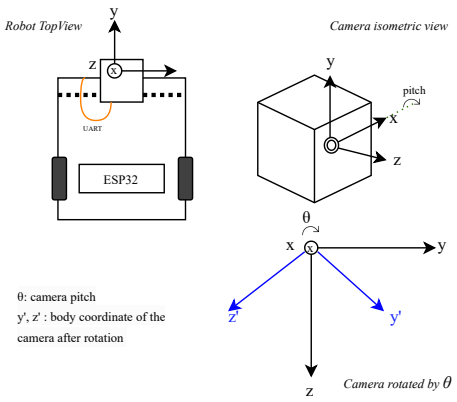

Figure 9: Camera representation.

In Figure 9, the rotated camera coordinate system is taken from the side view. The camera rotates around the inwards pointing x axis. The camera is pointing towards the z axis. The angle between the y and y' and x and x' is the angle that the camera is pointing towards the ground. This is the pitch angle obtained from the IMU sensor.

With the rotation angle in x direction, the image rotation correction library is used to transform the image by the camera rotation (pitch) and the translation (sin and cos components of the offset by the camera pitch). In other words, the image plane is parallel to the camera plane xy.

3.1.3 Image perspective correction

Parallel lines tend to converge at a vanishing point in raw images in perspective view. The vanishing point is usually towards the centre of the image, however, the pose of the camera plays a big role. In a perspective image, for same sized objects closer to the camera, its counterpart further from the camera is depicted smaller. Using this idea, a trapezoid with longer side towards the near end of the image, and shorter side towards the far end can be drawn like shown with green outlines in figure 10. Four points (source points) are required to enclose the trapezoid. When these four points are carefully and strategically chosen, a bird eye

view can be obtained by stretching these points to the full image frame (destination points). This method however is ad-hoc, and requires the practioner to adjust for different camera poses.

On a different note, unlike openMV, openCV has functions that can determine a relationship between the perspective view of an image and the top down view, called the homography matrix. The homography matrix in this case is an estimation, which can be further improved with robust detection algorithms to detect and match corners in the checkerboard patterns as intermediate points for the homography estimation. It is also possible in openCV to make homography estimation from the source and destination points as the homography is fundamentally the relationship between two images of the same scene. The homography transformation matrix is not obtainable in the openMV library, meaning that a mathematical description of the transformation is not obtained. This implies that the transformed region (corner points of the green trapezoid in figure 10), needs to be re-selected if there are changes in the camera setup.

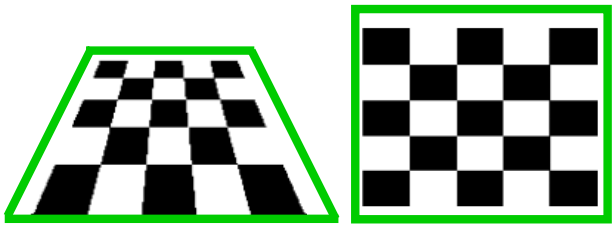

Figure 10: Chessboard pattern views for estimating the homography. Left: perspective view, right: Top-down view.

3.1.4 Lane Detection

The captured and processed images in the Nicla Vision provide binary points in the pixels that is utilised to fit a regression line through the points. As the image size is small, and the camera captures approximately 30 frames of images per second usually in operation, the regression line is constantly updated. Such updating series is then used to make the robot follow the path of the regression line for every time interval (100ms), for each update of regression line. Given enough binary points detected on the lane line, the better the regression is.

Implementation of this idea is performed in polar coordinates. This is due to the fact that polar coordinate system defines points in terms of the angle displaced horizontally (represented by θ), and in terms of the perpendicular distance to it (represented by ρ). These are essential components that are used to control the robot in later stages. This method is adapted from the example provided by OpenMV [13].

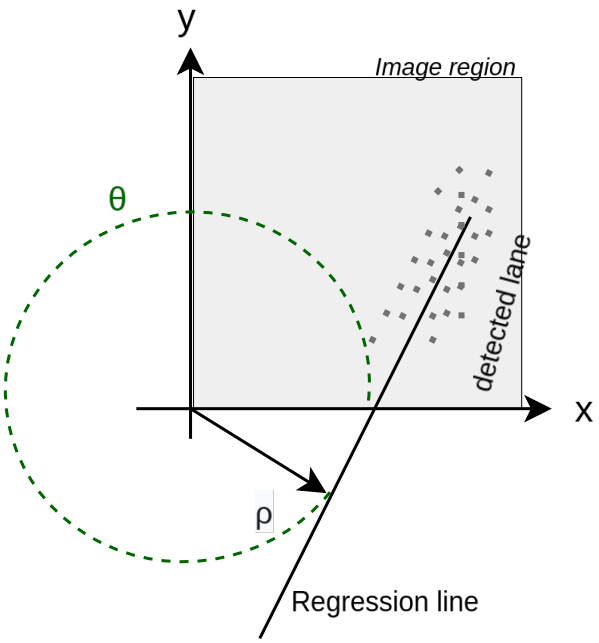

Figure 11: Lane detection model using regression and polar coordinates

The model in Figure 11 depicts some detected points (gray) in the image region of the camera, through which a regression is calculated and a line drawn. In the polar coordinate system, a regression line, or any line for that matter can be modeled using an angle that it makes with the horizontal axis, and the perpendicular distance to the origin from this line. For this project, the solution requires the output as a function of theta and the perpendicular distance to the regression line from the origin represented by the function of $e = f(\rho, \theta)$ and the output from the processing step is provided in the form, $(sin(\theta), \rho cos(\theta))$.

The error values $(sin(\theta), \rho cos(\theta))$ are sent to the Arduino Nano (ESP32) controller in the Alvik Robot, using the UART protocol. The robot is controlled to minimise this error function.

3.2 Data transfer

As the lane is identified by the vision part of the application, it transmits the errors to the robot so that it steers and the error is minimized in the next steps. To transfer the data from one embedded device to another, there are different methods available in Arduino boards, such as SPI (Serial Peripheral Interface), UART and I2C (Inter-Integrated Circuit). The solution in this project uses the UART (Universal Asynchronous Receiver-Transmitter) protocol. UART uses a serial communication protocol where there are two wires connected RX(receiving) and TX (transmission). The Nicla vision is the transmitter and the Arduino ESP32 is the reciever in this setup. Figure 12 shows a generic UART protocol with the Tx and Rx interfaces.

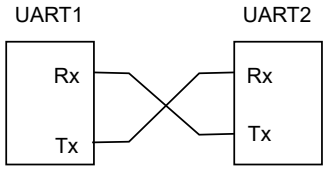

Figure 12: UART Interface

The data transfer is done in the following manner:

- Values of θ and ρ are floats, which are then packed into a binary format, with least significant byte transmitted first, as little endian floating point numbers is specified here. When data is transfered, it is also printed as hexadecimal on the terminal of the openMV IDE, to check if data is being transmitted.
- On the recieving end, the recieved binary data is unpacked into two floating point values for θ and ρ. The recieved bytes are interpreted as little-endian floating point numbers. The unpacked values are then returned in the function.

In the case of the Nicla Vision board, the UART pins as well as the I2C pins share the same ports. The Alvik robot is intended to be used with I2C, due to the designated I2C ports. However, it is possible to use these ports as UART ports. Moreover, the ESLOV pin connector for communication in the Nicla device was used to connect the UART pins which were connected on the other end to QWIIC pins in the Alvik. Although I2C can also be used, there were some blockers in using the I2C protocol. These are discussed in Chapter 5.

3.3 Robot Controls

This subsection explains how the robot is controlled. Firstly, the kinematic model of the differential drive robot is used as the plant model. Then a control logic is designed based on the PID (Proportional, Integral and Derivative) controller. The controller parameters such as gain and time for the PID controller is tuned and tested. Eventually, the end goal is to minimise the continuous error obtained from the Nicla Vision. The Arduino Nano ESP32 is the main control board on the Robot, in which the control program is written using Micropython. The prerequisite for controlling the robot is acquisition of error information $e = f(\rho, \theta)$ in the form, $(sin(\theta), \rho cos(\theta))$ from the image processing part via the UART protocol. There is also a certain amount of delay in the data transfer which needs to be taken into account during the controller design process.

3.3.1 Robot Modelling

The Arduino Alvik Robot is a differential drive robot, meaning that there are two independent drives to steer the robot. For instance, if the robot needs to turn left, the left drive outputs fewer wheel rotations per minute, and the right wheel (outer wheel) outputs higher wheel rotations per minute, ensuring a successful turn. In order to move forward, both the wheels rotate with the same angular speed. A block diagram of controlling such a differential drive is presented in Figure 15. A model of robot is presented in Figure 13.

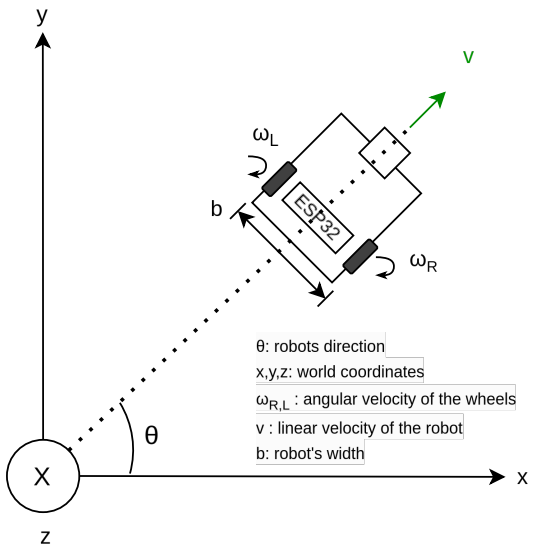

Figure 13: Robot motion model [21]

The robot has a linear velocity v directed according to the variation of wheel speeds. Variation in wheel speed turns the centroid of the robot by an angle θ_{robot}. The linear velocity of the wheels are given by the product of angular velocity and the radius of the wheel, i.e. $v_L = w_L r_w$ for left wheel and $v_R = w_R r_w$ for the right wheel. The magnitude of the overall robot's velocity

is the average of the sum of the two wheel velocities. The following relationships hold for the robot's model in Figure 13.

$$v_L = w_L r_w; \quad v_R = w_R r_w \tag{2}$$

$$v = \frac{v_R + v_L}{2} \tag{3}$$

$$\dot{\theta}_{Robot} = \frac{v_R - v_L}{b} \tag{4}$$

With the model described above, the robot can be controlled. The variables that need to be controlled therefore are the angular velocities of the two wheels (the radius of the wheel is always constant hence it need not be controlled), and thus the orientation of the robot θ_{Robot}. However, the setpoint error for θ_{Robot} is obtained from the regression error in the image processing as described in Chapter 2. Therefore, given a known angular setpoint, the task is to control the angular velocity of the wheels (independently), such that the error function $e = f(\rho, \theta)$ obtained from image processing is minimised.

The robot model here is described in analytical form, however, in the real world, it needs to be adjusted and tuned. For instance, the values for angular error obtained from the Nicla Vision need to be scaled, the kinematic model of the robot must be controlled for overshoot, steady state error and also needs to account for delayed responses.

3.3.2 Controller design

With the model of the robot, and the known control variables, a controller is designed akin to Figure 14. The reference of the system is a vertical regression line at the middle of the screen in the image acquired on the Nicla Vision, ensuring that the angular error $\theta = 0$, effectively ensuring that the robot is right in the middle over the line. As long as it is vertical, the robot should not steer, or in other words, both the drives in the differential drive system must output the same velocity. However, during robot movement, this error is not 0, which introduces an error that is fed through the PID controller. The controller has different gains, and time delays such that the response of the robot to the error is rather smooth, and that the error is reasonably accounted for.

The values from the controller is fed into the plant of the subsystem, which in this case is the differential drives that move the wheels. The drive model is described according to Figure 15. With the differential drive, steering is achieved by manipulation of the speed in each of the individual drives. The parameters that affect the steering is the speed of each drive, and the lateral length of the robot.

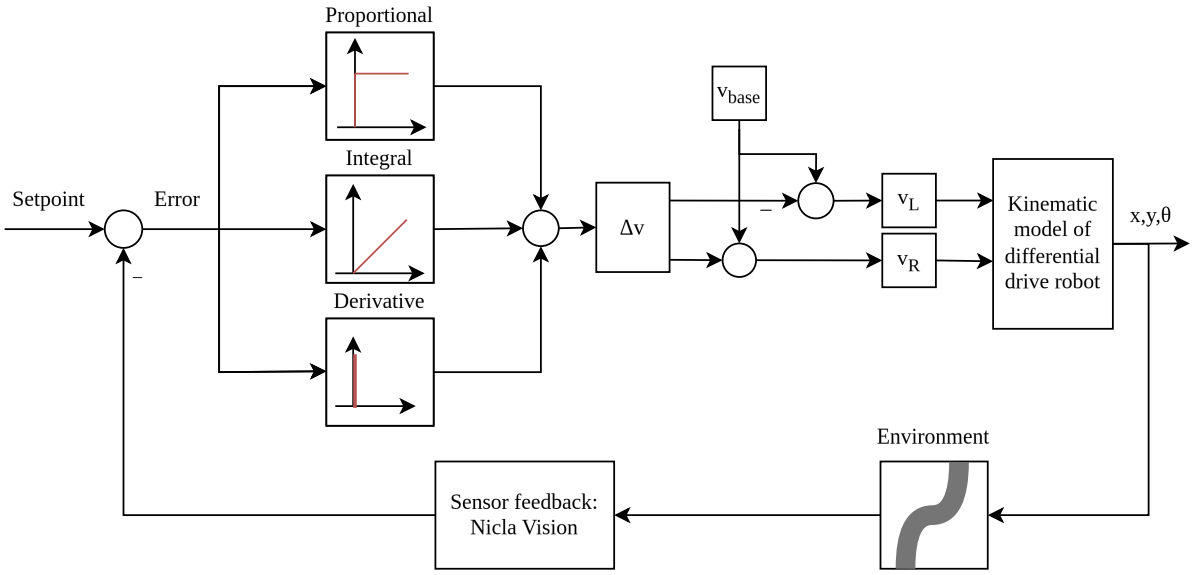

Figure 14: General control flow of the system

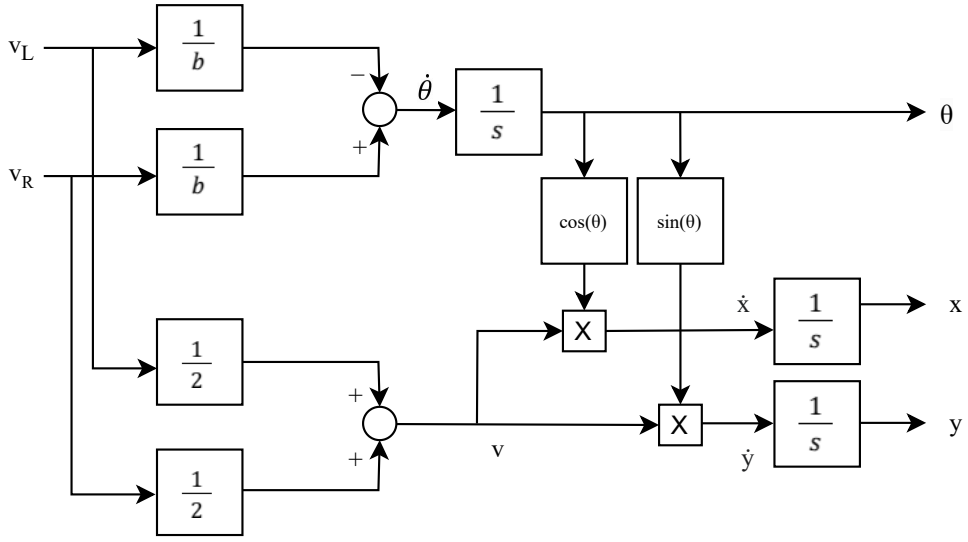

Figure 15: Block diagram of the kinematic model for differential drives [21].

Controller Tuning

Heuristic method namely the Ziegler-Nichols method [15] is utilised to find the controller gains for the PID controller. Initially, the recommended Ziegler-Nichols approach was used, however, due to inconsistencies in the subsystem communication in real time, the Ziegler-Nichols method was adapted for this project. The following steps were performed to collect the PID gains and delay times. Moreover, the tuning is done for the robot moving on a straight line. There is always an error value of small magnitude even on a straight line, that needs to be accounted for, and since the concern is of finding controller gains to ensure quick and smooth drive reaction, this was deemed to be the optimal approach.

1. Only the proportional gain is altered. The gain is started from 1 and increased until slight change in error values move the robot to and fro (oscillation).
2. The Ziegler-Nichols method requires that the oscillation period be used as time constants in the controller. However, due to the dependency on data from another device in this context, the time constant $T_e = 250ms$ is taken as a delayed time (sleep time) between acquiring two consecutive sets of data.
3. The effective proportional gain was evaluated to be $K_e = 20$ from experimentation (step 1). The Ziegler Nichols method recommends that the Proportional gain $K_p = 0.6 \cdot K_e = 12$, which was duly set.
4. The integral gain is supposed to be $K_i = 1.2 \cdot K_e/T_e = 9.6$. The integral gain was changed to $K_i = 8$ during further fine tuning. This is expected due to inconsistencies in subsystem communication. Moreover, there was a slight benefit in reducing the integral gain (about 15%) because the effect of cumulative error is also reduced from the previous step, particularly if the robot is moving well on the lane, the cumulation of error is not desired.
5. Finally, the derivative controller is set to be $K_d = 1.5$. This is again adapted slightly as opposed to Ziegler-Nichols in tha the time constant T_e is not considered. The reason is because the tuning is done in rather pessimistic way, meaning that if the system behaves quickly and data transfer is smooth during certain times, the derivative controls is put high enough such that the overshoot is damped. Due to the small working/operating area of the Nicla Vision, deviations are very sensitive.
6. In essence, the Ziegler-Nichols method is largely utilised, and due to its heuristic nature, in the case of this project it is further fine tuned and adjusted with repeated experimentation.

3.3.3 Program flow

With the controller parameters tuned, the next step is to implement this on an algorithm or a control program that is executed on the ESP32 Microcontroller. The control program for the robot has the following workflow depicted in Figure 16. The control algorithm is executed periodically, every 100ms.

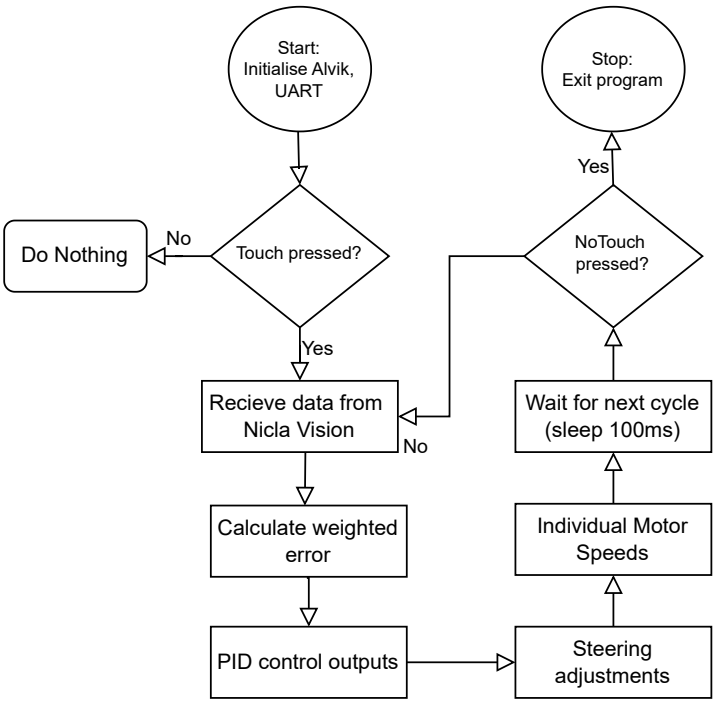

Figure 16: Flow diagram of the robot control program

3.4 Chapter Summary

In this chapter, the methodology to move and steer the Alvik Robot by means of lane following algorithm, utilising image processing is explained. Initially image of the lanes is acquired on the Nicla Vision device, processed for binarisation and perspective transformation. Then using point detection, a regression line was obtained using the detected points which are effectively lane lines. This regression line is designed to be constantly updated effectively providing an error function $e = f(\rho, \theta)$ to be minimised. The error information is transmitted via a UART protocol to the ESP32 controller on the robot.

Kinematic model of the robot is derived and controller design is then done. The PID controller is tuned with (adapted) Ziegler-Nichols method of controller parameter tuning. With this step a relatively stable control system is obtained. In addition, due to the robot having a differential drive, a block diagram of a differential drive steering control is presented. Finally, the program design of the robot control is presented as a flow diagram.

The implementation of this methodology and the obtained results is reported and discussed.

4 Implementation and Results

This chapter presents the implementation of the methodology discussed in Chapter 3. The first part discusses the image acquisition and processing on the Nicla Vision, as well as the data transmission, and the latter sections discuss the Alvik robot controls.

The Nicla Vision camera is placed at the middle of the robot, with the utilisation of 3D printed parts namely, holder arms on either side of the robot, a cross beam running between the two arms, secured by screws. The Nicla Vision is enclosed in a 3D printed box and is stuck onto the beam with adhesive. It must be noted that the Nicla Vision is attached at the back of the robot due to the ESP32 controller being at the front. This means that a usual positive direction for speed becomes negative and vice-versa. It however, does not have any effect on the results.

The robot as a complete system (image processing, communication and robot controller) is tested on a meter long track made using black tape on a pale table surface, as shown in Figure 17. The 5 black dots of tape on either side of the curvature are used as 'debug' points during experimentation. These points help to qualitatively assess how far away the robot is deviating from the lane during curves.

Both the devices are programmed in Micropython, with program being updated via a USB-C cable for the ESP32, and a USB-B (also for power input) for the Nicla Board. The UART connection is established using the pins allocated for I2C communication in both devices.

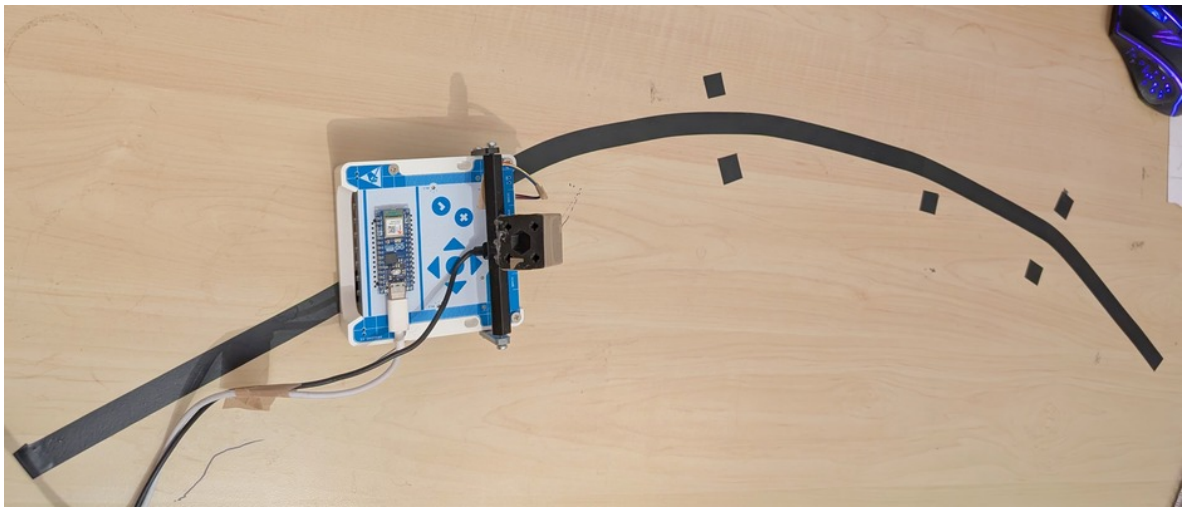

Figure 17: Test setup

4.1 Image Acquisition and Processing

Images can be taken from the camera in Nicla Vision in different frame sizes, frame rates, and colour formats. They have different computational demands on the processor. In this project the image is directly taken in gray scale in QQVGA format (320x240 pixels), and is converted to binary image and a histogram equalisation is done. The frame rate of image acquisition depends on the amount of processing such as thresholding the device has to do at any given scene, the robustness of regression line algorithm, and the noise in the camera. The frame rate has been seen to alter between 20 to 67 frames per second (fps). Higher frame rates provide better results and hence a smoother operation including in the robot control part.

Without image transformation:

The image that is taken from a static camera pose is used in this mode to gather the information from lane detection to control the robot. This mode of image acquisition simply provides a perspective view of the scene. This corresponds to the Perspective Image on Figures 18 and 19 (left side). The detected lane and its regression line is presented in Figure 20. It can be seen that the regression line is mostly at the centre of the lane line. A more continuous and complete result over the whole track is presented in a seperate video.

Perspective Correction:

Perspective distortion affects different applications of computer vision, and need to be corrected. The idea is that four points are to be chosen which encloses a region of interest in the image, that can be transformed to fill the complete image frame, while the non enclosed parts of the image are not included in the transformed frame. If the points are chosen strategically, a usable bird eye view can be obtained.

Figure 18 depicts the perspective correction of an image of a book. The corner points of the aforementioned enclosure are the two extreme points at the bottom of the image frame, and the top two corners of the book, effectively forming a trapezoid with the two edges of the book tending to converge at infinity. In the perspective-corrected image, the enclosed region of interest is stretched to the full image frame, and the edges remain parallel. It can also be seen from the difference in the word 'TIGER' in both the images that a bird eye view is obtained. An implementation of perspective correction in the robot test setup is shown in Figure 20.

Figure 18: Perspective Correction

Rotation Correction:

While perspective correction captures a birds eye view of an enclosed image area, rotation correction attempts to rotate the image to make the flat test surface virtually parallel to the face of the Nicla board, so that the image seems to be taken from top down view, with the help of IMU sensor in capturing the camera rotation relative to the surface. This method has corrected the perspective in that the edges of the book remain parallel in the transformed image compared to the perspective image (left) in Figure 19). However, this method does not provide a good bird's eye view.

Figure 19: Rotation Correction

The perspective as well as rotation correction and bird eye view transformation makes sense if larger scenes are captured. During the tests the different image transformations performed similarly as well as when the images are not perspective or rotation corrected. The implications of this result is discussed in Chapter 5.

A snapshot of the images acquired by the robot during operation in approximately same area of the image, is depicted in figure 20. The first image depicts image without transformation, the second image represents the perspective correction applied. It is visible that for same image scene, the normal images has 'elongated' lane and the perspective corrected image has a more top-down view of the lane, and it covers more of the screen. The final image is the rotation-corrected image.

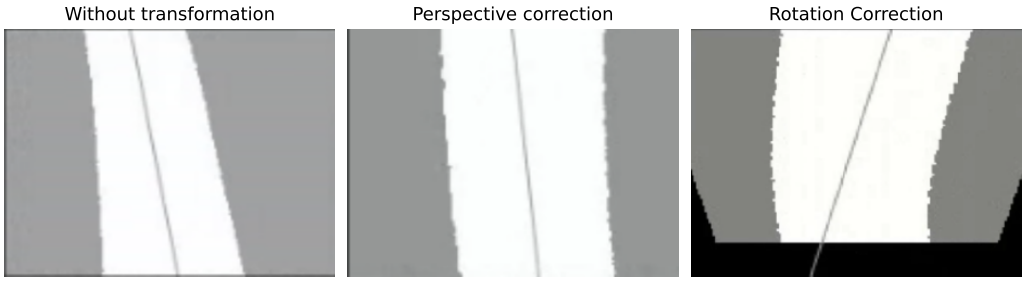

Figure 20: Lane image from camera point of view in perspective and rotation transformations.

4.2 Robot Control

A supplementary series of videos for movement of the robot in different image modes is provided along with this report, as result.

The robot has been tested with wheel speeds ranging from 5 rpm to 30 rpm. A base speed is constant speed, where the robot moves in a straight line. This base speed is altered according to the output of the PID controller in response to the error input, effectively subtracting or adding to the base speed to steer the robot in desired direction.

Beyond the speed of 22rpm, the robot moves in an uncontrolled manner. The higher speeds make the robot steer rather quickly, and it misses the lane in the image frame due to the quick steer. When this happens, the camera detects contrasting features akin to the thresholding for lane detection, from any scene it captures and makes a regression line from these features. An error is therefore always updated, and the wheels rotate to minimise this error, although it does not come from the scene of the lanes. The robot has to be manually placed on the lane to continue further.

At lower speeds 16-22 rpm, similar behavior is observed during curves, where the lane lines go out of the frame, and the robot is unable to find it back. At even lower speeds 5-15 rpm, the robot functions well as the steering is slower, and with constant error updates from the image processing, the robot remains relatively stable. The speed is kept in the video supplement at a speed of 12 rpm. At this speed, the robot has marginal view of the lane even at curved parts.

The control algorithm is executed periodically every 100ms. During this time, it moves continuously based on its previous error. If this sleep duration is removed, the robot can run smoothly even at higher base speeds, however, the data from the camera is not received quick enough, and therefore the motion has to be adjusted here. There are a few reasons for the delay in sending error information, discussed in Chapter 5.

5 Analysis and Conclusion

This chapter provides an analysis of the methods used in this work, the results obtained, and the limitations and future directions that the project can take. In addition, analysis of approaches that were not feasible, as well as validity of the used approach is discussed.

5.1 Analysis

Image processing

The images captured in the project are from a 2 Mega Pixels(MP) camera and in a small image frame (QQVGA format, 320x240 px). Due to limited resource in computing to capture, store and process images with RGB colour format, images were taken in gray scale format. On this grayscale image, a histogram equalisation is done, and the image is converted into binary image. This ensured a high contrast for the implemented algorithm to detect lanes. However, this posed a problem due to the sensitivity of the camera. Since the camera is enclosed in a 3D printed case, the edge of the opening on the case casts a shadow, which is captured by the camera, which is distinguished during the histogram equalisation as a set of lane points. Threshold was limited to mitigate this noise, with a trade off on number of detected lane points.

The range of view of the camera is also limited to a small area. This can be extended in the future using a wide angle lens. However, the images obtained might need to be undistorted, and the camera calibrated. The robustness of algorithms used also affect the computation load on the board, and slow down process.

Robustness of the detected linear regression line also plays a role in the accuracy of the error setpoint produced. For instance, for a non robust computation, the linear regression algorithm uses least squares method to evaluate the regression line from the detected points. Alternatively, for a more robust regression, where the algorithm handles outliers properly, uses the Theil-Sen method, which uses a median filter amongst the thresholded pixels. A median filter is known to handle outliers. The robust method's complexity is of order $O(N^2)$ [17]. This requires more processing resource, effectively slowing down the system, and the data transfer rate. The results presented in Figure 20, the non-robust linear regression algorithm is used. Data transfer rate suffers from the robust algorithm, and moreover, the non robust algorithm provides enough utility for the application here.

Data Transfer

Establishing the communication between Nicla Vision and ESP32 was initially tried using the I2C protocol, which was unsuccessful due to the following reasons:

- **Library function:** While the I2C protocol is available in both devices, it is not made available in the Micropython firmware for Alvik robot, to use it in a Slave configuration. It was possible however, to estabilish a communication between the two devices without the master-slave configuration, although data transmission was not possible.

- **Addressing conflict:** The Nicla vision board uses I2C at port 0x2B, so does the Alvik Robot at the recieving end. Although a communication could be estabilished, data could not be transferred due to this addressing conflict. Moreover, the address could not be changed (in the Alvik Device), due to the master-slave configuration issue.

Due to these conflicts in using the I2C Bus, UART was instead utilised. The data transfer is consistent and successful, as long as data is available. Meanwhile, on the recieving end, the data reception was slowed down by 100ms by using the $sleep(100ms)$ function. This time delay ensures that the data is present for the ESP32 to process.

There are a few identified reasons for the processing delays in data transfer from the Nicla Vision to the Alvik Robot. They can be summarised as follows;

- **Processing delays:** During image processing, there are a few things that have direct impact on the speed. Number of thresholded points in the acquired image is related to the amount of computation resource required to process the image. As mentioned earlier, the size of the image frame also plays a role. A bigger image frame means more thresholded points that the program needs to process. These delays have impacted the data transfer rate.
- **Robustness of regression line fitting algorithm:** The two different ways that regression lines are fit in the processed image also has a big impact on the speed of line generation and therefore transfer of data to the Alvik Robot. A robust algorithm (Theil-Sen method) is slower, while a non robust (least squares method) algorithm is faster.

Robot Control

As mentioned earlier, the error data $e = f(\theta, \rho)$ is transferred in the format $(sin(\theta), \rho \cdot cos(\theta)))$, which means that the angular error value $sin(\theta)$ is a range between 0 and 1, as the angle θ must be constrained within 0 to 90 degrees, (see figure 11 where the image formed is enclosed in one quadrant. With this information, the angular error is not scaled after reception. However, the error value of ρ (obtained from Nicla Vision) is scaled by -1, because in an image axis, the y axis points downwards.

Although the angle is not scaled, the robot's response to close the angle nevertheless needs to be led through certain controllers. The Ziegler Nichols method was used to obtain these PID parameters by moving the robot on a straight line. On each iteration of error, the robot needs to close it down by the delayed time (100ms). Every tenth of a second a new error is supplied, and it is closed in.

In the PID controller, in particular the Derivative and the Integral controllers, time is a variable that is taken into account. The time is in the order of magnitude of 10^{-3} seconds, scaling the output of the PID by 1000 fold. The steering commands provided by the controller is thus divided by 1000. This is visible in the Alvik control program.

5.2 Summary and Conclusion

Image processing techniques and robot control methods are utilised in this project to move an Arduino Alvik Robot along a lane. The robot system has two subsystems, for image acquisition and processing, and for controlling the robot kinematics and motion. The image processing is done on Arduino Nicla Vision, and the robot is controlled by an ESP32 controller, with UART protocol to communicate between the two devices.

Images are taken from the Nicla Vision, which are processed in gray scale and binary format to detect areas of lane markings. These detected points provide a set for a regression to be estimated. When the regression line is updated constantly in short time intervals, it approximates the lane in ideal situation. Using this idea, a lane detection algorithm is implemented. The regression line has distinct parameters (in polar form), namely the perpendicular distance to the line from the origin (represented by ρ), and the angle spanned by this line with respect to the horizontal axis (represented by θ). The ideal condition for the regression line is when it is in the centre of the image frame, vertically oriented, meaning that the robot is on the lane and that it requires no steering. Any deviation from this gives an error function $e = f(\theta, \rho)$, which is minimised.

When the error e exists, the robot needs to minimise this by steering. The values are sent from the Arduino Nicla Vision, which the Alvik Robot processes with some time delay. The Alvik Robot has a differential drive system, which is modelled and the underlying kinematics is utilised for robot controls. Moreover, in application, the system response (response from the differential drives) must be tuned. A PID controller is used, and is tuned using Ziegler-Nichols method for optimum system response to minimise the obtained error. A base speed for both the individual drives is given, and the output of PID is used for steering, which effectively reduces or adds to the individual drives, in order to steer to the desired direction. Finally, the system is tested on a test setup.

6 References

[1] Russell Keith and Hung Manh La: Review of Autonomous Mobile Robots for the Warehouse Environment. 2024. arXiv: 2406.08333 [cs.RO]. URL: https://arxiv.org/abs/2406.08333.

[2] Xiaoqun Liao et al.: Computer vision system for an autonomous mobile robot. In: Proceedings of SPIE - The International Society for Optical Engineering (Oct. 2000). DOI: 10.1117/12.403759.

[3] Dev Singh: Autonomous Mobile Robots: What do I need to know to design one? Apr. 24, 2024. URL: https://www.qualcomm.com/developer/blog/2022/07/autonomous-mobile-robots-what-do-i-need-know-design-one.

[4] Y. Xu et al.: Model predictive control for lane keeping system in autonomous vehicle. In: 2017 7th International Conference on Power Electronics Systems and Applications - Smart Mobility, Power Transfer and Security (PESA). 2017, pp. 1–5. DOI: 10.1109/PESA.2017.8277758.

[5] Ravi Raj and Andrzej Kos: A Comprehensive Study of Mobile Robot: History, Developments, Applications, and Future Research Perspectives. In: Applied Sciences 12.14 (2022). ISSN: 2076-3417. DOI: 10.3390/app12146951. URL: https://www.mdpi.com/2076-3417/12/14/6951.

[6] Sol Pedre et al.: Design of a Multi-purpose Low-Cost Mobile Robot for Research and Education. Sept. 2014. DOI: 10.1007/978-3-319-10401-0_17.

[7] Yeongcheol Cho, Seungwoo Kim, and Seongkeun Park: A Lane Following Mobile Robot Navigation System Using Mono Camera. In: Journal of Physics: Conference Series 806 (Feb. 2017), p. 012003. DOI: 10.1088/1742-6596/806/1/012003.

[8] Abdul Halim Ismail et al.: Vision-based system for line following mobile robot. In: vol. 2. Nov. 2009, pp. 642–645. DOI: 10.1109/ISIEA.2009.5356366.

[9] Gurjyot Kaur and Gagandeep Singh: A review of lane detection techniques. In: International Research Journal of Engineering and Technology (IRJET) 2.3 (2015). URL: https://www.irjet.net/archives/V2/i3/Irjet-v2i3270.pdf.

[10] Yingping Huang et al.: Lane Detection Based on Inverse Perspective Transformation and Kalman Filter. In: KSII Transactions on Internet and Information Systems 12.2 (2018). URL: https://koreascience.kr/article/JAKO201810237886663.pdf.

[11] Mohammed Desouky et al.: Vision-Based Road Tracking of Wheeled Mobile Robot. In: May 2013.

[12] Alfa Rossi et al.: Real-time Lane detection and Motion Planning in Raspberry Pi and Arduino for an Autonomous Vehicle Prototype. 2020. arXiv: 2009.09391 [cs.CV]. URL: https://arxiv.org/abs/2009.09391.

[13] Kwabena Agyeman: Linear Regression Line Following. In: OpenMV Blog (July 16, 2017). URL: https://openmv.io/blogs/news/linear-regression-line-following?srsltid=AfmBOor0nry4f-lngv2smSPELVNmoO2HDMQ6z55ZErbMuPWk4khChn8R.

[14] Sametögüten and Bilal Kabaş: PID Controller Optimization for Low-cost Line Follower Robots. en. 2021. DOI: 10.13140/RG.2.2.22102.98886. URL: http://rgdoi.net/10.13140/RG.2.2.22102.98886.

[15] Nichols N. B. Ziegler J. G.: Optimum settings for automatic controllers. In: Transactions of the American society of mechanical engineers 64.8 (1942), pp. 759–765.

[16] Norhayati A Majid, Z Mohamed, and Mohd Ariffanan Mohd Basri: VELOCITY CONTROL OF A UNICYCLE TYPE OF MOBILE ROBOT USING OPTIMAL PID CONTROLLER. In: Jurnal Teknologi 78.4-2 (2016), pp. 1–6.

[17] OpenMV Documentation. Online Documentation. Accessed on 2024-12-27. URL: https://docs.openmv.io/.

[18] Nicla Vision. Online Documentation. Accessed on 2024-12-27. URL: https://docs.arduino.cc/hardware/nicla-vision/.

[19] De Jong Yeong et al.: Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. Mar. 2021. DOI: 10.3390/s21062140.

[20] OpenCV Documentation. 2024. URL: https://docs.opencv.org/2.4/index.html.

[21] Peter Will: Mechatronic Systems - Mobile robots. Lecture slides. 2024.