NOVEMBER 2023 M.Sc. in Electrical and Electronics Engineering ISMAIL HAJ OSMAN

REPUBLIC OF TÜRKİYE

GAZİANTEP UNIVERSITY

GRADUATE SCHOOL OF NATURAL & APPLIED SCIENCES

VISUAL MOTION CONTROL FOR AUTONOMOUS WHEELCHAIR SIDEWALK FOLLOWING

M.Sc. THESIS

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

BY

ISMAIL HAJ OSMAN

NOVEMBER 2023

VISUAL MOTION CONTROL FOR AUTONOMOUS WHEELCHAIR SIDEWALK FOLLOWING

M.Sc. Thesis

in

Electrical and Electronics Engineering

Gaziantep University

Supervisor

Assoc. Prof. Dr. Tolgay KARA

Co-Supervisor

Assoc. Prof. Dr. Abdul Hafez Abdul HAFEZ

by

Ismail HAJ OSMAN

November 2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Ismail HAJ OSMAN

ABSTRACT

VISUAL MOTION CONTROL FOR AUTONOMOUS WHEELCHAIR SIDEWALK FOLLOWING

HAJ OSMAN, Ismail
M.Sc. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Tolgay KARA
Co-Supervisor: Assoc. Prof. Dr. Abdul Hafez Abdul HAFEZ

November 2023

74 pages

The independent mobility of wheelchair users is pivotal for their overall well-being. Negotiating electric wheelchairs through constrained spaces, especially for those with disabilities, presents notable challenges. In response, this research introduces a pioneering vision-based approach for sidewalk navigation, leveraging tactile paving features and advanced Machine Learning (ML) models. Our methodology encompasses the creation of a specialized dataset, the formulation of a custom control law, and the deployment of a lightweight real-time Gaussian Process (GP) model on a Raspberry Pi platform. Thorough experimentation substantiates the efficacy of our approach, demonstrating accurate and dependable autonomous wheelchair navigation on sidewalks. Beyond technical contributions, our solution offers a cost-effective alternative to conventional sensor-dependent systems, profoundly enhancing user mobility and overall quality of life. By empowering wheelchair users with enhanced navigation capabilities, this research strives to foster independence and inclusion in their daily lives.

Key Words: Sidewalk navigation, Wheelchair, Gaussian process learning, Dataset collection.

ÖZET

Otonom Tekerlekli Sandalyenin Kaldırım Takibi için Görsel Hareket Kontrolü

HAJ OSMAN, Ismail
Yüksek Lisans Tezi, Elektrik ve Elektronik Mühendisliği
Danışman: Doç. Dr. Tolgay KARA
İkinci Danışman: Doç. Dr. Abdul Hafez Abdul HAFEZ

Kasım 2023

74 sayfa

Tekerlekli sandalye kullanıcılarının bağımsız hareket kabiliyeti onların genel refahı açısından çok önemlidir. Elektrikli tekerlekli sandalyelerin kısıtlı alanlarda, özellikle de engelli kişiler için kullanılması, kayda değer zorluklar içermektedir. Buna yanıt olarak bu araştırma, dokunsal yüzey özelliklerinden ve gelişmiş Makine Öğrenimi (ML) modellerinden yararlanarak kaldırım navigasyonu için öncü, görsel veri temelli bir yaklaşım sunmaktadır. Metodolojimiz, özel bir veri kümesinin oluşturulmasını, özel bir kontrol yasasının formüle edilmesini ve hafif bir gerçek zamanlı Gauss Süreci (GP) modelinin Raspberry Pi platformunda devreye alınmasını kapsar. Kapsamlı deneyler, kaldırımda doğru ve güvenilir otonom tekerlekli sandalye navigasyonunu göstererek yaklaşımımızın etkinliğini kanıtlamaktadır. Teknik katkıların ötesinde çözümümüz, geleneksel sensöre bağlı sistemlere uygun maliyetli bir alternatif sunarak kullanıcı hareketliliğini ve genel yaşam kalitesini önemli ölçüde artırır. Bu araştırma, tekerlekli sandalye kullanıcılarını gelişmiş navigasyon yetenekleriyle güçlendirerek, bağımsızlığı ve günlük yaşamlarına dahil olmayı teşvik etmeye çalışmaktadır.

Anahtar Kelimeler: Kaldırım Takibi, Tekerlekli Sandalye, Gauss Süreci Öğrenmesi, Veri Seti Toplama

"Dedicated to my family"

ACKNOWLEDGEMENTS

I am deeply grateful to my supervisor, Assoc. Prof. Tolgay KARA, for his unwavering guidance, support, and expertise throughout the course of this study. His continuous encouragement and motivation played a pivotal role in the successful completion of this thesis.

I extend my heartfelt appreciation to my co-supervisor, Assoc. Prof. Abdul Hafez Abdul HAFEZ, whose inclusion in his research project and invaluable advice have been instrumental in the realization of this thesis. Without his expertise and guidance, this work would not have been possible.

I would also like to convey my love and profound gratitude to my family and friends for their unwavering support and understanding throughout this challenging journey.

Furthermore, I would like to acknowledge the support provided by The Scientific and Technological Research Council of Turkey (TUBİTAK) through project 117E173. Their financial assistance has been instrumental in the successful execution of this research endeavor.

My sincere appreciation also extends to all individuals who have contributed directly or indirectly to this study. Their insights and assistance have greatly enriched the quality and depth of this thesis.

Finally, I would like express how sad I am for my people in Syria and Palestine, Gazza who have been killed and suffered these days. I want to convey this success to all Martyrs and their loved ones and extend heartfelt condolences for them, praying for Syria and Palestine to be free soon.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OFABBREVIATIONS

IBVS	Image-based visual servoing
GP	Gaussian Process
HOG	Histogram of Gradients
EPW	Electric powered wheelchair
VS	Visual servoing
PID	Proportional integral derivative
SLAM	Simultaneous localization and mapping
YOLO	You only look once
ML	Machine learning
TP	Tactile Paving
PW	Power wheelchair
DVS	Direct visual servoing
ResNet	Residual network deep learning model
SLSQP	Scikit-learn python library
CNN	Convolutional neural network
MSE	Mean square error
AMSE	Absolute mean square error
SE	Squared exponential kernel function
δ	Kronecker delta
GT	Ground truth
IBVS	Image-based visual servoing
GP	Gaussian Process
HOG	Histogram of Gradients

CHAPTER 1
INTRODUCTION

Motivation of study

The motivation to study the visual motion control of wheelchairs stems from the critical importance of independent mobility for wheelchair users, as it profoundly impacts their social and physical well-being. Extensive research has consistently demonstrated that limited mobility can lead to social disorders such as depression, anxiety, and a fear of isolation [1], [2]. However, maneuvering an electric wheelchair, especially in constrained environments, presents significant challenges for individuals with disabilities. In cases of severe disability, users often rely on the assistance of others to navigate effectively. Moreover, safety is a major concern as individuals with neuromuscular or vision disabilities face difficulties in steering wheelchairs in confined spaces, thereby increasing the risk of collisions [3], [4]. Over the past 25 years, electric powered wheelchairs (EPWs) have emerged as the primary mobility aid for individuals with reduced mobility and vision [5]. Recently, EPWs have gained considerable attention as versatile devices capable of meeting the diverse demands of users [6]. To address the aforementioned challenges, various approaches have employed Visual Servoing (VS) techniques, widely utilized in robotics applications to control the movements of robots' end-effectors based on visual sensor input [7]. Specifically, image-based VS relies on information extracted solely from images, which can encompass points, lines, or other geometric features. Understanding and refining the visual motion control of wheelchairs through VS techniques hold promise in enhancing the mobility and overall well-being of wheelchair users. By leveraging visual information and applying advanced control strategies, it becomes possible to empower individuals with disabilities to navigate indep

endently and safely in various environment

1.2 Literature Review

Image-based VS, in particular, relies on information extracted solely from images, which can include points, lines, or other geometric features. For guiding a standard two-wheeled mobile robot's movement, research focuses on efficient control methods. The work presented in [8] introduces a robust motion control using behavior-based fuzzy control. Similarly, [9] explores a nonlinear PID controller for resilient trajectory tracking in wheeled mobile robots.

This research direction carries significant social and economic implications. It provides support for individuals with disabilities enabling them to enhance their mobility. We propose a vision-based cost-effective alternative to existing systems that rely on multiple expensive sensors for wheelchair navigation.

The literature encompasses a wide range of studies focusing on wheelchair navigation tasks. For instance, researchers have explored the use of VS to navigate wheelchairs through corridors, as demonstrated in works such as [10] and [11]. The problem of wheelchair navigation has received significant attention in the literature; however, the task of navigation through the sidewalk has been relatively understudied. Numerous studies have addressed the challenge of sidewalk navigation

for various robots. For example, works presented in [12], [13] explore the use of lidar sensors, semantic point cloud networks, and Hybrid filter SLAM to aid sidewalk navigation. In [14], navigation of a quadruped robot involves adhering to a map-derived route plan. Moreover, [15] and [16] introduce autonomous sidewalk navigation through an inverse reinforcement learning model. The work presented in [17] discusses how a general goal-conditioned model for visionbased navigation can be trained on data obtained from many distinct but structurally similar robots, and enable broad generalization across environments and embodiments. An alternative vision-based approach is presented in [18], utilizing sidewalk segmentation and cost-map generation modules. Additionally, [19] proposes robust navigation for an unmanned aerial vehicle via a 3D local path planning algorithm. In this work, we aim to address this important problem and shed light on its significance again. While previous attempts have been made to accomplish sidewalk navigation, such as the work reported in [20] that has utilized color histograms for road detection and

classification, and another work in [21] that focused on the detection of tactile paving using deep learning algorithms like YOLO-V3, there is still room for improvement. More recently, [22] presented an automatic tactile paving extraction method to detect and distinguish tactile paving areas from other areas on the road. While these approaches have made valuable contributions, our approach introduces a new state-of-the-art way for wheelchair navigation on sidewalks, utilizing tactile paving features and a Machine Learning (ML) model. Our work encompasses the development of a novel dataset and training an ML model to navigate through the sidewalk. This dataset consists of input images and the corresponding velocity to be commanded to the wheelchair for navigation purposes. A novel control task is defined to derive the control law to be used in calculating the velocity values from the extracted tactile paving surfaces. Furthermore, we have trained a machine learning model using our dataset to predict the velocity of the wheelchair corresponding to the real input images captured from a camera mounted on a real wheelchair. Through the combination of tactile paving features, and deep learning algorithms, our work presents a robust and effective solution to sidewalk navigation by the wheelchair. The entire process, from image acquisition to predicting velocity signals using a machine learning model, has been successfully implemented on a Raspberry Pi pocket computer. This proposed approach focuses on addressing the sidewalk navigation task for an Electric Powered Wheelchair (EPW). A Gaussian Process model has been adopted and trained using the generated data. This trained model is lightweight, allowing it to perform real-time inferences at a rate of 7 Hz, satisfying the timing requirements for wheelchair control tasks. It's worth noting that one of the motivations behind our work is the utilization of low-cost vision sensors in designing these systems. This work introduces a novel visual servoing approach to the sidewalk navigation challenge, which for the first time utilizes the edges of tactile paving as input features for the control law. It also presents a robust method for extracting these edges from tactile paving images. This newly developed combination of features serves as input edge features for the control law, generating the necessary data for model training. The dataset was annotated with velocity values corresponding to each input image of the sidewalk captured by the camera attached to the wheelchair. Notably, our work pioneers the application of a new control law and line features specifically tailored for sidewalks, which have not been explored in the existing literature. By leveraging these innovative features and the new control law, our work aims to achieve precise and resilient wheelchair

navigation, thereby ensuring safer and more efficient travel for users in realworld scenarios. The experimental results effectively demonstrate the prowess of our proposed method in achieving accurate and reliable autonomous wheelchair navigation on sidewalks. Our contributions can be summarized as follows:

1) A novel visual servoing approach that utilizes the two edges of the sidewalk for navigation along the sidewalk.

2) The creation of a new dataset suitable for training various machine learning models to facilitate navigation on sidewalks.

3) The development and deployment of a lightweight real-time Gaussian Process (GP) implementation on a Raspberry Pi edge device.

4) Design and implementation of an efficient control method for autonomous sidewalk following

1.3 Organization of Thesis

The structure of this paper is organized as follows. In Chapter 2, we provide an overview of the hardware components used in wheelchair hardware setup. Then, we give an introduction to the definition of the sidewalk navigation task and the basics of visual servoing. Chapter 3 delves into the details of visual features and the newly designed control law that enables the successful execution of the sidewalk following task. Moving on to Chapter 4, we introduce the GP model along with the dataset that was generated and utilized for training the model. Then, we shed light on the motion controller used in our experiments . Chapter 5 is dedicated to the presentation of our experimental results, showcasing the outcomes and effectiveness of our approach. Finally, in the concluding sections, we present the conclusions drawn from our research, as well as a comprehensive list of references.

CHAPTER 2
SIDEWALK NAVIGATION TASK AS VISION-BASED CONTROL PROBLEM

2.1 The Hardware Setup of Wheelchair System

2.1.1 The Electric Wheelcahir

The electric wheelchair represents a distinctive mobility device with inherent characteristics that distinguish it markedly from conventional automobiles and other modes of transportation. As illustrated in Figure 1, a representative model of such wheelchairs features a distinctive configuration. Notably, the rear wheels are equipped with propulsion systems, each independently driven by DC electric motors. In contrast, the two front caster wheels serve as passive components, bearing a portion of the wheelchair's weight and facilitating directional changes by adjusting their orientation during turns. In the analysis of this mechanical system, the consideration of forces and torques becomes pivotal for predicting its motion dynamics. However, particular emphasis is placed on the dynamic equations governing the rear wheels due to their integral role driven by electric motors. This emphasis reflects the critical importance of understanding and modeling the rear wheel dynamics in comprehending the overall behavior of electric wheelchairs.

Figure 1. Typical Electric Wheelchair

2.1.2 Raspberry Pi 4.0 with V2 Camera Module

Incorporating the Raspberry Pi Camera Module into robotics products alongside the Raspberry Pi 4 brings even more advantages. The module's affordability makes it a cost-effective choice for capturing visual data in robotic applications. Its universal compatibility with all Raspberry Pi models ensures flexibility and ease of integration. Additionally, its capability to output 4K video at 60 Hz and support for dual monitors enable high-quality visual feedback, a crucial feature for advanced robotic applications, enhancing user experience and situational awareness. Moreover, the camera module delivers exceptional image quality and supports high dynamic range (HDR) imaging, enabling robots to capture detailed and well-balanced visuals in diverse environments. This combination of the Raspberry Pi 4's enhanced capabilities and the Camera Module's features offers a compelling solution for robotics, empowering them with superior computing power and advanced visual perception. Figure 2 shows raspberry pi 4 module and its camera.

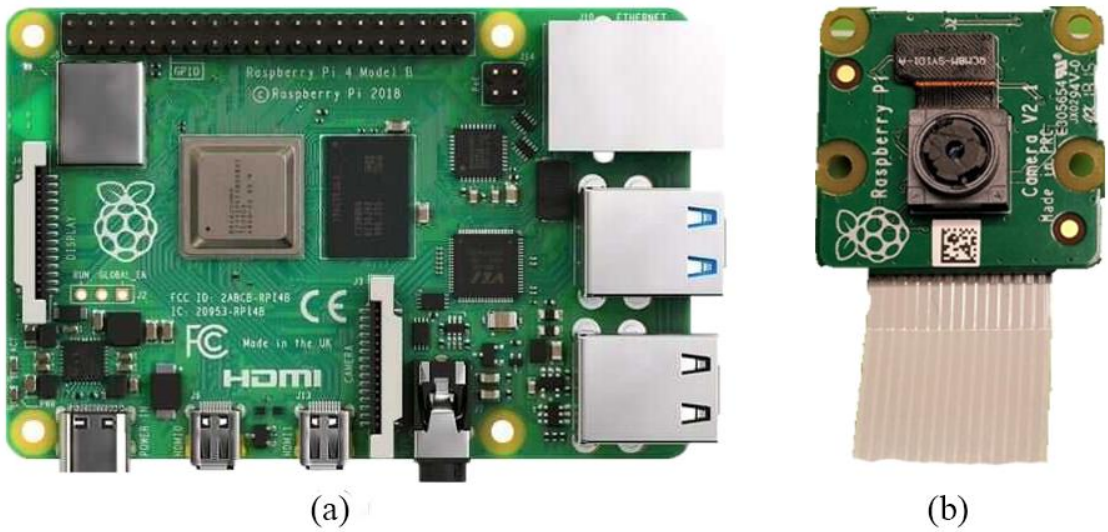


Figure 2. Shows Raspberry Pi 4 module with its Camera V2.1 where (a) represents the Raspberry Pi 4 and (b) represents the camera.

2.1.3 Sabertooth Motor Driver

The Sabertooth motor driver represents a groundbreaking innovation ideal for robotics applications. Its pioneering synchronous regenerative motor control technology not only enhances efficiency by recharging batteries during robot deceleration or reversal but also allows for rapid stops and reversals, imparting agility to robots. Sabertooth offers versatile motor control methods, including analog voltage, radio control, serial, and packetized serial communication, making it adaptable for a wide range of robotic systems and enabling the construction of increasingly complex robots over time. With support for both independent motor control and speed+direction operating modes, Sabertooth is perfectly suited for various robotics configurations, making it an indispensable component in the field of robotics. The following figure illustrates the Sabertooth motor driver, showcasing its pivotal role in robotics.

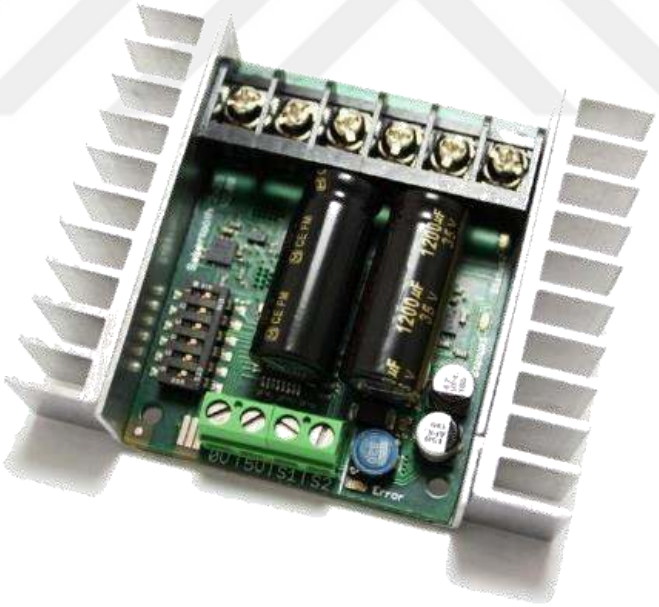


Figure 3. Sabertooth motor driver

2.1.4 Power Sources (24V and 5V)

The experimental setup for the wheelchair project comprises two distinct power sources meticulously chosen to cater to specific system components. The primary

power source consists of a 24V 20 AH rechargeable battery, strategically allocated to energize the wheelchair's DC motors and the Sabertooth motor driver. This high-capacity battery serves as the main power reservoir for the propulsion system, ensuring sustained and reliable motor operation. In parallel, a secondary power source in the form of a 5V power bank has been employed to supply power to the Raspberry Pi 4, the computational core of the setup. This dual-power arrangement ensures optimal performance, with the 24V battery dedicated to the demanding motor control tasks, while the 5V power bank provides stable and dedicated power for the Raspberry Pi, collectively establishing a robust and well-balanced power architecture for the experimental wheelchair system.

Figure 4. Power sources used in wheelchair setup during experiments (a) represents 24V battery and (b) represents 5V power bank

2.1.5 Circuit Diagram and Connections of Hardware System

Figure 5 presents a comprehensive circuit diagram depicting the interconnected electronic components within the wheelchair system. This diagram serves to elucidate the power distribution and control mechanisms. Specifically, a 24V power supply serves as the primary source, energizing the Sabertooth motor driver, which assumes the pivotal role of governing the operation and movement of the wheelchair's motors. In parallel, a distinct power source, the 5V power bank, is employed to provide electrical energy to the Raspberry Pi 4 module. The Raspberry Pi module, in turn, establishes communication and control over the Sabertooth motor driver through a

micro-USB cable, facilitating the precise management of the wheelchair's motor functions. Figure 5 serves as an illustrative representation of the interconnected hardware components and their respective connections within this system.

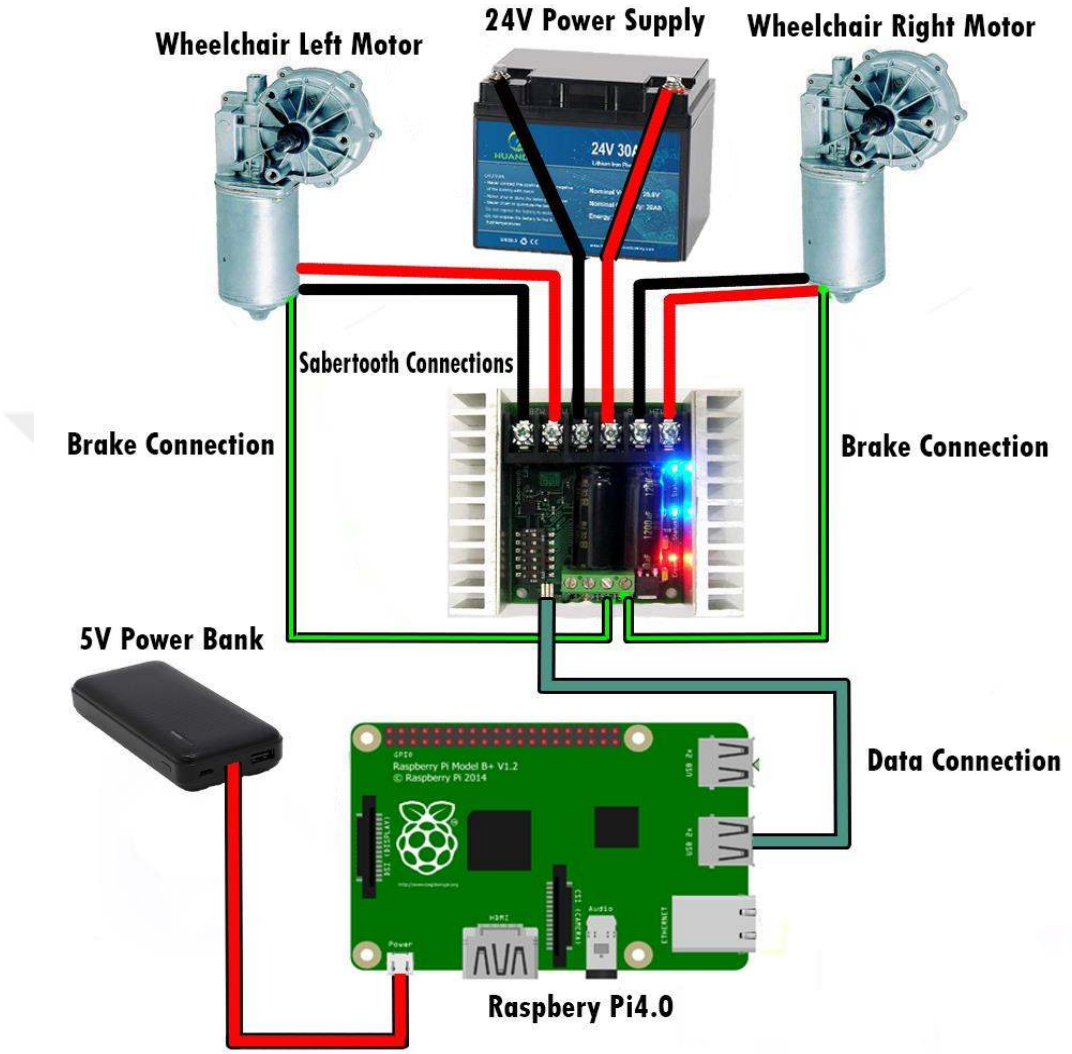


Figure 5. Circuit diagram of hardware parts in the wheelchair system

2.2 The Wheelchair Model

The experimental work presented in this article focuses on the implementation of a real wheelchair equipped with various components. The wheelchair comprises two large rear wheels driven by a differential controller and two separate DC motors, two passive front wheels for rotational movement assistance, and two small support wheels positioned behind the rear wheels for stability. A visual representation of the wheelchair model can be seen in Figure 6.

In order to analyze the kinematics of the wheelchair, we can consider it as a unicycle robot with non-holonomic constraints, as discussed in [23][24]. These constraints arise due to the wheelchair's differential drive system, where the two wheels are independently controlled by DC motors, along with the presence of four caster wheels for stability [24]. To regulate its motion, the wheelchair operates with two primary control velocities: the translational velocity (v) for forward/backward movement and the angular (steering) velocity (ω) for rotation and direction changes. In the modeling task, several Cartesian frames are considered. The world frame (F_W) is represented by F_W(W, x_W , y_W , z_W) , the wheelchair frame (F_r) is attached to the midpoint of the segment connecting the centers of the two driven wheels, denoted as F_r(R, x_r , y_r , z_r). The camera frame (F_c) is rigidly fixed to the wheelchair, and its representation is F_c(C, x_c , y_c , z_c). Figure 1 illustrates these frames and highlights their relationships. The transformation between the robot frame (F_r) and the camera frame (F_c) is the transformation that we will used in our work. By mounting the camera on the wheelchair at a height (h) relative to the

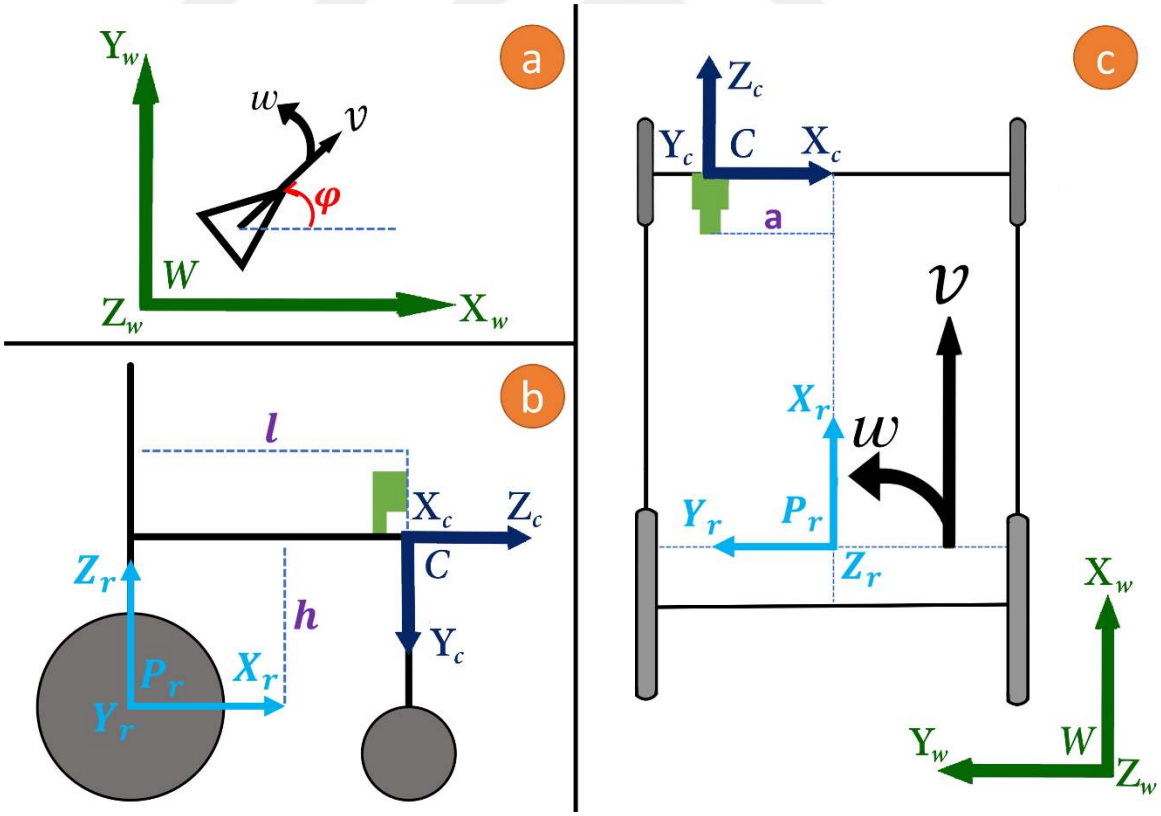


Figure 6. Wheelchair modeling related figures with the camera C, robot P_r , and world W frames (a) The robot model (b) Side view of the wheelchair (c) Top view of the wheelchair

segment formed by the centers of the actuated wheels, the translation vector ctr between the F_r and F_c frames can be defined as ${}^{c}t_r = [\mathrm{w} \quad \mathrm{h} \quad -\mathrm{l}]^T$.

This modeling provides a foundation for understanding the wheelchair's kinematics and the relationship between its components. It allows us to establish a clear understanding of the coordinate systems involved, enabling subsequent analysis and control strategies to be developed based on these relationships. Figure 1 illustrates the frames involved in our study. The $F_W(\mathrm{W}, \mathrm{x}_W , \mathrm{y}_W , \mathrm{z}_W)$ frame represents the world frame, while $F_r(\mathrm{R}, \mathrm{x}_r , \mathrm{y}_r , \mathrm{z}_r)$ denotes the frame attached to the wheelchair, positioned at the middle of the segment formed by the centers of the two deferentially actuated wheels. Furthermore, we define $F_c(\mathrm{C}, \mathrm{x}_c , \mathrm{y}_c , \mathrm{z}_c)$ as the camera frame rigidly fixed to the wheelchair, where C corresponds to the camera's optical center [25]. Our primary objective is to determine the transformation that establishes the relationship between F_r and F_c. This transformation allows us to relate vectors expressed in the robot frame to their counterparts in the camera frame.

$$P_c = {}^{c}R_r P_r + {}^{c}t_r \tag{1}$$

where P_c is a vector P with respect to camera frame and P_r is the same vector with respect to robot frame. If we mount the camera on the wheelchair such that it is at height h from the segment connecting the centers of the two actuated wheels, then the translation vector ${}^{c}t_r$ between F_r and F_c will be ${}^{c}t_r = [\mathrm{w} \quad \mathrm{h} \quad -\mathrm{l}]^T$. The transformation matrix that relates the camera velocity to the robot velocity is denoted as ${}^{c}W_r$. This matrix allows us to establish the relationship between the motion in the camera frame and the corresponding motion in the robot frame. By utilizing it, we can accurately analyze and control the velocity of the wheelchair based on visual information captured by the camera. The matrix is given by

$${}^{c}W_r = \begin{bmatrix} {}^{c}R_r & {}^{c}t_r \times {}^{c}R_r \\ 0_{3\times3} & {}^{c}R_r \end{bmatrix} \tag{2}$$

where ${}^{c}R_r$ is the rotation matrix that models the fixed orientation of the robot frame relative to the camera frame, and it is given by

$$ {}^{c}R_r = \begin{bmatrix} \sin\theta & -\cos\theta & 0 \\ 0 & 0 & -1 \\ \cos\theta & \sin\theta & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \quad (3) $$

The rotation matrix ${}^{c}R_r$ can be applied for a camera with any orientation relative to the y axis. However, in our specific case, we only have one camera with an orientation angle of θ = 0◦. In the global coordinate frame, the velocity of the unicycle robot depicted in Figure 1 can be expressed as follows, as described in the work by [23]:

$$ \begin{aligned} v_x &= v\cos\varphi \\ v_y &= v\sin\varphi \\ \omega_z &= \omega \end{aligned} \quad (4) $$

where $\mathrm{u} = (\mathrm{u}, \omega)$ represents the control velocities.

The kinematic model of the robot with respect to the global coordinate frame and can be expressed compactly as

$$ {}^{g}V_r = J_r \mathrm{u} \quad (5) $$

where u is the robot velocity. By substituting the kinematics equation (4) into (5), we have

$$ {}^{g}V_r = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (6) $$

Then camera velocity can finally be written as

$$ u_c = {}^{c}W_r \, J_r \mathrm{u} \quad (7) $$

2.3 Sidewalk Navigation Task Definition

The objective of the wheelchair sidewalk navigation task is to enable the wheelchair to track and follow the tactile paving while navigating the sidewalk, assuming it is moving from place A to place B. The wheelchair perceives the sidewalk through a camera mounted on the wheelchair looking forward to the direction of the wheelchair's motion. The perceived images by the camera are processed and a set of suitable features are extracted, an image of the tactile paving on the sidewalk is presented in Figure 7. These features are fed as an input to the trained ML model, which in turn

generates the required velocity signals that keep the wheelchair as closer as possible to the tactile paving. By selecting a proper distance function to be minimized, this navigation problem can be considered similar to the image-based visual servoing (VS). The formulation of the navigation task as VS problem helps us in collecting and annotating the required data to train the ML model which predicts the suitable velocity signal to drive the wheelchair along the sidewalk. To do this, we have designed a VS control law that considers the edges of the tactile paving as features. The minimization process considers reference values for the edge lines of the tactile paving and tries to minimize the distance between them and the currently observed edge lines in the captured image. In other words, our aim is to minimize the error between the current position of the wheelchair and its desired position when it is exactly at the center of the tactile paving. A diagram that visualize the process of the sidewalk following task is represented in Figure 8. The details of the control law design, the Jacobian matrix calculation, and the feature extraction and combination are presented in the next section. The designed control law is used to generate the corresponding velocity to each of the training images. The velocity annotated set of image data is used then to train the ML model to be able to predict the suitable velocity given a captured image in the run time. Similar previous work [26] uses a dataset of corridor images collected from the web to train a Resnet deep model to predict the suitable velocity. The work presented in [27] used the same data to train a GP model.

Experimental evaluations have shown superior performance and faster running time by the GP model. This allows us to adopt the GP model here in this work. In our implementation of the sidewalk following task, we simplify the control strategy by focusing on calculating the angular velocity ω_y around the y-axis through the control law, while treating the translational velocity as a constant. Our proposal aims to design a controller that exclusively relies on the visual feature vector x extracted from the current image I, eliminating the need for explicitly computing the Jacobian matrix. By simplifying the control scheme to a single degree of freedom, i.e. specifically the angular velocity ωy around the y-axis, we derive the following equation. We will only control the angular movement of the wheelchair according to visual features extracted from tactile paving images. The calculation of the angular velocity values ωy relies on a geometric-based control law that heavily relies on precise extraction of geometric features and techniques for detecting lines. To enhance the accuracy of our

observations, we leverage advanced image processing techniques. This improved accuracy enables us to obtain reliable data for training the machine learning model. A comprehensive discussion on the processes of data collection, model training, and testing can be found in section IV. Moreover, section III-C provides a detailed explanation of the visual data detection and extraction methods employed in our work.

Figure 7. Image sample of the tactile paving used as training dataset

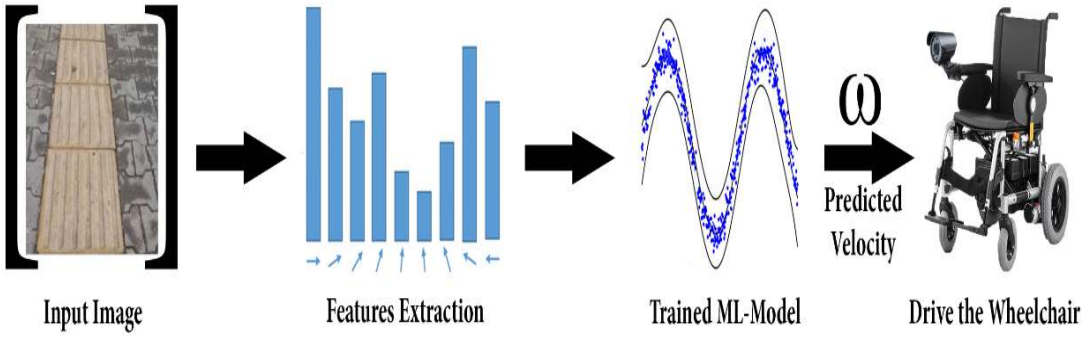


Figure 8. Diagram shows main steps included in sidewalk following task

CHAPTER 3
A NEW FEATURES AND CONTROL LAW FOR THE SIDEWALK TASK

The focus of this section is on presenting the development of a cutting-edge feature extraction method and designing the new control law. Starting with the basics of visual serving, we introduce the general control law depending on the camera velocity. Then, in the next section, the control law is tolerated for our problem such that calculating velocities based on the information gathered from the two extracted lines. After that, the geometric features extraction method used in our approach will be introduced as well.

3.1 Visual Servoing Basics

Visual Servoing Control is a methodology that utilizes computer vision data to control the motion of a robot. This approach combines techniques from image processing, computer vision, and control theory to enable precise manipulation and navigation tasks [28]. In our specific context, the visual controller's output is a velocity vector denoted as uc, which contains the velocities required to guide the robot's joints and steer the end-effector towards the desired pose within a given time step [10], [28], [29]. The objective of the visual controller is to generate the velocity vector u_c in a manner that minimizes the error $\rho\ (x\ , x_d)$ between the extracted features x from the current image and the corresponding desired features x_d from a reference (desired) image. The aim is to achieve a velocity vector that effectively reduces this error, enabling accurate

control of the robot's motion. In cases where the feature vector x is a global descriptor, meaning that the image is directly fed into the control law, the Visual Servoing approach is commonly referred to as Direct Visual Servoing (DVS). The general control law used in visual servoing tasks follows the form of

$$\dot{e} = L_e \nu \tag{1}$$

where ν represents the instantaneous camera velocity, and L_e is the interaction matrix that describes the variations of the error e with respect to the velocity ν. In order to achieve exponential convergence of the visual servoing equation, we assume that

$$\dot{e} = -\lambda \mathrm{e} \tag{2}$$

where λ is a positive real number used to tune the convergence speed.

By combining the (1) and (2), and solving for ν, we can derive the classical control law as follows:

$$\nu = -\lambda \widehat{\mathrm{L}_e^+} e \tag{3}$$

Here $\widehat{\mathrm{L}_e^+}$ represents the approximation of the pseudo-inverse of the interaction matrix L_e.

In our study, we simplify the control law by focusing on finding the interaction matrix L_e specifically for the desired visual features, which in this case are the side lines of the tactile paving. By solving for the camera velocity ν using the simplified control law, we aim to achieve accurate and efficient control of the visual servoing system. The interaction matrix L_e plays a crucial role in capturing the relationship between the error in the visual features and the corresponding camera velocity, enabling us to achieve precise control in our visual servoing tasks.

3.2 The Development of The New Control Law Using Sidewalk Edge Features

To achieve tactile paving tracking and facilitate precise wheelchair sidewalk navigation (as described in section II), our main goal is to develop an advanced visual servoing controller. This innovative controller will generate the required data to train

an ML model that is able to control the wheelchair's movement to follow the tactile paving on the sidewalk. The proposed control method will utilize visual information from the tactile paving's sidelines as key geometric features. These features will drive the control mechanism to generate velocity signals, effectively controlling the wheelchair's motion. This will ensure the realization of the sidewalk-following task. Through this proposed visual servoing controller, we aim to generate the data that enhance the wheelchair's navigation, enabling the trained model to smoothly and accurately follow the tactile paving's path for efficient sidewalk traversal.

For deriving the control law we start by assuming side lines are extracted from tactile paving images and ready to be used as geometrical visual features, each of which is represented in polar coordinates using ρ and θ notation. The interaction matrix of the lines feature with respect to ρ and θ is written as follows [30]:

$$L_\rho = \begin{bmatrix} \lambda_\rho cos\theta & \lambda_\rho sin\theta & \lambda_\rho \rho & (1+\rho^2)sin\theta & -(1+\rho^2)cos\theta & 0 \end{bmatrix}$$

$$L_\theta = \begin{bmatrix} \lambda_\theta cos\theta & \lambda_\theta sin\theta & \lambda_\theta \rho & -\rho cos\theta & -\rho sin\theta & -1 \end{bmatrix} \quad (4)$$

Using two lines feature, the above two matrices are concatenated to form a 4-rows matrix of the form:

$$L(\rho,\theta) = \begin{bmatrix} \lambda_\rho cos\theta & \lambda_\rho sin\theta & \lambda_\rho \rho & S_\rho & -C_\rho & 0 \\ \lambda_\theta cos\theta & \lambda_\theta sin\theta & \lambda_\theta \rho & -\rho cos\theta & -\rho sin\theta & -1 \\ \lambda_\rho cos\theta & \lambda_\rho sin\theta & \lambda_\rho \rho & S_\rho & -C_\rho & 0 \\ \lambda_\theta cos\theta & \lambda_\theta sin\theta & \lambda_\theta \rho & -\rho cos\theta & -\rho sin\theta & -1 \end{bmatrix} \quad (5)$$

Where

$$S_\rho = (1+\rho^2)sin\theta$$

$$C_\rho = (1+\rho^2)cos\theta$$

With

$$\lambda_\rho = -A\rho cos\theta - B\rho sin\theta - C \quad (6)$$

And

$$\lambda_\theta = -Asin\theta + Bcos\theta \quad (7)$$

If we assume that the plane that contains the extracted lines has the equation of:

$$A_1X + B_1Y + C_1Z + D_1 = 0 \quad (8)$$

In our study we assume that $A_1 = C_1 = 0$ and $Y = h_1$ so that the plane equation will be: $B_1Y + D_1 = 0$

By solving this Equation for B_1 we will

$$B_1 = -D_1/h_1 \quad (9)$$

And with:

$$A = -A_1/D_1 \ , \ B = -B_1/D_1 \ , \quad C = -C_1/D_1 \quad (10)$$

According to our assumptions; such as $A = 0$, (7) will become:

$$\lambda_\theta = B\cos\theta \quad (11)$$

From Equation (10) we find that $B = -B_1/D_1$, if we substitute B_1 according to (9) such that $B_1 = -D_1/h_1$, Equation (7) becomes

$$\lambda_\theta = \cos\theta/h_1 \quad (12)$$

Similarly, following the same procedure in deriving λ_ρ , starting from our assumptions, Equation (6) becomes:

$$\lambda_\rho = -B\rho\sin\theta \quad (13)$$

where $B = -B_1/D_1$ and $B_1 = -D_1/h_1$.

Substituting B and B_1 values in (6) will give λ_ρ as:

$$\lambda_\rho = (-\rho/h_1)\sin\theta \quad (14)$$

The camera velocity (V_c) is derived from robot velocity in a robot frame (u) as follows:

$$V_c = T_r u \quad (15)$$

where u refers to wheelchair velocity and it is defined as

$$u = [v_r \ \omega_r]$$

And camera velocity is defined as:

$$V_c = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

T_r denotes the camera-robot velocity transformation and it is given as:

$$T_r = \begin{bmatrix} 0 & -l \\ 0 & 0 \\ 1 & -\omega \\ 0 & 0 \\ 0 & -1 \\ 0 & 0 \end{bmatrix} = [T_v \; T_\omega]$$

Considering our features in this study to be 2D lines, and starting from the general control law in visual servoing IBVS:

$$\dot{s} = L_s V_c \tag{16}$$

Where s is visual features, L_s is the image interaction matrix corresponding to the features s, and V_c is the camera velocity. Equation (15) can be written in more detailed form as:

$$\begin{bmatrix} v_c \\ \omega_c \end{bmatrix} = [T_v \; T_\omega] \begin{bmatrix} v_r \\ \omega_r \end{bmatrix}$$

$$V_c = [v_c \; \omega_c] = [T_v v_r + T_\omega \omega_r]$$

Using the interaction matrix $L(\rho, \theta)$ in Equation (5) and velocity matrix V_c, the visual servoing control low in (16) can be written for two visual lines (four visual features (ρ_1, θ_1, ρ_1, θ_2)):

$$\dot{s} = \begin{bmatrix} L_{\rho_1} \\ L_{\theta_1} \\ L_{\rho_2} \\ L_{\theta_2} \end{bmatrix} [T_v] v_r + \begin{bmatrix} L_{\rho_1} \\ L_{\theta_1} \\ L_{\rho_2} \\ L_{\theta_2} \end{bmatrix} [T_\omega] \omega_r \tag{17}$$

If we define Jacobian matrices J_v and J_ω as follows:

$$J_v = \begin{bmatrix} L_{\rho_1} \\ L_{\theta_1} \\ L_{\rho_2} \\ L_{\theta_2} \end{bmatrix} [T_v] = \begin{bmatrix} -\lambda_{\rho_1}\rho_1 \\ -\lambda_{\theta_1}\rho_1 \\ -\lambda_{\rho_2}\rho_2 \\ -\lambda_{\theta_2}\rho_2 \end{bmatrix} \tag{18}$$

And

$$J_\omega = \begin{bmatrix} L_{\rho_1} \\ L_{\theta_1} \\ L_{\rho_2} \\ L_{\theta_2} \end{bmatrix} [T_\omega] = \begin{bmatrix} -l\lambda_{\rho_1} cos\theta_1 + \omega\lambda_{\rho_1}\rho_1 + (1 + {\rho_1}^2) \\ -l\lambda_{\rho_1} cos\theta_1 + \omega\lambda_{\rho_1}\rho_1 + \rho_1 sin\,\theta_1 \\ -l\lambda_{\rho_2} cos\theta_2 + \omega\lambda_{\rho_2}\rho_2 + (1 + {\rho_2}^2 cos\,\theta_2 \\ -l\lambda_{\rho_2} cos\theta_2 + \omega\lambda_{\rho_2}\rho_2 + \rho_2 sin\,\theta_2 \end{bmatrix} \tag{19}$$

Equation (17) will have the form:

$$\dot{s} = J_v\, v_r + J_\omega\, \omega_r \tag{20}$$

where ω_r is the rotational velocity and v_rr is the transnational velocity of the wheelchair. Assuming the transnational velocity to be constant such that $v_r = v^*$= const, solving the control law in Equation (20) for ω_r will give:

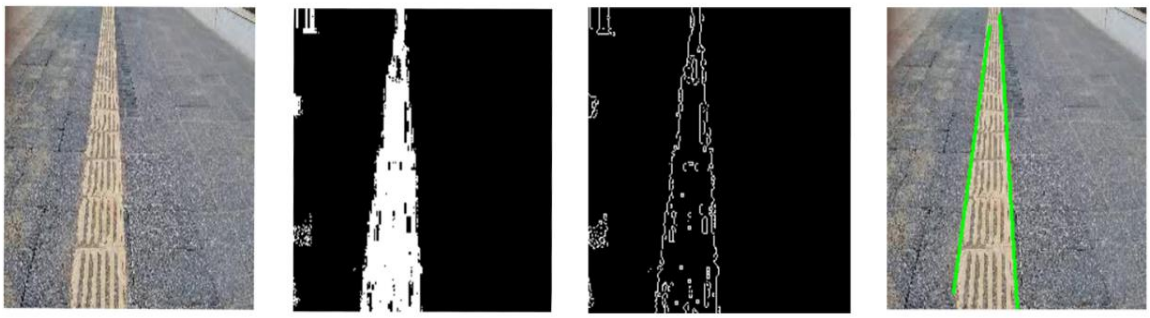
$$\omega_r = -J_\omega^+(\,\lambda e + J_v\, v^*\,) \tag{21}$$

where J_v and J_ω are given in (18) and (19) respectively. To ensure the accuracy of the generated angular velocities we need robust features to be used in the control law. In section III-C we introduce the methods and algorithms used for robustly extract the two lines features from tactile paving.

3.3 Robust Feature Extraction From Images

Crafting a precise control law begins with extracting robust geometric features from tactile paving images. This multi-stage process involves Color Segmentation to isolate pertinent regions based on color, followed by Edge Detection for identifying edges and boundaries. The third step, Line Extraction, analyzes edges for coherent line features. Using color segmentation, we partitioned the image into sections, distinguishing the tactile paving's yellow section. These steps enhance our methodology's success in crafting an effective control law.

Having successfully separated the tactile paving from other elements in the image, our next step focused on accurately detecting the edges and lines within the tactile paving region. To accomplish this, we utilized two essential algorithms: Canny edge detection and Hough Transform. Firstly, we converted the image into grayscale, a crucial preprocessing step that facilitates subsequent processing stages. By working with a grayscale image, we eliminated color variations and focused solely on intensity values, thereby enhancing the performance of edge and line extraction algorithms. Subsequently, we applied the Canny edge detection algorithm, which leverages intensity gradients across the grayscale image to identify significant changes in pixel values, thereby highlighting the edges within the tactile paving region. The Canny edge detection algorithm effectively filters out noise and emphasizes the salient edges. After the edge detection stage, we employed the Hough Transform algorithm to identify all possible straight lines within the image. This algorithm transforms the image into a parameter space representation, where lines can be identified and extracted based on a voting-based approach. By accumulating votes in the transformed space, the Hough Transform algorithm identifies the prominent linear structures present within the image. In the final stage of our approach, we examined the position coordinates of the detected lines to determine the two farthest sidelines, which represent the most significant edges within the dataset images. Extracting these prominent lines provided crucial geometric information about tactile paving, which is vital for subsequent analysis and control tasks. For a more comprehensive visual understanding of the application of the color segmentation, Canny edge detection, and Hough Transform algorithms on dataset images for obtaining the annotated image, Figure 9 provides clear depictions of the outcomes generated by these algorithms, effectively highlighting the steps followed to extract the sidelines from the dataset images.



(a) (b) (c) (d)

Figure 9. The process of extracting two lines features from dataset images involves several steps: (a) displaying the original image, (b) showing the image after applying color segmentation, (c) demonstrating the image with edges extracted using the Canny Edge detection technique, and (d) presenting the annotated dataset images with the extracted sidelines.

CHAPTER 4
DATA COLLECTION AND TRAINING THE MACHINE LEARNING MODEL

4.1 Image Collection And Annotation For Training

The dataset utilized to train and test the ML model was meticulously curated to reflect the real-world complexity of tactile paving structures found on sidewalks. This dataset was compiled from an assortment of ten distinct videos, each offering diverse viewpoints of these tactile paving structures. These videos, captured using mobile phones, were chosen deliberately to encapsulate a range of scenarios, thereby enhancing the robustness and generalization capabilities of the model.

To facilitate compatibility with the subsequent ML model, the videos underwent a pre-processing stage. They were segmented into individual frames, resulting in a comprehensive dataset encompassing a total of 5568 images. These images were subjected to a standardization process, wherein they were resized to a uniform dimension of 240 by 240 pixels. Importantly, this comprehensive dataset encompasses both the training and testing data, ensuring a holistic representation of the tactile paving structures and their corresponding angular velocities. Strategically partitioning the dataset was crucial for the training and evaluation phases. To this end, a meticulous allocation was established, where 90% of the images were designated for training purposes, while the remaining 10% were reserved for testing. This partitioning strategy sought to strike a balance between providing an ample amount of data for robust model learning and affording an independent dataset for rigorous performance assessment. The visual depiction of the collected dataset examples is showcased in Figure 10, offering a tangible glimpse into the diversity and complexity of the data streamlining the model's training and testing processes. The process of extracting the dual side lines

from each image necessitated the development and application of a specialized algorithm. This algorithm is designed through a combination of edge detection techniques, and the Hough Transform enabled the accurate isolation and extraction of the pertinent geometric features. Further elucidation regarding this intricate extraction process can be found in Section 3.3, where a comprehensive breakdown of the employed methodologies is presented.

Figure 10. Images samples of the dataset used in training and testing the ML model

Once the sidelines were successfully extracted, the subsequent phase involved computing the corresponding angular velocities for each image. This entailed the utilization of the control law derived in Section 3.2, establishing a mathematical framework that establishes the relationship between the extracted features and the desired velocities. This translation of visual cues into control signals underpins the fundamental principle driving the model's predictive capabilities.

With the dataset prepared and the angular velocities derived, the model's training journey commenced. This iterative optimization process revolved around training the selected model. Here, we selected the GP model for its superior performance and lightweight compared to deep learning models. The GP model and its training are described in the next section. A diagram shows the steps followed in dataset collection and training the model is represented in Figure 11.

Having successfully completed the training phase, the model's efficacy was subject to rigorous evaluation using the reserved testing dataset. This assessment encompassed the calculation of various performance metrics, including the Mean Square Error (MSE), Absolute Mean Square Error (AMSE), and R-score. These metrics, instrumental in gauging the precision and accuracy of the model's predictions in comparison to the ground truth velocities, collectively underscored the model's proficiency. Impressively, the testing process yielded a mean square error and absolute mean square error of 0.2836, accompanied by an R-score of 0.718. These outcomes substantiated the model's prowess in approximating the desired velocities. The meticulous evaluation and validation undertaken not only affirmed the model's robust performance within the context of the curated dataset but also solidified its readiness for subsequent experimental endeavors. The culmination of this rigorous process positions the model as an indispensable tool for conducting intricate experiments, as detailed in chapter 5.

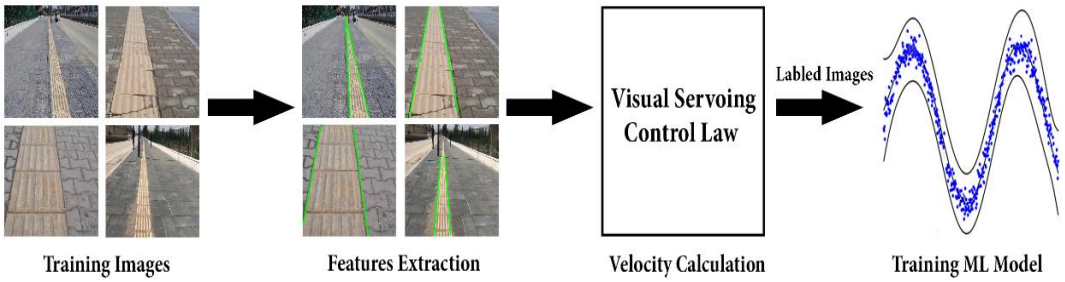


Figure 11. Diagram shows the steps done in the dataset preparing and model training

4.2 Model Selection And Training

In order to identify the most appropriate machine learning model for our research, a thorough comparative analysis was conducted, considering the findings of other related works in the field. Among these notable references was the study presented at [31], which utilized the ResNet CNN model. Another study reported in [32], has implemented a Gaussian Process (GP) model for corridor following. It is worth mentioning that several GP successful applications have been reported in the literature [33]–[35]. Mathematically, GPs can be defined as a collection of random variables that follow a joint Gaussian distribution [36]–[38] . A GP is fully specified by a normal distribution denoted as N_{GP}, expressed as follows:

$$\mathrm{f(x)} = \mathrm{N_{GP}}(m(x), k(x, x')) \tag{22}$$

where $m(x)$ and $k(x, x')$ are mean and covariance functions of the GP respectively, x and x' are the feature vectors that represent two different input images respectively. Using the Squared Exponential (SE) kernel function, the covariance will have the form

$$k(x, x') = \sigma_{\omega}^2 \exp(-\frac{1}{2\,l^2}\,(x - x')^2) + \sigma_n^2 \delta \tag{23}$$

where σ_{ω}^2 and σ_n^2 are signal variance and Gaussian noise variance respectively, l is length scale and δ is a Kronecker delta which equals one when x = x′ and zero otherwise. Here σ_{ω}^2 , σ_n^2 , and l are the hyperparameters of the covariance function.

The GP model employs a systematic training process. The training data is meticulously organized, consisting of pairs that associate input image features with corresponding noisy angular velocity measurements. Leveraging the covariance function's hyperparameters, namely σ_{ω}^2 , σ_n^2 , and l, the GP model learns the underlying relationships between the input features and the noisy angular velocities. This learning process involves iteratively optimizing the hyperparameters to maximize the model's fit to the provided training data. Starting from the control law that utilizes features from pairs of lines, the GP model uses the velocities $\widehat{\omega}(x)$ computed by the control law that employs the pseudo-inverse of the Jacobian matrix. This inversion yields a predictive formula that encapsulates the mapping from image features to angular velocity estimates.

Our task is to predict the angular velocity command $\widehat{\omega}(x)$ to be produced by the visual controller by using image features as input. Since visual information are noisy in nature, we do not have access to the function values themselves but only noisy versions of them, so we assume that

$$\omega_y = \widehat{\omega}(x) + \epsilon \tag{24}$$

where ω_y is the noisy observations of the angular velocity around the y-axis, and ϵ denotes Gaussian noise with variance σ_n^2. The function values to be predicted by the GP function are denoted by $\widehat{\omega}(x)$.

During the training phase of the Gaussian Process (GP), it is crucial to optimize the hyperparameters (σ_{ω}^2, σ_n^2) and l to ensure accurate model performance. This optimization process is accomplished through maximum likelihood estimation, utilizing the SLSQP algorithm, which stands for sequential least square programming. We integrated the SLSQP algorithm into the Scikit-learn Python library to facilitate this optimization procedure.

4.3 Fuzzy Control of Wheelchair Motion

In this chapter, we focus on the development and implementation of a fuzzy control system for controlling wheelchair motion, specifically designed to take advantage of visual feedback obtained from a Raspberry Pi camera. The aim is to create a responsive and adaptable wheelchair navigation system that can react to the dynamic visual environment to ensure robust and efficient mobility of the wheelchair. The core of our wheelchair control system lies in the utilization of a fuzzy control type controller. Fuzzy control enables us to handle imprecise input data and make decisions based on degrees of truth rather than binary true/false values. In the context of wheelchair motion control, this approach allows us to smoothly and intuitively respond to the varying visual data captured by the Raspberry Pi camera. Figure 12 visualiz the fozzy controller used in our work. In our system the fuzzy controller takes the velocity value predicted by the GP model according to the caputred image from the camera as input and determines the appropriate steering command. That is by defining different ranges such as "right," "direct" and "left". The control logic will assign streeing commands to the defined ranges based on the measured velocities using the following method :

- If the velocity falls within a predefined range of "left," the wheelchair is commanded to move left.
- Conversely, if the velocity falls within a predefined range of "right," the wheelchair is commanded to move right.

- For velocities in between, the controller interpolates the commands and instruct the wheelchair to move forward allowing for smooth and gradual steering adjustments.

The fuzzy control logic, allows for responsive and dynamic control, ensuring safe and efficient navigation in real-world environments. In the subsequent chapter, we will present a block diagram of the whole system including fuzzy control part.

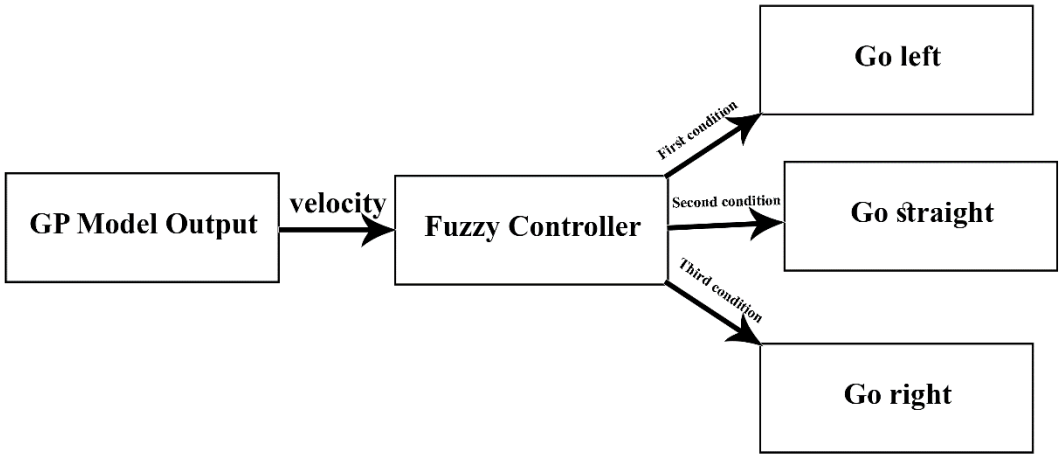


Figure 12. Fuzzy control logic used in our work

4.4 Block Diagram Representation of Control System

In this subsection, we present a detailed explanation of the block diagram representing the control system architecture for our visual servoing-based wheelchair navigation system. The architecture consists of several interconnected components, each playing a vital role in ensuring safe and efficient navigation. Figure 13 represents the block diagram of the closed loop control system used in our work.

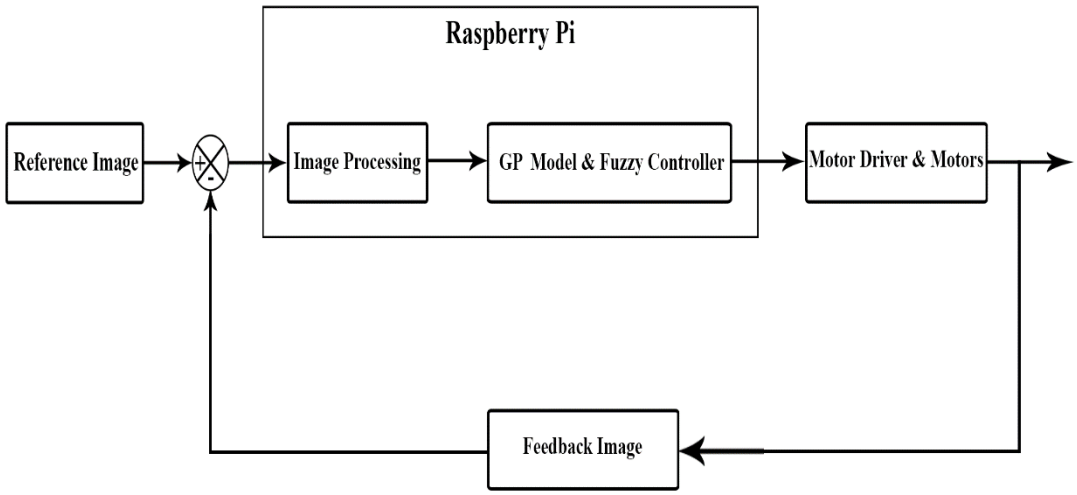


Figure 13. Block diagram of the closed loop control system

The functions and interactions of each block in the diagram can be explained as follows:

4.4.1 Raspberry Pi Camera

The Raspberry Pi Camera serves as the system's primary sensor, capturing images of the sidewalk environment. Its high-resolution capabilities and compact design make it an ideal choice for real-time image acquisition. The camera is mounted on the wheelchair to provide a continuous stream of visual data, which is the foundation of our control system.

4.4.2 Image Processing Unit

The Image Processing Unit represents the first step of the control system. It receives the image data from the Raspberry Pi Camera which is the sidewalk following inclusing tactile paving, and processes it to extract critical the excat possition of the tactile paving in the image. This unit employs a range of image processing techniques, including color segmentation , feature extraction, object recognition, and hough lines detection to identify the tactile paving on the sidewalk. The resulting processed image serves as input for subsequent control blocks.

4.4.3 Fuzzy Controller and GP Model

Within this block, we employ two significant components: a trained Gaussian Process (GP) Model and Fuzzy Controller. The GP Model plays a pivotal role in mapping image data to control actions. It is trained on large dataset of the tactile paving pictures, such that, it can recognize the desired image and provide corresponding steering velocity to the fuzzy controller according to the features extarted from input image in the previous block. The Fuzzy Controller then transfer the velocity into an appropriate steering command for wheelcair motors according to previously defined mutible ranges of velocities. Doing that, fuzzy controller ensure wheelchair movement in the desired direction. These two components work in tandem to make real-time decisions based on image analysis and tarined GP model.

4.4.4 Motor Driver and Wheelchair Motors

The Motor Driver and Wheelchair Motors block represents the hardware responsible for translating the generated movement signals - provided in the previous block - into physical wheelchair movements. Motor drivers are used to control the speed and direction of the motors, enabling the wheelchair to move left, right, or straight, as dictated by the control signals. Precise control of the motors is essential to ensure the safety and accuracy of the navigation system. That is achieved by using fuzzy controller in conjunction with the trained GP model.

4.4.5 Feedback Loop

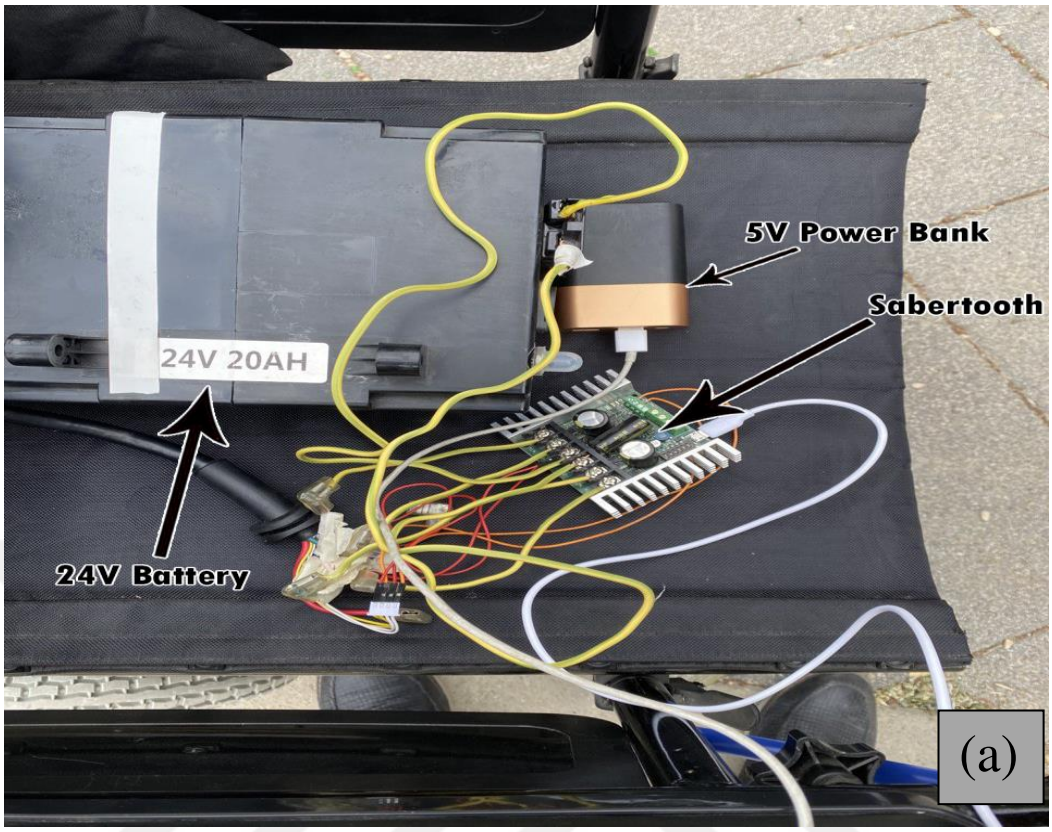
In the control system, the feedback loop is an essential component that closes the control loop. It facilitates the adjustment of the image analysis and control signals based on the current state of the wheelchair and the environment. After every movement of the wheelchair, a new picture of the enironment will be taken form the camera. The picture will be compared to the desired image and processed. Then, the corresponding control signals will be sent to wheelchair motors to adjust its continuous movement and navigation. This feedback mechanism ensures that the system can adapt to unexpected obstacles, changes in the environment, or deviations from the desired path. It plays a crucial role in maintaining system robustness and safety navigation of the wheelchair on the sidewalk.

CHAPTER 5
EXPERIMENTAL EVALUATION

This section introduces performance evaluation metrics for assessing the GP model's effectiveness. These metrics play a vital role in quantitatively analyzing its performance. After presenting these metrics, we evaluate the model's performance on the original image set using them. To validate the model's robustness and practicality, real wheelchair experiments are conducted. These tests examine the model's response to interventions by a human operator, offering insights into its behavior in real-world scenarios. Comparing our model to a classical control method that uses two line features for direct velocity extraction follows. This contrast highlights the relative advantages of our approach and identifies areas for potential enhancement, allowing us to make informed comparisons and improvements.

5.1 Experimental Setup

This subsection serves the purpose of providing a concise introduction to the hardware components employed in the execution of experiments involving a real-world wheelchair. Primarily, two distinct power sources were deployed for this purpose: a 24V DC battery was employed to supply power to both the wheelchair's motor and the Sabertooth motor driver. Additionally, a separate 5V DC power source was utilized to energize the Raspberry Pi module and its accompanying camera. Furthermore, the Sabertooth motor driver was used to control the movement of the motors of the wheelchair. Communication between the Sabertooth module and the Raspberry Pi was established via serial communication protocols. For a more comprehensive understanding of the interconnections and the physical positioning of these components on the actual wheelchair, refer to Figure 14.


5V Power Bank
Sabertooth
24V 20AH
24V Battery
(a)

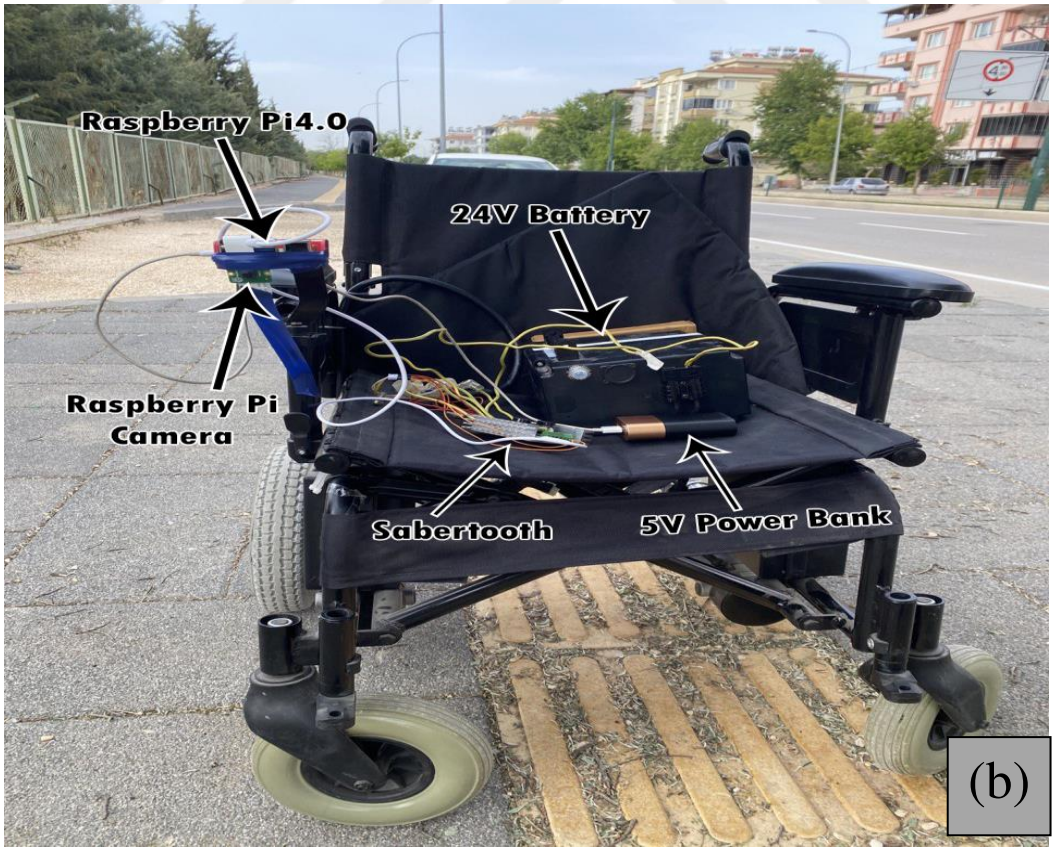

Raspberry Pi4.0
24V Battery
Raspberry Pi
Camera
Sabertooth
5V Power Bank
(b)

Figure 14. Figures include the hardware components used during experiments. (a) reprents top view of the wheelciar and (b) represents front view of it.

5.2 Analysis of GP-Model Accuracy

To thoroughly assess the proposed GP model, we conducted an experiment comparing its output velocities with the ground truth velocities used in training. This analysis gauges the accuracy and reliability of the GP model's predictions. The experiment details the setup, methodology, and data visualization to elucidate the evaluation process. Examining the experiment's results carefully reveals further explanations and visualization, presented in the following.

Estimated Velocities vs. Ground Truth Velocities:

To assess the GP model's predicted velocities, we compared them with those calculated using the control law in Section 3.2. This law utilizes tactile paving's sidelines as inputs and yields accurate angular velocities. Applying both methods to the same images, we compared their output velocities in Figure 15. Both methods correctly estimated motion direction, with minor velocity differences. This demonstrates the GP model's accuracy in estimating motion direction and its ability to provide comparable velocity estimates to the control law. These findings highlight the GP model's potential as a reliable controller for visual servoing tasks.

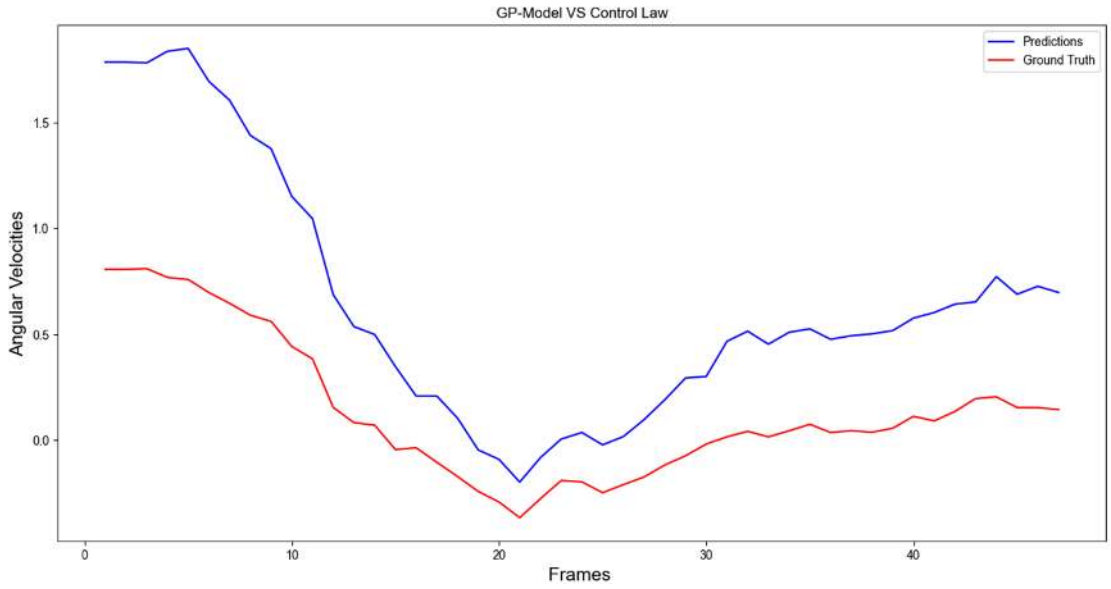


Figure 15. Comparison between the angular velocities generated from two different methods when applied to the same set of tactile paving images. Blue line refers to velocities generated from the GP model and red line

represents the velocities generated directly from the control law (Ground Truth)

5.3 Experiments With Real Wheelchair Setup

Real experiments were conducted on the Gaziantep University adjacent sidewalk to validate our approach. Using a Raspberry Pi 4 with a camera module, live sidewalk images were captured and processed by the GP model to predict angular velocities. The Sabertooth motor driver facilitated wheelchair movement based on velocity signals from the Raspberry Pi. This entire process runs at about 7 Hz, taking around 143 ms per motion cycle. This frequency ensures prompt interaction among system components, ensuring efficient wheelchair movement.

5.3.1 Start From Arbitrary Position

To comprehensively study the wheelchair's motion in diverse conditions, we executed experiments from different sidewalk starting points, ensuring continuous camera visibility of the tactile paving. We explored three camera views: 1) Paving on the left. 2) Paving centered. 3) Paving on the right. In each case, the paving stayed in the camera's view, essential for wheelchair movement.

In the first scenario with tactile paving on the left, the wheelchair shifts left to align with the paving, reaching the image center. The wheelchair then maintains this position, continuously tracking the paving's trajectory until completion or loss of visibility. Refer to Figure 16 for a graphical depiction of the wheelchair's angular motion indicated by angular velocity. An experiment was performed with the wheelchair starting at the center of the tactile paving to observe its motion. In this setup, the wheelchair displayed smooth and uninterrupted movement without abrupt starts. It moved forward while closely following the tactile paving until its completion or loss of camera visibility. Refer to Figure 17 for the angular motion representation. We also analyzed the scenario where the initial position of the tactile paving was to the right of the image. Here, the wheelchair's behavior was analogous to the first case, but with reversed direction. It moved to align with the paving, bringing it to the center of the camera's view. Once at the center, the wheelchair maintained its position, tracking the paving's trajectory until completion or loss of camera visibility. See

Figure 18 for the angular motion. These experiments provided valuable insights into the wheelchair's motion and demonstrated its ability to navigate and track the tactile paving under different initial conditions.

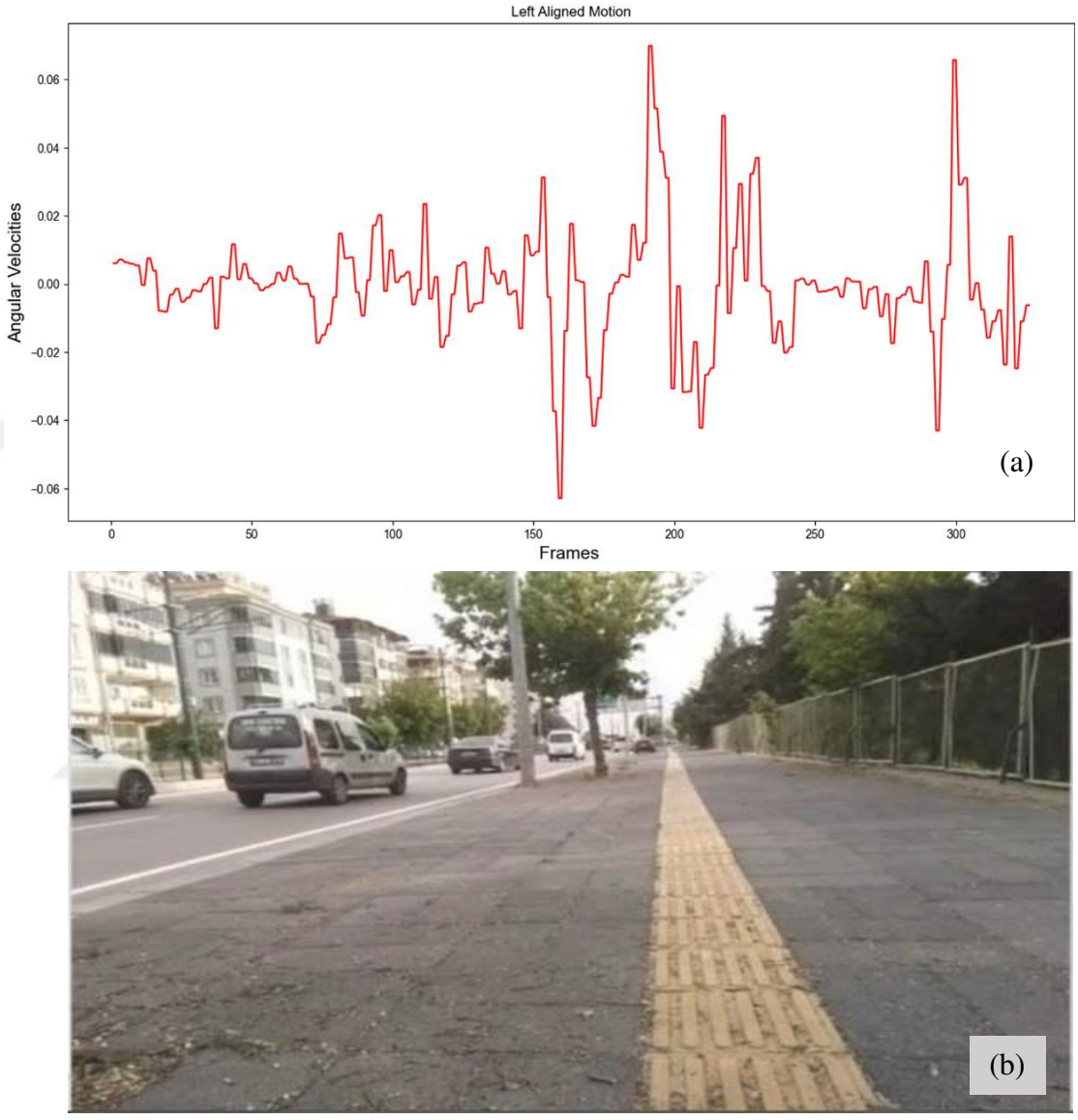


Figure 16. (a) Shows angular velocities generated from the GP model in the case when the wheelchair is located to the left of the tactile paving. (b) Shows the initial wheelchair position with respect to the tactile paving on the sidewalk.

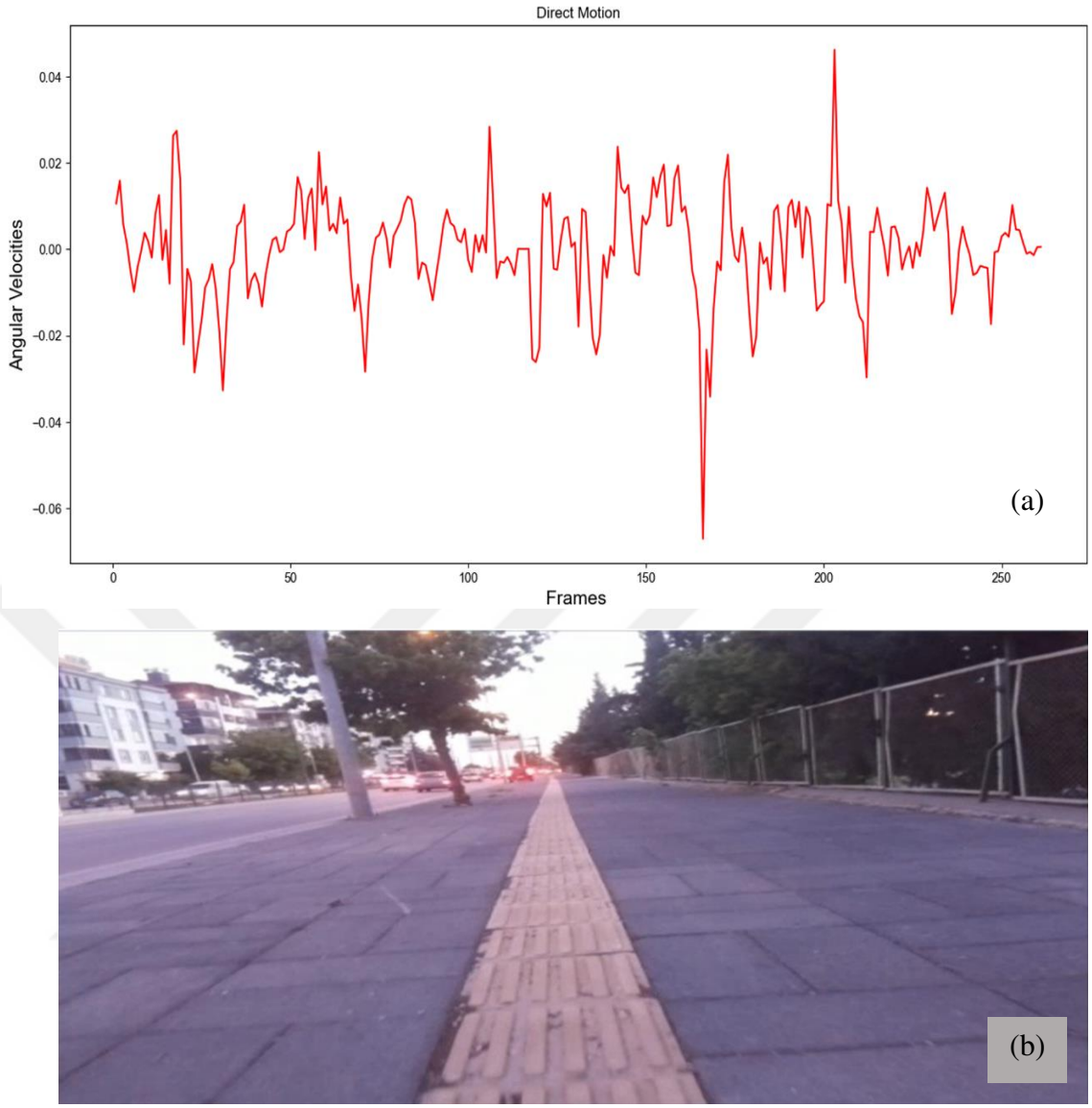


Figure 17. (a) represents Angular velocities generated from the GP model in the case when the wheelchair is initially located directly with the tactile paving. (b) shows the initial wheelchair position with respect to the tactile paving on the sidewalk.

Figure 18. (a) shows angular velocities generated from the GP model in the case when the wheelchair is initially located to the right of the tactile paving. (b) shows the initial wheelchair position with respect to the tactile paving on the sidewalk.

5.3.2 Wheelchair motion behavior with obstacle intervention

Another experiments were made to further investigate the behavior of the wheelchair in the presence of obstacles obstructing the tactile paving. These experiments aimed to examine how the wheelchair responded when the tactile paving disappeared from the camera's field of view due to the presence of an obstacle. In cases where an obstacle temporarily obscured the tactile paving from the camera, the wheelchair halted, waiting for the obstacle to clear or the paving to reappear. Once the obstacle was resolved, the wheelchair automatically resumed its path, smoothly following the paving. The stopping mechanism relies on the GP model behavior; it generates extremely low velocity when no paving is detected, causing the motor controller to halt the motors. This calibration effectively stops the wheelchair in these situations. Multiple experiments tested this scenario, similar to previous tests on a real wheelchair, covering three initial paving positions as discussed in Subsection 5.3.1. Experimental outcomes for the scenario with the wheelchair aligned left of the paving are visually presented in Figure 19. For a comprehensive view of the experiments, we included frames from the Raspberry Pi camera capturing obstacles blocking the tactile paving's path. Corresponding wheelchair velocity responses were recorded in the Raspberry Pi terminal. See Figures 20 and 21 for these visual representations. These experiments illustrate how the wheelchair behaves when confronted with obstacles, demonstrating its ability to briefly pause and then resume following the tactile paving after obstacle removal. These outcomes offer valuable insights into the wheelchair's adaptability and responsiveness in real-world situations where the tactile paving may encounter intermittent obstructions.

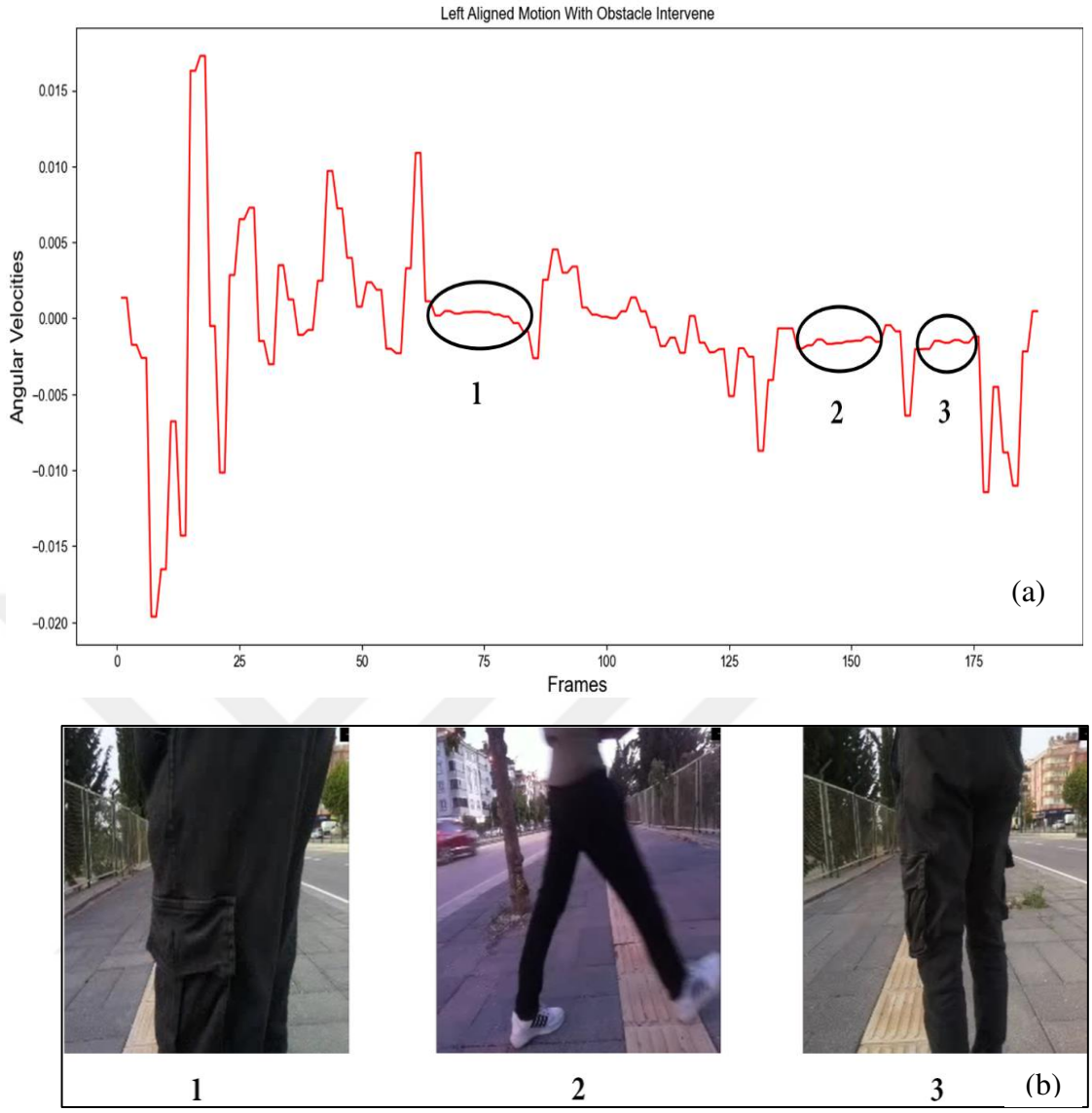


Figure 19. Generated angular velocities when an obstacle intervenes the camera field of view or the tactile paving is not visible, where (a) represents generated angular velocity values and (b) shows frame samples corresponding to the angular velocities when an obstacle intervene.

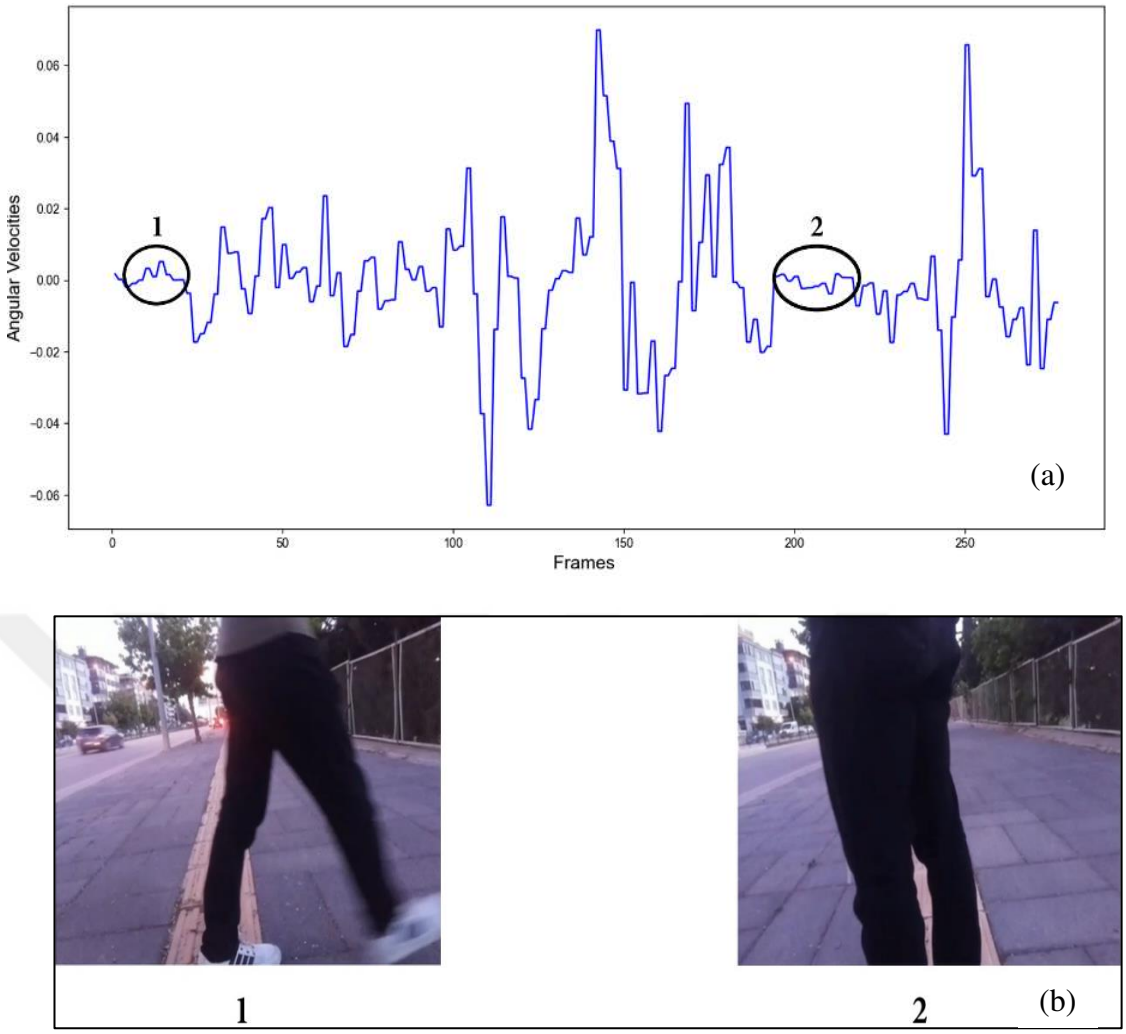


Figure 20. A sample data of the generated angular velocities, when an obstacle intervenes the camera field of view or the tactile paving, is not visible, where (a) represents generated angular velocity values, and (b) shows frame samples corresponding to the angular velocities when an obstacle intervene.

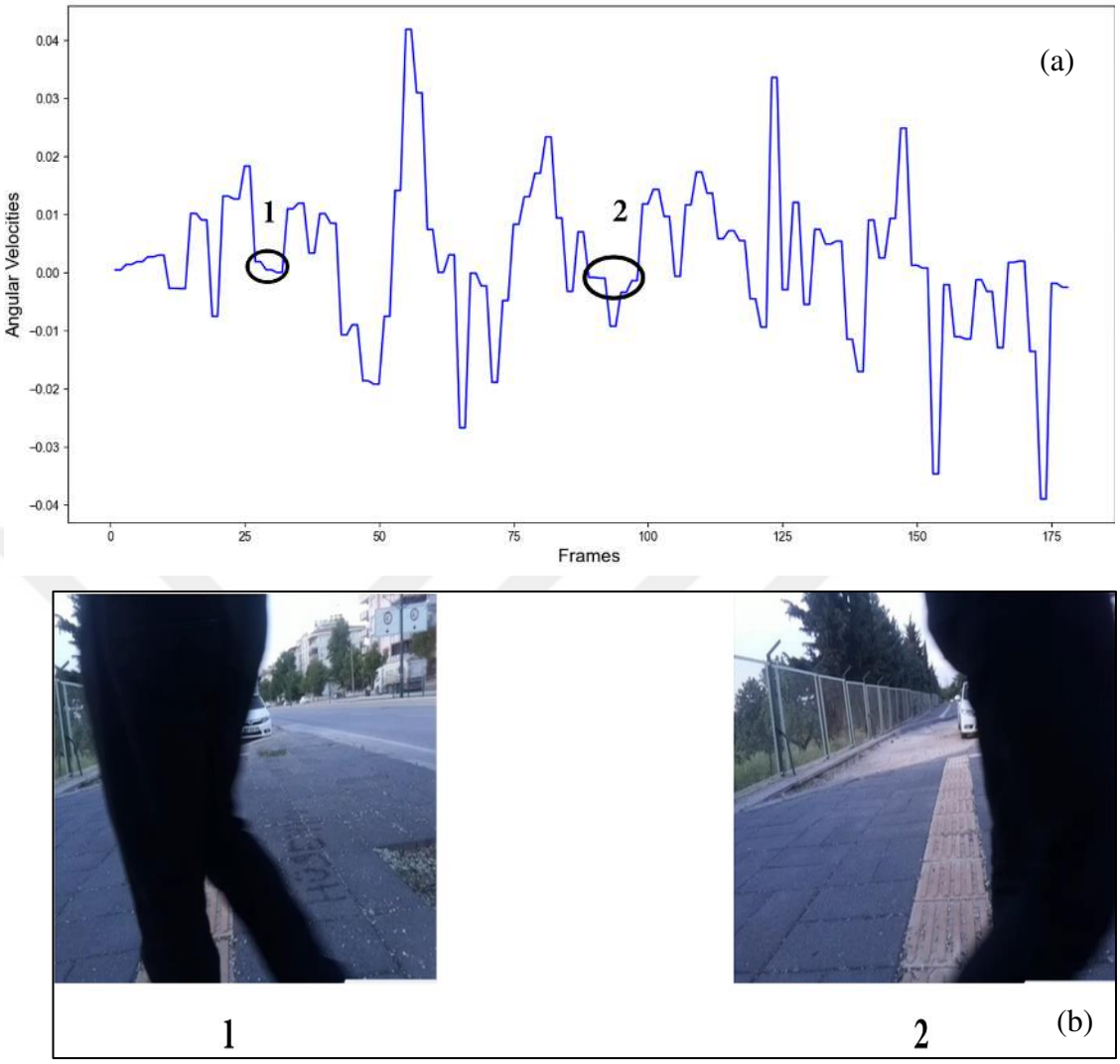


Figure 21. Another experiment result data of the generated angular velocities when an obstacle intervene the camera field of view or the tactile paving is not visible, where (a) represents generated angular velocity values, and (b) shows frame samples corresponding to the angular velocities when an obstacle intervene.

5.3.3 Real time comparison between GP model and classical control law

To assess the robustness of the proposed GP model, we conducted a comparative analysis between its performance and that of the classical control law discussed in Section 3.2. The aim was to show the GP model's ability and advantages in generating accurate velocity signals for the wheelchair as it navigates a sidewalk. First, we tested the GP model by employing it to predict the angular velocities necessary for the wheelchair to follow the desired path. Subsequently, we applied the classical control law to generate velocity signals for the wheelchair, enabling it to follow the same sidewalk. By comparing the angular velocities generated by the two methods, we could compare the effectiveness of the GP model in relation to the traditional control approach. The obtained results are visually presented in Figures 22 and 23. These figures provide a clear depiction of the performance disparity between the two methods. From our analysis, it can be concluded that the proposed GP model demonstrated superior stability and robustness compared to the classical control law. The GP model exhibited a higher level of accuracy and precision in generating the required angular velocities for the wheelchair, leading to more reliable motion control. This finding underscores the potential of the GP model as an effective and robust control mechanism for wheelchair navigation on sidewalks.

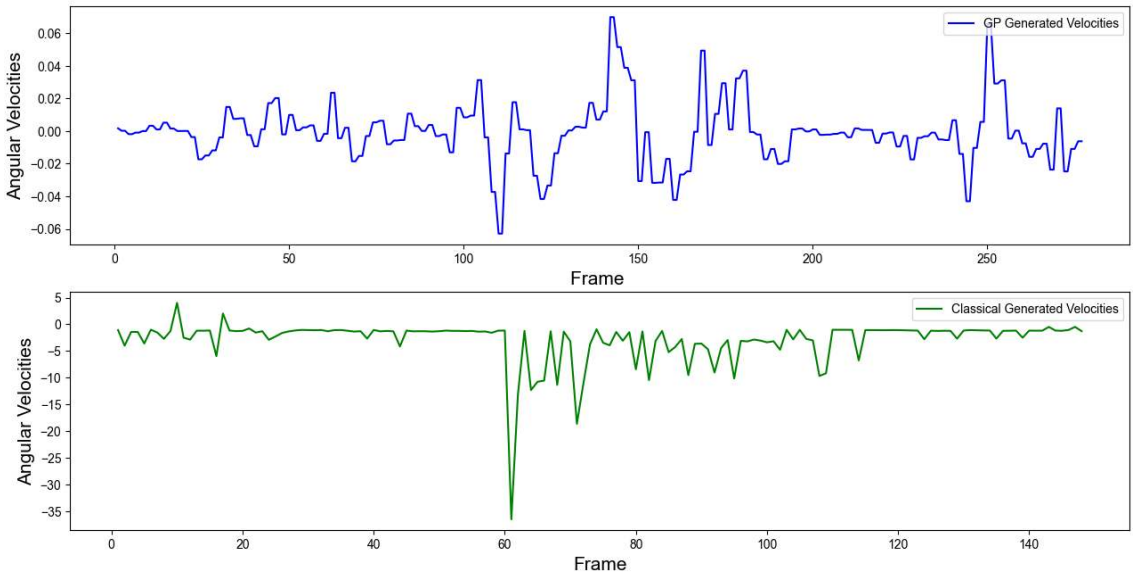


Figure 22. Experimental results of the comparison between GP model and classical control law in generating angular velocities from the same set of images.

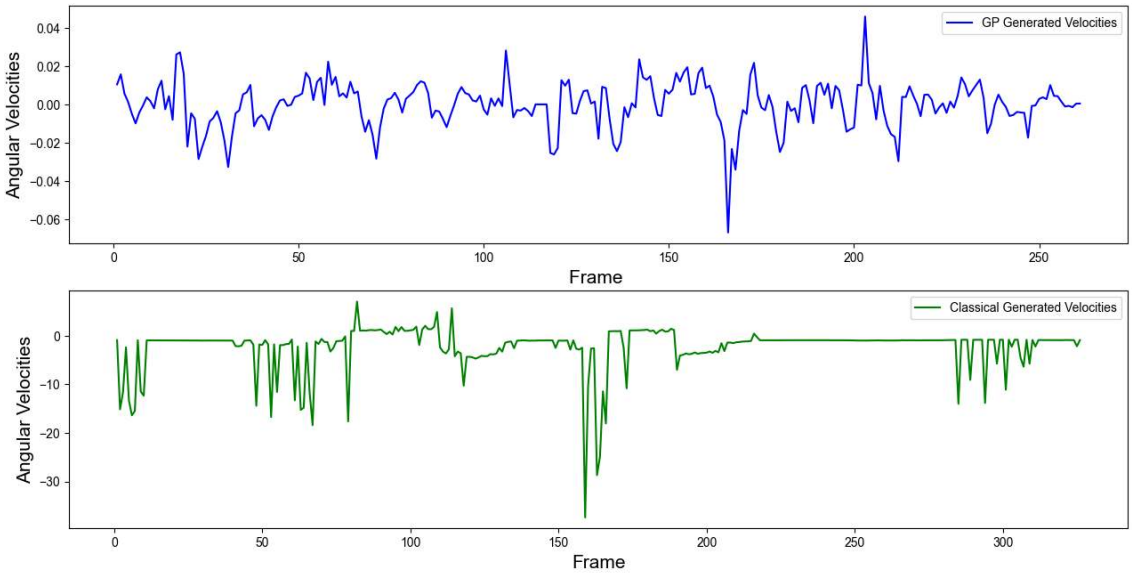


Figure 23. More experimental results of the comparison between GP model and classical control law in generating angular velocities from the same set of images.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In conclusion, this study effectively demonstrates the GP-based visual controller's prowess in sidewalk following tasks. By utilizing extracted lines from image processing as input and training the GP to generate velocity signals, our approach showcases robustness and real-time performance in various real-world scenarios. Conducted experiments validate our system's capabilities, including obstacle detection and intervention, and aligning the wheelchair with the sidewalk, even from the center. Promising future research could involve using larger, diverse datasets for further improvements. Such advances will contribute to system development and optimization, ultimately enhancing wheelchair users' mobility and independence. The conducted experiments validate the capabilities of our system, such as obstacle detection and intervention, as well as aligning the wheelchair with the sidewalk, even when starting from the center. As a promising avenue for future research, we anticipate that further improvements can be achieved by employing larger and more diverse datasets. These advancements will contribute to the continued development and optimization of such systems, ultimately enhancing the mobility and independence of wheelchair users.

REFERENCES

[1] M. Finlayson, T. Van Denend, Experiencing the loss of mobility: perspectives of older adults with ms, Disability and rehabilitation 25 (20) (2003) 1168–1180.

[2] L. I. Iezzoni, E. P. McCarthy, R. B. Davis, H. Siebens, Mobility difficulties are not only a problem of old age, Journal of general internal medicine 16 (2001) 235–243.

[3] A. Murarka, S. Gulati, P. Beeson, B. Kuipers, Towards a safe, lowcost, intelligent wheelchair, in: Workshop on Planning, Perception and Navigation for Intelligent Vehicles (PPNIV), Citeseer, 2009, pp. 42–50.

[4] M. Weiss, S. Chamorro, R. Girgis, M. Luck, S. E. Kahou, J. P. Cohen, D. Nowrouzezahrai, D. Precup, F. Golemo, C. Pal, Navigation agents for the visually impaired: A sidewalk simulator and experiments, in: Conference on Robot Learning, PMLR, 2020, pp. 1314–1327.

[5] H. S. Kaye, T. Kang, M. P. LaPlante, Mobility device use in the United States, Vol. 14, National Institute on Disability and Rehabilitation Research, US Department . . . , 2000.

[6] A. Mihailidis, P. Elinas, J. Boger, J. Hoey, An intelligent powered wheelchair to enable mobility of cognitively impaired older adults: An anticollision system, IEEE Transactions on Neural Systems and Rehabilitation Engineering 15 (1) (2007) 136–143.

[7] A. Thotakuri, T. Kalyani, M. Vucha, M. Chinnaaiah, T. Nagarjuna, Survey on robot vision: techniques, tools and methodologies, International Journal of Applied Engineering Research 12 (17) (2017) 6887–6896.

[8] L. Zeinalova, B. Jafarov, Mobile robot navigation with preferencebased fuzzy behaviors, in: 11th International Conference on Theory and Application of Soft Computing, Computing with Words and Perceptions and Artificial Intelligence-ICSCCW-2021 11, Springer, 2022, pp. 774– 782.

[9] L. D. Fufa, E. Ayenew, Trajectory tracking of a two-wheeled mobile robot using backstepping and nonlinear pid controller, in: International Conference on Advances of Science and Technology, Springer, 2022, pp. 290–304.

[10] A. H. A. Hafez, C. V. Jawahar, Visual servoing by optimization of a 2d/3d hybrid objective function, in: Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007, pp. 1691–1696 doi:10.1109/ROBOT.2007.363566.

[11] I. T. Lakmal, K. N. Perera, H. H. Y. Sarathchandra, C. Premachandra, Slam-based autonomous indoor navigation system for electric wheelchairs, in: 2020 International Conference on Image Processing and Robotics (ICIP), IEEE, 2020, pp. 1–6.

[12] M. Wen, Y. Dai, T. Chen, C. Zhao, J. Zhang, D. Wang, A robust sidewalk navigation method for mobile robots based on sparse semantic point cloud, in: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2022, pp. 7841–7846.

[13] A. Panah, H. Motameni, A. Ebrahimnejad, An efficient computational hybrid filter to the slam problem for an autonomous wheeled mobile robot, International Journal of Control, Automation and Systems 19 (2021) 3533–3542.

[14] M. Sorokin, J. Tan, C. K. Liu, S. Ha, Learning to navigate sidewalks in outdoor environments, IEEE Robotics and Automation Letters 7 (2) (2022) 3906–3913.

[15] T. Wang, V. Dhiman, N. Atanasov, Inverse reinforcement learning for autonomous navigation via differentiable semantic mapping and planning, Autonomous Robots (2023) 1–22.

[16] T. Zhang, Z. Liu, Z. Pu, J. Yi, Y. Liang, D. Zhang, Robot subgoal-guided navigation in dynamic crowded environments with hierarchical deep reinforcement learning, International Journal of Control, Automation and Systems (2023) 1–13.

[17] D. Shah, A. Sridhar, A. Bhorkar, N. Hirose, S. Levine, Gnm: A general navigation model to drive any robot, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2023, pp. 7226– 7233.

[18] M. Wen, J. Zhang, T. Chen, G. Peng, T. Chia, Y. Ma, Vision based sidewalk navigation for last-mile delivery robot, in: 2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV), IEEE, 2022, pp. 249–254.

[19] D. Seo, J. Kang, Collision-avoided tracking control of uav using velocity-adaptive 3d local path planning, International Journal of Control, Automation and Systems 21 (1) (2023) 231–243.

[20] J. S. Seng, T. J. Norrie, Sidewalk following using color histograms, Journal of Computing Sciences in College 23 (6) (2008) 172–180.

[21] A. Aktas¸, B. Dog˘an, O¨ . Demir, Tactile paving surface detection with deep learning methods, Journal of the Faculty of Engineering and Architecture of Gazi University 35 (3) (2020) 1685–1700.

[22] M. C. Ghilardi, R. C. Macedo, I. H. Manssour, A new approach for automatic detection of tactile paving surfaces in sidewalks, Procedia computer science 80 (2016) 662–672.

[23] Gulati, S. and Kuipers, B., 2008, May. High performance control for graceful motion of an intelligent wheelchair. In 2008 IEEE International Conference on Robotics and Automation (pp. 3932-3938). IEEE.

[24] Andaluz, V.H., Canseco, P., Varela, J., Ortiz, J.S., Pérez, M.G., Morales, V., Robertí, F. and Carelli, R., 2015. Modeling and control of a wheelchair considering center of mass lateral displacements. In Intelligent Robotics and Applications: 9th International Conference, ICIRA 2015, Portsmouth, UK, August 24–27, 2015, Proceedings, Part III (pp. 254-270). Springer International Publishing.

[25] Pasteau, F., Narayanan, V.K., Babel, M. and Chaumette, F., 2016. A visual servoing approach for autonomous corridor following and doorway passing in a wheelchair. Robotics and Autonomous Systems, 75, pp.28-40.

[26] V. S. Dorbala, A. Abdul Hafez, C. Jawahar, A deep learning approach for robust corridor following, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 3712–3718. doi:10.1109/IROS40897.2019.8967914.

[27] A. Tello, A. H. Abdul Hafez, B. Sarakbi, Real-time gp-based wheelchair corridor following, in: 2021 29th Signal Processing and Communications Applications Conference (SIU), 2021, pp. 1–4. doi:10.1109/SIU53274.2021.9478010.

[28] F. Chaumette, S. Hutchinson, Visual servo control. i. basic approaches, IEEE Robotics & Automation Magazine 13 (4) (2006) 82–90.

[29] A. H. H. A. A. Hafez, Visual servo control by optimizing hybrid objective function with visibility and path constraints, Journal of Control Engineering and Applied Informatics 16 (2) (2014) 120–129.

[30] E. Dombre, W. Khalil, Robot manipulators: modeling, performance analysis and control, John Wiley & Sons, 2013.

[31] V. S. Dorbala, A. A. Hafez, C. Jawahar, A deep learning approach for robust corridor following from an arbitrary pose, in: 2019 27th Signal Processing and Communications Applications Conference (SIU), IEEE, 2019, pp. 1–4.

[32] A. A. Hafez, A. Tello, S. Alqaraleh, Visual place recognition by dtw-based sequence alignment, in: 2019 27th Signal Processing and Communications Applications Conference (SIU), IEEE, 2019, pp. 1–4.

[33] Z. Fan, L. Meng, T. Q. Chen, J. Li, I. M. Mitchell, Learning motion predictors for smart wheelchair using autoregressive sparse gaussian process, in: 2018 IEEE

International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 713–718.

[34] S. Cho, D. H. Shim, Visual servoing framework using gaussian process for an aerial parallel manipulator, Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 233 (9) (2019) 3408–3425.

[35] Z. Hu, P. Sun, J. Pan, Three-dimensional deformable object manipulation using fast online gaussian process regression, IEEE Robotics and Automation Letters 3 (2) (2018) 979–986.

[36] E. Schulz, M. Speekenbrink, A. Krause, A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions, Journal of Mathematical Psychology 85 (2018) 1–16.

[37] C. Williams, C. Rasmussen, Gaussian processes for regression, Advances in neural information processing systems 8 (1995).

[38] S. T. Ounpraseuth, Gaussian processes for machine learning (2008).

APPENDIX

Code Samples:

1. Extarct hough lines and calculate corresponding velocity of each image in the dataset:

```
import cv2
import numpy as np
import math
from glob import glob
import pandas as pd
from openpyxl import load_workbook
csv_name =
'C:\\Users\\ismai\\Desktop\\GP_based_DVS\\GP_Based_DVS\\Dataset\\W-Compare
-TwoLines.xlsx'
wdf = pd.read_excel(csv_name, sheet_name="w All")
writer = pd.ExcelWriter(csv_name, engine='openpyxl',
mode="a",if_sheet_exists='replace') # writing DataFrame into excel sheets.
writer.book = load_workbook(csv_name) # open excel sheet
writer.sheets = dict((ws.title, ws) for ws in writer.book.worksheets)
path_new =
"C:\\Users\\ismai\\Desktop\\GP_based_DVS\\GP_Based_DVS\\Dataset\\TP\\Datas
et\\6\\*.png"
images_paths = glob(path_new)
for ii,image_path in enumerate (images_paths):
    showagain = 1
    if showagain == 1:
        showagain = 0
        image = cv2.imread(image_path)
        image_name = image_path.split("\\")[-1]
        save_images_path =
"C:\\Users\\ismai\\Desktop\\GP_based_DVS\\GP_Based_DVS\\Dataset\\TP\\Datas
et-Annotated\\6\\"
        save_images_path = save_images_path + "{0}".format(image_name)
        original = image.copy()
        image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        lower = np.array([3, 27, 118], dtype="uint8")
        upper = np.array([69, 99, 255], dtype="uint8")
```

```
        mask = cv2.inRange(image, lower, upper)
        edges = cv2.Canny(mask, 200, 250) # 200, 250
        lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 40, minLineLength= 80,
maxLineGap=40)  # 60 ,120 , 18
        bigger_x = []
        smaller_x = []
        big = 0
        small = 100
        count = -1
        y_prev = 0
        small_count = 0
        big_count = 0
        right_side = []
        left_side = []
        left = []
        right = []
        if lines is not None:
            print ("number of lines: ",len(lines))
            for line in lines:
                count = count + 1
                x1, y1, x2, y2 = line[0]
                list_line = line[0].tolist()

                if abs(y1 - y2) <10:
                    pass
                if x1 == 0:
                    if y1 > y_prev:
                        y_prev = y1
                    else:
                        continue
                elif x1 <= x2 :
                    if y2 > y1:
                        right_side.append(list_line)
                        #print("Right Side : ", right_side)
                        bigger_x.append(x2)
                    if y2 <= y1:
                        if x1 > 80 :
                            right_side.append(list_line)
                            bigger_x.append(x2)
                        else:
                            left_side.append(list_line)
                            smaller_x.append(x1)

                    for b in bigger_x:
                        if b > big:
                            big = b
                            big_count = count
```

```
            else:
                continue

        for s in smaller_x:
            if s < small:
                small = s
                small_count = count
            else:
                continue
        big_line = lines[[big_count][0]]
        small_line = lines[[small_count][0]]
    if (big_line[0] == small_line[0]).all:
        continue
print (image_name)
print(ii)
print("The Biggest :", big_line[0])
print("The Smallest :", small_line[0])
bx1, by1, bx2, by2 = big_line[0]
sx1, sy1, sx2, sy2 = small_line[0]
cv2.line(original, (bx1, by1), (bx2, by2), (0, 255, 0), 2)
cv2.line(original, (sx1, sy1), (sx2, sy2), (0, 255, 0), 2)
############ Left Line #################
if sx2 == sx1:
    sx1 = sx2 - 0.00000001
small_m = (sy2 - sy1) / (sx2 - sx1)
small_a = -small_m
small_b = 1
small_c = small_m * sx1 - sy1
left_rho = -small_c / math.sqrt(small_a ** 2 + small_b ** 2)
left_theta_rad = np.arctan(small_b / small_a)
left_theta_deg = (left_theta_rad * 180) / np.pi
############ Right Line #################
if bx2 == bx1:
    bx1 = bx2 - 0.00000001
big_m = (by2 - by1) / (bx2 - bx1)
big_a = -big_m
big_b = 1
big_c = big_m * bx1 - by1
right_rho = -big_c / math.sqrt(big_a ** 2 + big_b ** 2)
right_theta_rad = np.arctan(big_b / big_a)
right_theta_deg = (left_theta_rad * 180) / np.pi
print("Left Rho", left_rho)
print("Left Theta radian", left_theta_rad)
print("Right Rho", right_rho)
print("Right Theta radian", right_theta_rad)
print(" ----------------------------")
cv2.imwrite(save_images_path, original)
## Initialize Variable for Contro  Law ##
h = 1.6
```

```
            l = 0
            w = 0

            sinl = np.sin(left_theta_rad)
            cosl = np.cos(left_theta_rad)
            sinr = np.sin(right_theta_rad)
            cosr = np.cos(right_theta_rad)

            lambda_rho_1 = (-left_rho / h) * sinl
            lambda_rho_2 = (-right_rho / h) * sinr

            lambda_theta_1 = cosl / h
            lambda_theta_2 = cosr / h

            error = [[left_rho], [left_theta_rad], [right_rho], [right_theta_rad]]
            error = np.matrix(error)
            le = 100 * error

            Jv = [[-lambda_rho_1 * left_rho], [- lambda_theta_1 * left_rho], [-
lambda_rho_2 * right_rho],
                  [-lambda_theta_2 * right_rho]]

            Jv = np.matrix(Jv)

            vconst = 0.2

            Jw = [[(1 + np.square(left_rho)) * cosl], [left_rho * sinl], [(1 +
np.square(right_rho)) * cosr],
                  [right_rho * sinr]]

            Jw = np.matrix(Jw)

            pinv = (-1) * (np.linalg.pinv(Jw))

            fmat = le + (Jv * vconst)

            w = pinv * fmat
            w = float(w)
            print("W = ", w)
            wdf.loc[wdf["Image"] == image_name, "w6"] = np.round(w, 6)
        else:
            print("No Hough Lines Detected!")

wdf.to_excel(writer, sheet_name="w All", startrow=1, header=False, index=False)
writer.save()
```

2. Extarct Histogram of Gradients (HOG) features from the Images:

```
"""
In this module, HOG related functions are gathered, it can be run from
Feature_Extraction.py

"""

import cv2
import numpy as np
from Image_functions import read_resize_image
import time


def hog_parameters():

    features_per_cell = 9
    block_size = (32, 32)
    block_stride = (32, 32)
    cell_size = (32, 32)

    return features_per_cell, block_size, block_stride, cell_size


def extended_hog_parameters(image_height, image_width):
    # HoG descriptor parameters
    derivAperture = 1
    winSigma = -1.
    histogramNormType = 0
    L2HysThreshold = 0.2
    gammaCorrection = False
    nlevels = 64
    signedGradients = False
    winSize = (image_width, image_height)

    return derivAperture, winSigma, histogramNormType, L2HysThreshold,
gammaCorrection, nlevels, signedGradients, winSize


def hog_func(images_paths, image_height, image_width, dataset_name):

    # HoG descriptor parameters
    features_per_cell, block_size, block_stride, cell_size = hog_parameters()

    derivAperture, winSigma, histogramNormType, \
    L2HysThreshold, gammaCorrection, nlevels, \
    signedGradients, winSize = extended_hog_parameters(image_height, image_width)

    block_size_in_cell = np.array([block_size])/np.array([cell_size])
    block_overlap = block_size_in_cell -
np.array([block_stride])/np.array([cell_size])
```

```
    blocks_per_image = (np.divide(np.array([image_width, image_height]),
np.array([cell_size]))
                        - block_size_in_cell)/(block_size_in_cell-block_overlap)+1
    number_of_cells = (np.prod(blocks_per_image) * np.prod(block_size_in_cell))
    hog_descriptor_length = int(number_of_cells * features_per_cell)

    hog = cv2.HOGDescriptor(winSize, block_size, block_stride, cell_size,
features_per_cell, derivAperture, winSigma,
                            histogramNormType, L2HysThreshold, gammaCorrection,
nlevels, signedGradients)

    number_of_images = len(images_paths)
    descriptors = np.empty((number_of_images, hog_descriptor_length))
    images_labels = []
    elapsed_time = np.empty((number_of_images, 1))
    for ii, image_path in enumerate(images_paths):
        print(ii)
        image_name = image_path.split("\\")[-1].split(".")[0]
        images_labels.append(image_name)
        resize_image = read_resize_image(image_path, image_width, image_height)
        start = time.time()
        descriptors[ii, :] = np.transpose(hog.compute(resize_image, winSize))
        elapsed_time = time.time() - start

    descriptor_file = "_".join(["Descriptors/" + dataset_name, "Hog",
str(features_per_cell), str(block_size[0]),
                                str(block_stride[0]), str(cell_size[0]),
"descriptor"])
    np.savez(descriptor_file, descriptors=descriptors,
number_of_cells=number_of_cells,
             image_height=image_height, image_width=image_width,
number_of_images=number_of_images,
             features_per_cell=features_per_cell, block_size=block_size,
block_stride=block_stride,
             cell_size=cell_size, derivAperture=derivAperture, winSigma=winSigma,
histogramNormType=histogramNormType,
             L2HysThreshold=L2HysThreshold, gammaCorrection=gammaCorrection,
nlevels=nlevels,
             ignedGradients=signedGradients, winSize=winSize,
images_labels=images_labels, elapsed_time=elapsed_time)
```

```
"""
This script allows the user to apply a feature extraction method on a specific
dataset.

"""

from Image_functions import collect_images_paths
from HOGFunc import hog_func
from CNN import CNN
```

```
# Initial Configuration
features_extraction_index = 0   # ["Hog", "ResNet18"]

# ResNet18 layer number (refer to CNN.py)
layerIndex = 12

base_path = "Dataset\\"
dataset_name = "Noisy"
image_extension = "png"
image_width, image_height = (224, 224)

all_features_methods = ["Hog", "ResNet18"]
features_extraction_method = all_features_methods[features_extraction_index]

number_of_images, images_paths, images_labels = collect_images_paths(base_path +
dataset_name + "\\", image_extension)

if features_extraction_method == "Hog":
    hog_func(images_paths, image_height, image_width, dataset_name)
elif features_extraction_method == "ResNet18":
    CNN(features_extraction_method, layerIndex, images_paths, dataset_name)
```

3. Adjust and tune Gaussian Process (GP) parameters for training:

```
import numpy as np
from scipy.optimize import minimize
import time

"""
Python Gaussian Process module.
This code is supposed to accompany the tutorial 'An Introduction
to Gaussian Processes'

P.L.Green
University of Liverpool
22/05/19
"""
```

```
def Kernel(squared_distance, L):
    '''  Kernel function (squared exponential, with length scale L)

    '''

    return np.exp(-1/(2*L**2)*squared_distance)


def FindGramMatrix(X, L, Sigma, N, squared_distances):
    ''' Function that creates and inverts gram matrix for a squared
        exponential kernel with length-scale L.

    '''

    # squared_distances = FindSquaredDistances(X)
    K = Kernel(squared_distances, L)
    C = K + Sigma**2*np.identity(N)
    InvC = np.linalg.inv(C)
    return K, C, InvC


def FindSquaredDistances(X):
    ''' Function that finds the squared distances between
        inputs points (efficiently).

    '''

    if np.size(X[0]) == 1:
        Xsq = X**2
        temp = (Xsq[:, None] + Xsq[None, :])
        squared_distances = -2*np.outer(X, X) + temp
    else:
        squared_distances = np.empty((X.shape[0], X.shape[0]))
        # for ii in range(X.shape[0]):
        #     for jj in range(X.shape[0]):
        #         squared_distances[ii, jj] = sqeuclidean(X[ii, :], X[jj, :])
        #         print(ii, jj)
        # Xsq = np.sum(X**2, 1)
        for ii in range(X.shape[0]):
            print(ii)
            squared_distances[ii, :] = np.sum((X[ii, :] - X) ** 2, 1)
            # squared_distances[ii, :] = -2. * np.dot(X[ii, :], X.T) + (Xsq[ii] +
Xsq)

        # squared_distances = -2.*np.dot(X, X.T) + (Xsq[:, None] + Xsq[None, :])
    return squared_distances
```

```
def NegLogLikelihoodFun(Theta, a):
    ''' Returns negative log-likelihood function in a form
        suitable for scipy.optimize.fmin_bfgs.

    '''

    X = a[0]
    Y = a[1]
    N = a[2]
    squared_distances = a[3]
    L = Theta[0]
    Sigma = Theta[1]
    K, C, InvC = FindGramMatrix(X, L, Sigma, N, squared_distances)
    (Sign, LogDetC) = np.linalg.slogdet(C)
    LogDetC = Sign*LogDetC
    return 0.5*LogDetC + 0.5*np.dot(Y, np.dot(InvC, Y))


def dC_dL_Fun(X, L, K, N, squared_distances):
    ''' Derivative of matrix C w.r.t L (length scale).

    '''

    # squared_distances = FindSquaredDistances(X)
    return np.power(L, -3) * np.multiply(squared_distances, K)


def dC_dSigma_Fun(Sigma, K, N):
    ''' Derivative of matrix C w.r.t sigma (noise std).

    '''
    return 2*Sigma*np.eye(N)


def dNLL_dTheta(Theta, a):
    ''' Derivative of the negative log-likelihood w.r.t hyperparameters in a
        form suitable for scipy.optimize.fmin_bfgs.

    '''

    X = a[0]
    Y = a[1]
    N = a[2]
    squared_distances = a[3]
    L = Theta[0]
    Sigma = Theta[1]
    K, C, InvC = FindGramMatrix(X, L, Sigma, N, squared_distances)

    # Find dC / dL
```

```
    dC_dL = dC_dL_Fun(X, L, K, N, squared_distances)

    # Find dC / dSigma
    dC_dSigma = dC_dSigma_Fun(Sigma, K, N)

    # Find dlogp / dL
    dLogL_dL = (0.5 * np.trace(np.dot(InvC, dC_dL)) -
                0.5 * np.dot(Y, np.dot(InvC, np.dot(dC_dL, np.dot(InvC,
                                                                  Y)))))

    # Find dlogp / dSigma
    dLogL_dSigma = (0.5*np.trace(np.dot(InvC, dC_dSigma)) -
                    0.5*np.dot(Y, np.dot(InvC, np.dot(dC_dSigma, np.dot(InvC,
                                                                        Y)))))

    # Gradient vector
    gradient = np.array([dLogL_dL, dLogL_dSigma])

    return gradient


def Train(L0, Sigma0, X, Y, N):
    ''' Update hyperparameters using scipy.minimize

    '''

    Theta0 = [L0, Sigma0]
    squared_distances = FindSquaredDistances(X)
    a = (X, Y, N, squared_distances)
    b1 = (1e-6, 4)
    b2 = (1e-6, 1)
    bnds = (b1, b2)
    start_time = time.time()

    sol = minimize(NegLogLikelihoodFun, x0=Theta0, args=(a, ),
                   method='SLSQP', jac=dNLL_dTheta, bounds=bnds)

    elapsed_time = time.time() - start_time
    ThetaOpt = sol.x
    L = ThetaOpt[0]
    Sigma = ThetaOpt[1]
    K, C, InvC = FindGramMatrix(X, L, Sigma, N, squared_distances)
    return L, Sigma, K, C, InvC, elapsed_time


def callbackF(Theta):
    ''' Callback function for the fmin_bfgs optimisation routine

    '''
```

```
    print(Theta[0], Theta[1])


def Predict(X, xStar, L, InvCxY):
    ''' GP prediction

    '''
    diff = X - xStar
    squared_distances = np.einsum('ij,ij->j', diff, diff)
    # squared_distances = np.sum((X - xStar) ** 2, 1)
    k = Kernel(squared_distances, L)

    # yStarMean = np.dot(k, InvCxY)
    yStarMean = np.einsum('i,i->', k, InvCxY)

    return yStarMean
```

4. Train the GP on the annotated dataset:

```
import numpy as np
import LIRU_GP2 as GP
from Configuration import load_desc, exp_parameters, files_names
from Image_functions import collect_images_paths, read_resize_image
from sklearn.model_selection import train_test_split
import csv
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import LIRU_SparseGP as sGP


exp = exp_parameters()
np.random.seed(exp["seed"])
dataset_path = "Dataset\\"
image_extension = "png"

save_path = "Results\\"
saved_file_name, desc_file, test_dataset_path = files_names(exp, dataset_path,
save_path)

ground_truth_velocities_file_name = "Ground Truth.csv"  # Velocities
with open(ground_truth_velocities_file_name) as csv_file:
    velocities_file = csv.reader(csv_file, delimiter=',')

    velocities_file_content = np.array([[int(row[0].split(".")[0]), float(row[1])]
for row in velocities_file])


sorter = np.argsort(velocities_file_content[:, 0])

images_paths, velocities, images_labels = [], [], []
for ii in exp["train_datasets"]:
    number_of_images, current_images_paths, current_images_labels =
collect_images_paths(dataset_path + ii + "\\", image_extension)
    images_paths = np.concatenate([images_paths, current_images_paths])
    labels_indices = sorter[np.searchsorted(velocities_file_content[:, 0],
current_images_labels, sorter=sorter)]
    velocities = np.concatenate([velocities,
velocities_file_content[labels_indices, 1]])
    images_labels = np.concatenate([images_labels, current_images_labels])

if exp["test_dataset_index"] is None:
    training_images_paths, testing_images_paths, \
        training_labels, testing_labels, \
        training_velocities, testing_velocities = \
        train_test_split(images_paths, images_labels, velocities,
test_size=exp["test_percent"], random_state=exp["seed"])
```

```
else:
    if exp["test_subset"] == "JPEG_Compression":
        image_extension = "jpeg"

    _, sub_images_paths, sub_images_labels =
collect_images_paths(test_dataset_path, image_extension)
    labels_indices = sorter[np.searchsorted(velocities_file_content[:, 0],
                                            sub_images_labels, sorter=sorter)]
    number_of_testing_images = round(exp["test_percent"] * len(images_labels))
    test_percent = number_of_testing_images / len(labels_indices)
    _, testing_images_paths, _, testing_labels, _, testing_velocities = \
        train_test_split(sub_images_paths, sub_images_labels,
velocities_file_content[labels_indices, 1],
                         test_size=exp["test_percent"], random_state=exp["seed"])
    test_indices = np.invert(np.isin(images_labels, testing_labels))
    training_images_paths = images_paths[test_indices]
    training_labels = images_labels[test_indices]
    training_velocities = velocities[test_indices]

velocities_mean = np.mean(training_velocities)
velocities_std = np.std(training_velocities)
training_velocities = (training_velocities - velocities_mean) / velocities_std
testing_velocities = (testing_velocities - velocities_mean) / velocities_std

number_of_training_images = len(training_images_paths)
number_of_testing_images = len(testing_images_paths)

# Load descriptors
ground_truth_labels = []
descriptors = []
training_descriptors = []
if exp["features_extraction_method"] in ("Hog", "ResNet18"):
    descriptors, ground_truth_labels = load_desc(desc_file[0])
    for jj in range(1, len(exp["train_datasets"])):
        current_descriptors, current_ground_truth_labels =
load_desc(desc_file[jj])
        descriptors = np.concatenate([descriptors, current_descriptors], 0)
        ground_truth_labels = np.concatenate([ground_truth_labels,
current_ground_truth_labels])

    training_images = np.array([descriptors[ground_truth_labels ==
str(int(label)), :] for label in training_labels]).squeeze()

    del descriptors

elif exp["features_extraction_method"] == "Raw":
    training_images = np.empty((number_of_training_images, exp["image_width"] *
exp["image_height"]))
    for ii in range(number_of_training_images):
```

```
        # print("Training Image:", ii)
        training_images[ii, :] = read_resize_image(training_images_paths[ii],
                                                   exp["image_width"],
exp["image_height"]).flatten()


# Train GP
L0 = exp["L0"]        # Initial length scale
Sigma0 = exp["Sigma0"]   # Initial noise standard deviation

if exp["GP_type"] == "GP":
    L, Sigma, K, C, InvC, train_elapsed_time = \
        GP.Train(L0, Sigma0, training_images, training_velocities,
number_of_training_images)  # Train GP
elif exp["GP_type"] == "GP_Sparse":
    M = exp["M"]      # No. sparse points
    NoCandidates = exp["NoCandidates"]   # No. of candidate sets of sparse points
analysed
    L, Sigma, K, C, InvC, Xs, Ys, LB_best, train_elapsed_time = \
        sGP.Train(L0, Sigma0, training_images, training_velocities,
number_of_training_images, M, NoCandidates)
    training_images = Xs
    training_velocities = Ys
    number_of_training_images = M

print('Hyperparameters:', L, Sigma)      # Print hyperparameters
print('Elapsed Time:', train_elapsed_time)     # Print time taken to train GP


##### Test Section
ground_truth_labels = []
descriptors = []
if exp["features_extraction_method"] in ("Hog", "ResNet18"):
    descriptors, ground_truth_labels = load_desc(desc_file[0])
    for jj in range(1, len(exp["train_datasets"])):
        current_descriptors, current_ground_truth_labels =
load_desc(desc_file[jj])
        descriptors = np.concatenate([descriptors, current_descriptors], 0)
        ground_truth_labels = np.concatenate([ground_truth_labels,
current_ground_truth_labels])

    test_features = np.array([descriptors[ground_truth_labels == str(int(label)),
:] for label in testing_labels]).squeeze()
    del descriptors

# Make some predictions
Y_StarMean = np.zeros([number_of_testing_images])         # mean of GP predictions
Y_StarStd = np.zeros([number_of_testing_images])          # std of GP predictions
test_elapsed_time = np.zeros([number_of_testing_images])
```

```
for ii in range(number_of_testing_images):
    # print("Testing Image:", ii)
    if exp["features_extraction_method"] in ("Hog", "ResNet18"):
        test_descriptor = test_features[ii, :]
    else:
        test_descriptor = read_resize_image(testing_images_paths[ii],
exp["image_width"], exp["image_height"]).flatten()
    Y_StarMean[ii], Y_StarStd[ii], test_elapsed_time[ii] =
GP.Predict(training_images, test_descriptor, L, Sigma,

training_velocities, K, C, InvC, number_of_training_images)


mean_squared_error_numpy = np.mean((Y_StarMean - testing_velocities) ** 2)
print("MSE = ",mean_squared_error_numpy)

r_score = r2_score(testing_velocities, Y_StarMean)
MSE= mean_squared_error(testing_velocities, Y_StarMean),
MASE = mean_absolute_error(testing_velocities, Y_StarMean)
print("R Score = ",r_score)
print("Mean Square Error = ",MSE)
print("Mean Absolute Square Error = ",MASE)


np.savez(saved_file_name, exp=exp, L=L, Sigma=Sigma, K=K,
         C=C, InvC=InvC, train_elapsed_time=train_elapsed_time,
Y_StarMean=Y_StarMean,
         Y_StarStd=Y_StarStd, test_elapsed_time=test_elapsed_time,
squared_mean_error=mean_squared_error,
         training_labels=training_labels, testing_labels=testing_labels,
r_score=r_score,
         training_velocities=training_velocities,
testing_velocities=testing_velocities,
         velocities_mean=velocities_mean, velocities_std=velocities_std)
```

5. Test the trained GP model and integrate it with Fuzyy Controller on new images (Conduct real experiment)

```
import time
import cv2
from HOGFunc import hog_parameters, extended_hog_parameters
from GP_New import Predict
import numpy as np
from picamera.array import PiRGBArray
from picamera import PiCamera
from basic_subertoorh_instructions import initialization, send_vel, stop
import matplotlib.pyplot as plt
from collections import deque
```

```
image_height = 224
image_width = 224
fileName = "GP_train_Normal_test_Normal_Hog_9_32_32_32_ALL.npz"
normal_descriptor_name = "Normal_Hog_9_32_32_32_descriptor.npz"

with np.load(fileName) as results:
    L = results["L"]
    Sigma = results["Sigma"]
    K = results["K"]
    C = results["C"]
    InvC = results["InvC"]
    training_velocities = results["training_velocities"]
    training_labels = results["training_labels"]

number_of_training_images = len(training_velocities)

with np.load(normal_descriptor_name) as normal_descriptor_file:
    normal_descriptor = normal_descriptor_file["descriptors"]
    normal_ground_truth_labels = normal_descriptor_file["images_labels"]


velocities_mean = np.mean(training_velocities)
velocities_std = np.std(training_velocities)
training_velocities = (training_velocities - velocities_mean) / velocities_std

ground_truth_labels = np.concatenate([normal_ground_truth_labels])

training_descriptors = np.concatenate([normal_descriptor], 0)
training_descriptors = np.array([training_descriptors[ground_truth_labels ==
str(int(label)), :] for label in training_labels]).squeeze().T

features_per_cell, block_size, block_stride, cell_size = hog_parameters()
derivAperture, winSigma, histogramNormType, L2HysThreshold, gammaCorrection, \
nlevels, signedGradients, winSize = extended_hog_parameters(image_height,
image_width)

hog = cv2.HOGDescriptor(winSize, block_size, block_stride, cell_size,
features_per_cell, derivAperture, winSigma,
                        histogramNormType, L2HysThreshold, gammaCorrection,
nlevels, signedGradients)

InvCxY = np.dot(InvC, training_velocities)

sabertooth_object = initialization()

camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 20
```

```
rawCapture = PiRGBArray(camera, size=(640, 480))  #640, 480
time.sleep(0.1)
end = []
counter = 0

w = []
fig, ax = plt.subplots()
plt.ion()
plt.show()
plt.ylim(-4, 4)
plt.xlabel("Time (s)", fontsize=10)
plt.ylabel("w (rad/s)", fontsize=10)
plt.xticks()
# plt.legend("w")
a1 = deque([0]*50)
b1 = deque([0]*50)
line, = plt.plot(a1)
fourcc = cv2.VideoWriter_fourcc(*'XVID')  # 'x264' doesn't work
out = cv2.VideoWriter('005_output.avi',fourcc, 20, (image_width, image_height))
for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
    image = frame.array
    start_cycle = time.time()
    cv2.imshow("Frame", image)
    image = cv2.resize(image, (image_width, image_height))

    out.write(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.medianBlur(image, 3)  # Removes salt and pepper noise by
convolving the image with a (3,3) square kernel
    image = cv2.GaussianBlur(image, (3, 3), 0)

    rawCapture.truncate(0)

    descriptor = hog.compute(image, winSize)

    Y_StarMean = Predict(training_descriptors, descriptor, L, InvCxY)
    w.append(Y_StarMean)

    if cv2.waitKey(1) & 0xFF == ord("s"):
        stop(sabertooth_object)
        break

    Y_StarMean = Y_StarMean * velocities_std + velocities_mean
    send_vel(Y_StarMean, sabertooth_object)

    end.append(time.time() - start_cycle)
    print("Time = ", end[counter])
    sumTime = np.sum(end)
```

```
    plt.xlim(sumTime-5, sumTime + 0.5)
    a1.appendleft(w[counter])
    datatoplot = a1.pop()
    b1.appendleft(sumTime)
    datatoplot2 = b1.pop()
    line.set_ydata(a1)
    line.set_xdata(b1)
    minor_ticks = np.arange(sumTime-5, sumTime + 0.5, 0.5)
    major_ticks = np.arange(sumTime-5, sumTime + 0.5, 0.5)
    ax.set_xticks(major_ticks)
    ax.set_xticks(minor_ticks, minor=True)
    plt.draw()
    plt.pause(0.000001)

    counter = counter + 1

out.release()
cv2.destroyAllWindows()
plt.show()
exp = 5
saved_file_name = "exp_" + str(exp) + ".npz"
np.savez(saved_file_name, exp=exp, L=L, Sigma=Sigma, K=K,
         C=C, InvC=InvC, Y_StarMean=w,
         test_elapsed_time=end,
         training_labels=training_labels,
         training_velocities=training_velocities,
         velocities_mean=velocities_mean, velocities_std=velocities_std)
```

```
# File for programming Sabertooth commands on the wheelchair.

# !/usr/bin/env python
from pysabertooth import Sabertooth
import serial.tools.list_ports as port
import time


def forward():
    print("forward")
    saber.drive(1, 12)
    saber.drive(2, 12)
    time.sleep(2)
    saber.drive(1, 0)
    saber.drive(2, 0)


def backward():
    print("backward")
    saber.drive(1, -12)
    saber.drive(2, -12)
```

```
    time.sleep(2)
    saber.drive(1, 0)
    saber.drive(2, 0)


def left():
    print("left")
    saber.drive(1, -15)
    saber.drive(2, 15)
    time.sleep(2)
    saber.drive(1, 0)
    saber.drive(2, 0)


def right():
    print("right")
    saber.drive(1, 15)
    saber.drive(2, -15)
    time.sleep(2)
    saber.drive(1, 0)
    saber.drive(2, 0)


def stop(saber):
    print("stop")
    saber.drive(1, 0)
    saber.drive(2, 0)


def translate(value, leftMin, leftMax, rightMin, rightMax):
    # Figure out how 'wide' each range is
    leftSpan = leftMax - leftMin
    rightSpan = rightMax - rightMin

    # Convert the left range into a 0-1 range (float)
    valueScaled = float(value - leftMin) / float(leftSpan)

    # Convert the 0-1 range into a value in the right range.
    return rightMin + (valueScaled * rightSpan)


def callback(data):
    # Callback function whenever a /cmd_vel is received from RoS
    # print data
    saber.text('m1:startup')
    saber.text('1,p100')
    saber.text('md:1000\r\n')

    saber.text('m2:startup')
    saber.text('r1:1940\r\n')
    saber.text('r2:1940\r\n')

    # The data.linear.x is in the range [-1, 1]
    speed = translate(data.linear.x, -1, 1, -60, 60)
    # speed = translate(data.linear.x,-1,1,-2047,2047)
```

```
    SPEED = 'md: {}\r\n'.format(speed)
    angle = translate(data.angular.z, -1, 1, 20, -20)
    # angle = str(translate(-data.angular.z,-1,1,2047,-2047))
    ANGLE = 'mt: {}\r\n'.format(angle)
    # saber.text('m1:startup')
    # saber.drive(1,speed)
    # saber.drive(2,speed)
    if angle + speed > 20:
        speed1 = 20
    else:
        speed1 = angle + speed
    if speed - angle < -20:
        speed2 = -20
    else:
        speed2 = speed - angle

        saber.drive(1, speed)
        saber.drive(2, speed)

    if angle > 0:
        print("negative")
        saber.drive(1, speed2)
        saber.drive(2, speed1)
    elif angle < 0:
        print("positive")
        saber.drive(1, speed2)
        saber.drive(2, speed1)

    # saber.text('m2:startup')
    # MD: 0\r\n
    print(SPEED)
    print(ANGLE)

    # saber.text(SPEED)

    pass
    # print message


def sabertoothStatusCallback(data):
    temperature = ('T [C]: {}'.format(saber.textGet('m2:gett')))

    battery = ('battery [mV]: {}'.format(saber.textGet('m2:getb')))
    print(battery, temperature)


def travel():
    while True:
        forward()
        stop()
        left()
        stop()
        right()
        stop()
        backward()
```

```
        stop()


wl = 0
wr = 0


def send_vel(w, saber):

    speed = 20

    if w < -0.002  and w > -0.05:
        print("left")
        saber.drive(1, speed)
        saber.drive(2, speed + 100*abs(w))
    elif w > 0.015 and w < 0.06 :
        print("right")
        saber.drive(1, speed + 100*abs(w))
        saber.drive(2, speed)
    else:
        print("straight!")
        saber.drive(1, speed)
        saber.drive(2, speed)
    print("w = ", w)

def send_vel_stop(w, saber):


    if abs(w) < 0.0005:
        speed = 0
        saber.drive(1, speed)
        saber.drive(2, speed + 8*abs(w))
        print ("w = ",w)
        print ("v = ",speed)
    else:
        speed = 20

        if w < -0.002  and w > -0.05:
            print("left")
            saber.drive(1, speed)
            saber.drive(2, speed + 100*abs(w))
        elif w > 0.015 and w < 0.06 :
            print("right")
            saber.drive(1, speed + 100*abs(w))
            saber.drive(2, speed)
        else:
            print("straight!")
            saber.drive(1, speed)
            saber.drive(2, speed)
        print("w = ", w)

def send_vel_2(w_org, saber, max_w, min_w):

    angle = 8
    speed = 30
```

```
    w = translate(w_org, min_w, max_w, 0, 1)
    w_norm_lin = w * 0.5
    speed_to_saber = (1 - w_norm_lin) * speed + w * angle

    if w_org < -0.2:
        print("left")
        saber.drive(1, (1 - w_norm_lin) * speed)
        saber.drive(2, speed_to_saber)
    elif w_org > 0.2:
        print("right")
        saber.drive(1, speed_to_saber)
        saber.drive(2, (1 - w_norm_lin) * speed)
    else:
        print("straight!")
        saber.drive(1, (1 - w_norm_lin) * speed)
        saber.drive(2, (1 - w_norm_lin) * speed)

def initialization():
    print("\nDetecting sabertooth....\n")
    pl = list(port.comports())
    print(pl)
    address = ""
    for p in pl:
        print(p)
        if "Sabertooth" in str(p):
            address = str(p).split(" ")
    print("\nAddress found @")
    print(address[0])

    saber = Sabertooth(address[0], baudrate=9600, address=128, timeout=0.1)

    return saber


if __name__ == '__main__':
    saber = initialization()
    travel()
```

CURRICULUM VITAE

Name SURNAME: Ismail HAJ OSMAN

Education:

Gaziantep University , Bachelor's Degree , Electrical And Electronics Enginnering (2015 - 2020)

Gaziantep University , Master Degree, Electrical And Electronics Engineering (2021 - 2023)

Publications: