Autonomous Lane Keeping for UGV Using a Monocular Vision System and Machine Learning

Lars Muggerud, Ørjan Øvsthus, Didrik Nicolai Robsrud

***Abstract*—**
Autonomous robot navigation and machine learning are fields within robotics that have been getting increased traction in recent years. Lane detection using machine vision and a machine learning model has been applied to an unmanned ground vehicle to achieve autonomous lane keeping with decent accuracy at slow speed. Two different methods for lane detection and visual measurement is explored.

***Index Terms*—LIDAR, Clearpath Husky, ROS2 Foxy, Navigation 2, Ubuntu 20.04**

I. Introduction

THIS project aims to create a lane detection algorithm to detect road lanes and autonomously navigate a UGV to drive within the lanes. Two different lane detection algorithms is explored, traditional edge-detection based approach and lane detection using machine learning. Camera calibration is done in order to relate camera coordinates to real world coordinates. As a 2D-camera is unable to measure dept directly, two different techniques to extract the real world coordinates from the camera image is explored. One technique relies on using data from the camera calibration to generate a dynamic z-coordinate based on the extrinsic calibration of the camera. The other technique relies on using a depth image together with data from the intrinsic calibration matrix to calculate real world coordinates based on the pinhole camera model and reported distance from camera to the specific point in the image.

As a part of this project a Husky A200 UGV is set up with Robot Operating System 2(ROS2) to have autonomous navigational capabilities using Simultaneous Localisation and Mapping and a well known navigation system for ROS2. The documentation on this UGV setup is found at: [1].

A. Motivation

Unmanned ground vehicles can do repetitive and time consuming tasks. Common tasks for an UGV is transportation, data collection and military use to name a few. To be able to autonomously accomplish these tasks, autonomous navigation is required. Autonomous lane keeping is a part of the broad subject that is autonomous navigation and will be the scope of this project. In order to achieve autonomous navigation, such as lane keeping, a robot would need some perception of its environment. Robot perception is achieved by adding various sensors to the system. Increasing the number of different sensors will diversify the data input and make the total measurement more accurate and reliable. By using a vision system to detect road lanes and then implementing this lane detection system with a Simultaneous Localisation and Mapping(SLAM) based navigation system, a robust lane keeping system for robot navigation can be achieved.

Ajit Jha is with the Department of Engineering Sciences, University of Agder, Grimstad, Norway

B. Project Outline

This report is made up of 8 sections including this section, I - Introduction. Section II - Related work, describes similar previous works, advantages/disadvantages to their solutions and how this project overcomes the problems. Section III - Method, describes the techniques used to solve the technical problems of this project. Section **??** - Results & Discussion, presents the results of the project work. Section V - Conclusion, sums up the objectives of the project and whether or not the results were satisfactory.

II. Related work

A. Sensors

Red, green and blue (RGB in short) is the three base colours used for representing visible light, every colour can be considered a combination of these three. Cameras often have a red, green and a blue sensor and the output is three layered matrices with dimensions $pixel\ height \cdot pixel\ width$, one layer for each colour.

Laser imaging, detection, and ranging (LiDAR) sensors measure distance using laser. In robotics it is common for LiDARs to rotate or sweep in order measure in 2 dimensions. 2D LiDARs are usually configured to measure in a plane parallel to the ground, normally with only one laser. 3D LiDARs often use multiple lasers arranged in a fan-shape rotating together, or an adjustable mirror.

Data processing with some related paper

Sensor fusion with some related paper

Fusing the odometry and the imu data

All real sensors have some uncertainty associated with them, which causes imprecise measurements. Multiple sensors can be combined to increase their accuracy. This combination of sensor data is often done with a Kalman filter, where more accurate data have a greater influence than the less accurate. Kalman filter is only suitable for linear systems, and requires that the observation equation is linear. Eksteded Kalman filter (EKF) is an non linear filter [2]. The ”husky system” is non linear and EKF is therefor used.

B. SLAM

SLAM is a well known technique used in indoor robot navigation for autonomous robots. It is a technique which allows a robot to build a map of the environment and, at the same time, determine its own position in this environment [3]. Hence, the name ”Simultaneous Localisation And Mapping”. SLAM techniques can roughly be divided into visual based and LiDAR based SLAM [3].

Visual based SLAM is inspired by human vision and utilises cameras, usually stereo or monocular, to do SLAM. Visual SLAM often struggles in environments that lack texture, such as warehouses with blank white walls. LiDAR SLAM relies on data from a LiDAR sensor, usually 2D or 3D, to do SLAM. A LiDAR provides spatial information of the environment in form of points, either as a 2D scan or as a 3D point cloud. Compared to visual SLAM, LiDAR SLAM is more robust and is therefore often employed in application such as automated guided vehicles [3].

C. Navigation & Path Planning

Effective indoor navigation is not a new concept, a notable mentions are DERVISH [4] from 1994, and RHINO [5] and MINERVA [6]. For outdoor navigation, [7] proposed an outdoor navigation system for a pedestrian-like autonomous navigation in an outdoor city environment. For robots utilising ROS2, the go-to navigation system is the Navigation2 navigation Stack [8]. Navigation 2 needs a map and a localisation algorithm, this could be handled by SLAM Toolbox [8].

D. Image Processing

Edge detection and stuff

E. Lane Detection

Lane detection is a significant part of outdoor autonomous navigation, as roads often come with painted lines. Traditionally, this is done through extraction of features(example [9]). This usually has good results, but struggles with robustness according to [10]. Therefore, the use of CNN to train lane detection models has become increasingly popular. The work of [11] use an existing neural network to perform lane and vehicle detection at frame rate required for a real-time system. They state that their results ads to the hypothesis that deep learning holds promise for autonomous driving.

Although work on CNN for lane detection seem promising, [11] states that CNN based approaches needs large data-sets that represents all possible driving environments and scenarios.

III. Method

This chapter describes the methods used to solve technical issues during the project, as well as the use of software during the development of the solution. Note: as mentioned in section I, a website-like report describing how to set up a Husky A200 UGV, with an array sensors, to do SLAM navigation is also made as a part of this project, please refer to this for details: [1]

A. Hardware setup

The husky is an unmanned ground vehicle from the robot company Clearpath [12]. It has 4 wheels, motor control, 24v battery. The sensors and computers attached to the robot is powered by the 24v battery through a voltage converter/regulator system. The converter have outputs of 24, 12 and 5 volts. This is a 4 wheel drive differential drive UGV, which gives the robot 2 degrees of freedom (forward, reverse and turning).

Nvidia Jetson AGX Xavier is a low-power embedded computer with an ARM architecture CPU [13]. This computer interfaces with the Husky by serial communication, and are responsible for control of the Husky.

IMU or Inertial measurement unit is an acceleration - and rotation sensor that outputs orientation, angular velocity, linear acceleration and the covariance related to the outputs.

The Ouster LiDAR used in this project is a 3D LiDAR, which means that is can measure distance in 3D space. This LiDAR outputs two relevant data-sets; ”PointCloud2”, a 3D point cloud and ”LaserScan”, a 2D scan along the $0\,\mathrm{deg}$ laser in the sensor.

Initially four e-CAM130A CUXVR cameras was supposed to be fused with the Ouster LiDAR and used for this project. The supported software for e-C‘CAM130A requires JetPack 4.6 witch is based on Ubuntu 18.04. Ubuntu 18.04 does not support Robot Operating System 2 (ROS2) Foxy, and since communicating across different ROS platforms is not officially supported and can lead to problems, it was decided to use the L515 Intel® RealSense™ LiDAR Camera, as this was available. Figure 1 illustrates how the hardware is set up and the interfaces between the different hardware parts in the system.

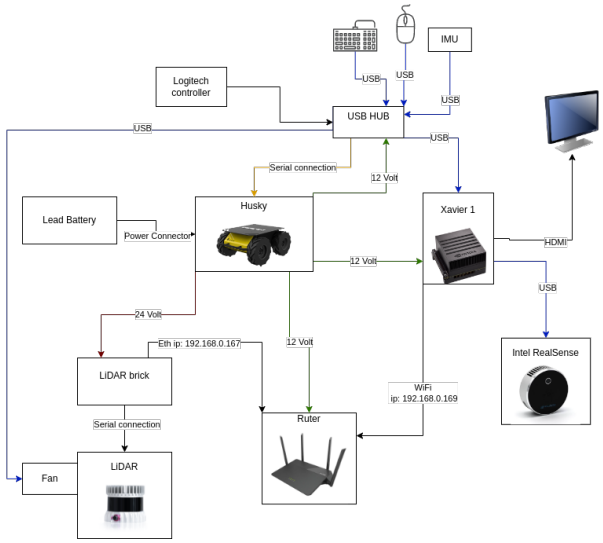

Fig. 1. Block diagram of hardware

One Nvidia Jetson AGX Xavier is set up with Jetpack 5.0.2 and Robot Operating System 2 (ROS2) Foxy. ROS2 contains a set of software libraries and tools for building robot applications [14]. JetPack is an operating system based on Ubuntu designed for the Nvidia Jetson series. This system is

used to set up the robot platform and to get every part of the robot to work together.

B. ROS2 Setup

The Husky UGV is set up with ROS2 Foxy, which takes care of the communication between all the different sensors and actuators on the UGV, as mentioned above. Figure 2 shows a simplified overview of how ROS2 fuses the different sensors and systems together to achieve autonomous 2D-LiDAR navigation.

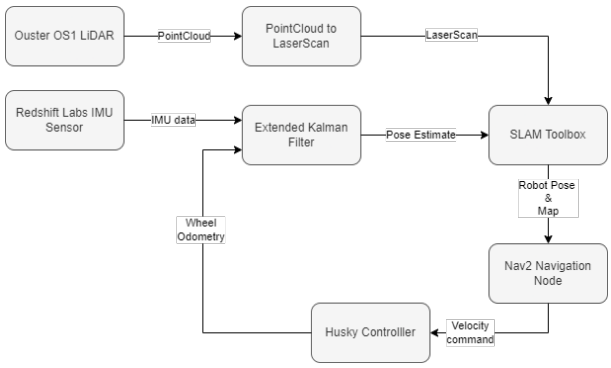

Fig. 2. Simplified overview of ROS2 control system.

The LiDAR node from figure 2 creates a 3D point cloud of the surrounding area based on LiDAR data and publishes this information to ROS2. It also creates a 2D laser scan where it publishes distance data from the laser closest to $0°$. However, this method of generating a laser scan will only generate a 2D plane and has no way of detecting objects below or over this plane. Figure 3 illustrates how the UGV would not be able to detect a box under the line of "sight", while the box is clearly big enough to cause issues.

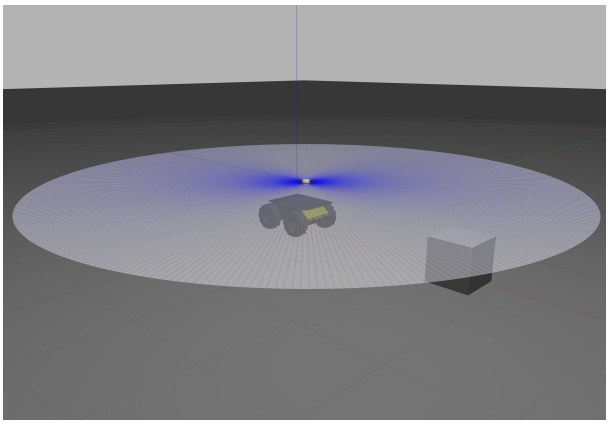

Fig. 3. Gazebo simulation of normal laser scan. Notice how the box is below the laser and therefore, it is not detected.

The ROS2 package "*pointcloud_to_laserscan*" [15], seen in figure 2, is used in order to deal with the issue illustrated in figure 3 and take advantage of the 3D point cloud of the LiDAR. The built-in method uses the laser closest to $0°$ as laser scan. The PtL method takes in a minimum height, Z_{min}, and maximum height, Z_{max}, then flattens every detected obstacle between these heights and sends this to laser scan. Therefore, using this method, the box in figure 3 would be included in the laser scan, as long as it is detected by the LiDAR point cloud. The converted laser scan is then published to ROS2.

The Extended Kalman Filter from figure 2 fuses data from the IMU sensor and the robots odometry system. The filter will weight the two inputs based on their perceived quality and then generates a pose estimate based on the provided data and their weights. The fused result of this data is then published to ROS2.

The ROS2 package "*slam_toolbox*" [16], illustrated in figure 2 as "SLAM Toolbox", takes in laser scan from the PointCloud to LaserScan node and pose estimation from the extended kalman filter. This information is used to do LiDAR SLAM, as mentioned in section II-B, which generates a map of the environment and calculates a more accurate pose estimate using pose graph estimation. The SLAM toolbox then publishes the map and pose data to ROS2.

The Nav2 Navigation node, from figure 2, utilises the map and pose data published from the SLAM Toolbox node in order to set up autonomous robot navigation [8]. In short, the navigation node use provided information and a path planning algorithm to safely manoeuvre the robot to a given point in the 2D map provided by the SLAM Toolbox node.

C. Lane Detection

In this section, two different methodologies for lane detection is described. The first method involves lane detection based on traditional machine vision, where the vision system finds features that are typical for a road lane and then extracts spatial information from the image. The second method involves lane detection based on machine learning, specifically CNN, where a model trained to detect road lanes is used to detect lanes based on a machine vision input. Figure 4 illustrates the structure of how lane detection could be implemented with the UGV through ROS2 Navigation2 system.

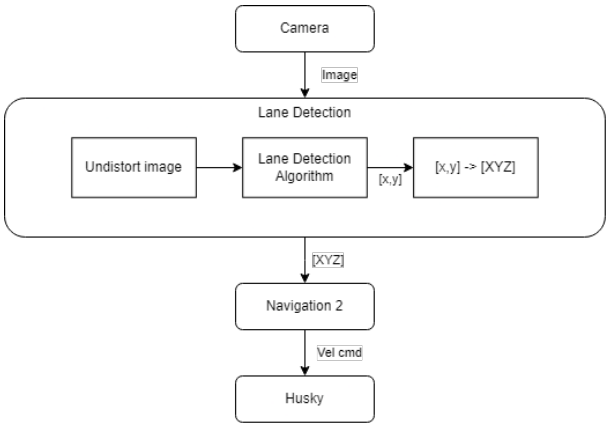

Fig. 4. Simplified overview of lane detection implementation through the ROS2 Navigation2 system

1) Traditional Lane Detection: A simple program for lane detection was developed in python. This program captures

an image with an camera and then processes the image with functions from OpenCV. The image gets blurred with Gaussian blur, then edges are extracted with Canny edge detection. HoughLines are used to draw lines on top of the Canny-filtered image. The four longest lines are kept. A point is created from the mean of the four lanes. This process repeats several times each second. The median point for every 10th iteration is drawn on top of the image along with the lines and presented to the user, as seen in figure 5.

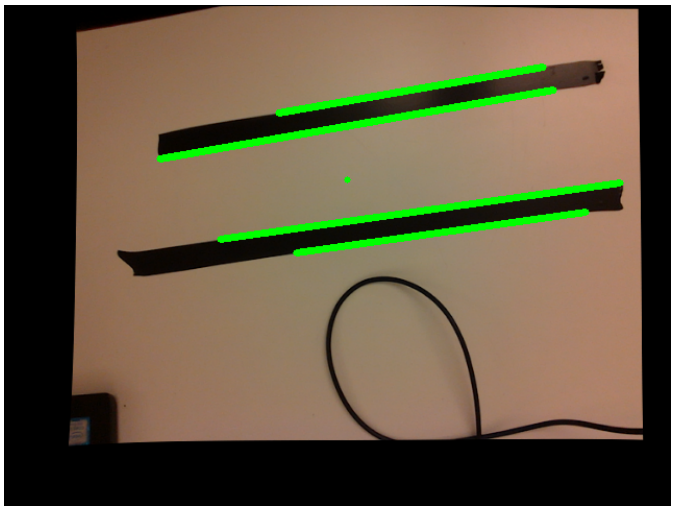

Fig. 5. Simple line detection, dot marks the centre

2) CNN Lane Detection: Lane detection using convolutional neural networks(CNN) is achieved by utilising CNN to train a deep learning model to detect road lanes. In order to get a robust model, large data sets of high quality is needed. In this context, a high quality data set could be a set of images of roads with all different possible types of wear and images taken during all different weather conditions. This is difficult to achieve though. Thus, a ”perfect” data set that is not too large is difficult to produce. The decision was therefore made to use a pre-trained CNN model provided from [17].

The pre-trained CNN model is trained to detect lanes on pictures taken from the point of view of a camera mounted on a car driving along a road, as illustrated in figure 6. Figure 6 also illustrates how the image is coloured green where the road is detected. This is due to some processing of the model prediction output. Notice how this model detects the road lane being driven in and not the opposite one, this would be inconvenient as it could cause head-on collisions.

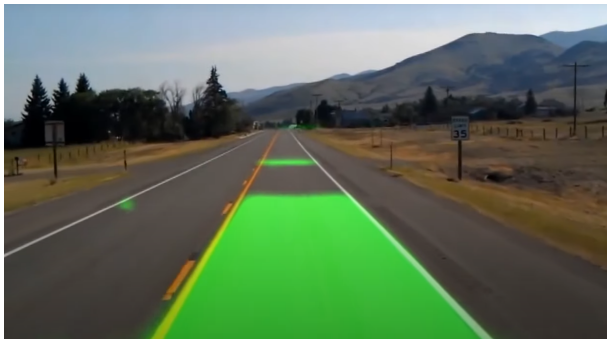

Fig. 6. Lane detection model detecting a road lane. Notice how the detected lane is colored green.

The roads are coloured green when the model detects lanes, as mentioned above. This is due the nature of how the model predicts lanes, combined with some image processing after model prediction. The prediction output of the model is an array where each pixel is weighted from 0-1 based on the likelihood of this pixel being located within a road lane. The likelihood array are turned to a propperly scaled layer of green and layered on top of the original image, with some transparency. This creates the overlay illustrated in figure 6. A ”centre of mass”, or the mean, of the likelihood array is used as the ”desired” point and this would have been passed robot navigation system. The Image processing algorithm is presented in algorithm 1.

Algorithm 1 Algorithm for extracting CNN model prediction data.

Input: image $(n\ X\ m\ X\ 3)$
Output: $processed\ image$, $mass[u, v]$

1: Resize input image to fit model
2: Pass resized image into model
3: Get prediction from model
4: Multiply returned prediction by 255
5: $average\ fit$: Get an average of the 5 last model predictions.
6: $mass[u, v]$: Find the center of mass from the average fit array.
7: $center$: Generate a sparse array of same size as the average fit. This array is only populated with the value 255 at the calculated center of mass, the rest is 0.
8: $lane\ image$: Stack the average fit array between a zero array and the center array. All of the same size.
9: Resize the lane image to match the size of the input image: $(n\ X\ m\ X\ 3)$.
10: $processed\ image$: Blend the input image and the lane image to make the processed image.
11: **return** $processed\ image$, $mass[u, v]$

D. Extracting Real Coordinates from Image

Extracting real world coordinates from image coordinates traditionally requires intrinsic and extrinsic image calibration. Two alternative methods to extract dept from the vision system is therefore described. The first method involves using algebra to rotate the imaging plane so that it becomes parallel to

the ground instead of parallel to the camera focal plane. The second method involves utilising the cameras built in LiDAR to calculate the distance to a certain point in the image and then use this information to extract real world coordinates.

1) Camera calibration:

a) Intrinsic camera calibration:

A model of a pinhole camera is often used to model cameras. However, cameras usually boast a lens which usually brings some distortions to the captured image [18], as mentioned in section III-D. Intrinsic camera parameters can be obtained trough intrinsic calibration, and these can be used to reduce the distortion of captured images. The parameters usually consist of focal length and the optical centre for x and y direction, together with skew, but skew will not be considered here. The pinhole model can be applied to the calibrated camera [19]. The intrinsic calibration matrix will be unique for each unique camera.

The intrinsic calibration is realised with a Python package called cv2, from OpenCV. The tool from OpenCV produces an intrinsic matrix $\boldsymbol{A}$, containing the intrinsic parameters, where f_x and f_y are focal lengths, and c_x and c_y are the optical centre.

$$\boldsymbol{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Figure 7 shows a graphic interpretation of the pinhole model. u and v represent "pixel coordinates" with origin in the upper left corner.

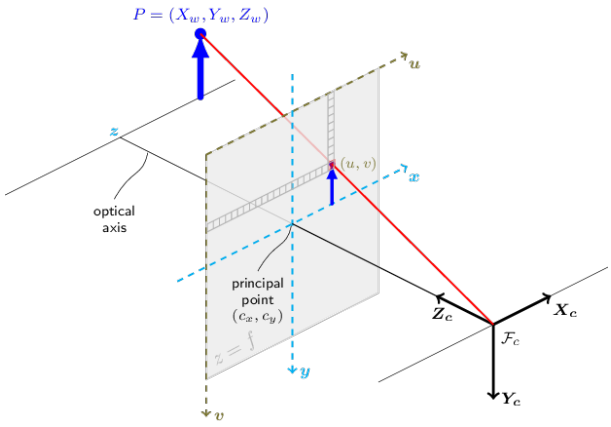

Fig. 7. Pinhole camera model. Figure from [18]

x and y represent the same information as u and v, but with origin at the optical centre (c_x, c_y), with the following property:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} u - c_x \\ v - c_y \end{bmatrix} \tag{2}$$

$\mathcal{F}_c$ is the origin of of the camera coordinate system and the projective centre [19]. P is a point given in world coordinates. The computation of the intrinsic matrix is done, in short, by compering several pictures of a checker board taken form different positions. The shape of the checker board is known, the squares are about $26mm \cdot 26mm$ and there is a 6 by 9 grid where two black corners meet. Figure 8 displays the checker board with highlighted corners. The checker board printed on to a piece of laminated A4 paper for extra stiffness, as the calibration relies on the checker board being flat.

Fig. 8. Checker board with highlighted corners

b) Extrinsic calibration:

It seems like the idea of the camera model is to convert real world 3D coordinates to 2D pixel coordinates, as described by the following equation 3 [18]

$$sp = A[R|t]P_w \tag{3}$$

where p is a 2D "pixel point", s is a scaling factor, R is a rotation matrix, t is a transnational vector and P_w is a 3D point similar to the point in figure 7.

The transnational vector and rotational matrix are obtained by extrinsic calibration. These extrinsic parameters are calculated with a similar method to the one used to find the intrinsic parameters. A tool from OpenCV is essentially used to calculate the position of the camera relative to a specific corner of a checker board, which is considered as the wold coordinate origin. The X_w, Y_w-plane lays on top of the checker board and the Z_w axis is normal to the checker board. Figure 9 shows the checker board with the "world origin" drawn on the lower right inner corner. The red circle is placed at optical centre, from the intrinsic calibration. The black border is a consequence of un-distortion, made possible by the intrinsic calibration.

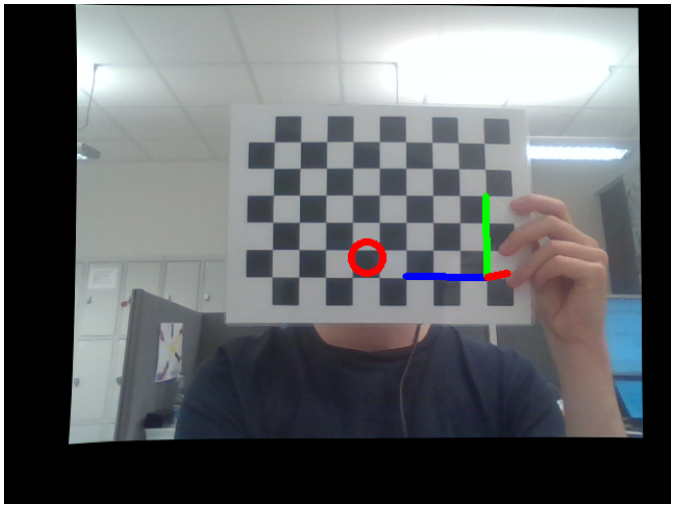

Fig. 9. Checker board with highlighted ”world origin” and a circle to mark the centre

c) Backprojection:

Converting a 3D point to a 2D point mathematically is not relevant for this project, as this happens automatically in the camera, the reverse process however is desired. The process of turning a 2D ”pixel point” to a 3D point is called backprojection [19]. The following equation [18] relates u and v to X_c, Y_c and Z_c:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \frac{X_c}{Z_c} + c_x \\ f_y \frac{Y_c}{Z_c} + c_y \end{bmatrix} \tag{4}$$

This can be rewritten to the following expression:

$$\begin{bmatrix} X_c \\ Y_c \end{bmatrix} = \begin{bmatrix} \frac{x}{f_x} \\ \frac{y}{f_y} \end{bmatrix} \cdot Z_c \tag{5}$$

Equation 5 can be used for backprojection by setting Z_c equal to the Z-component of the translational vector (III-D1c). This form of backprojection is illustrated in figure 10.

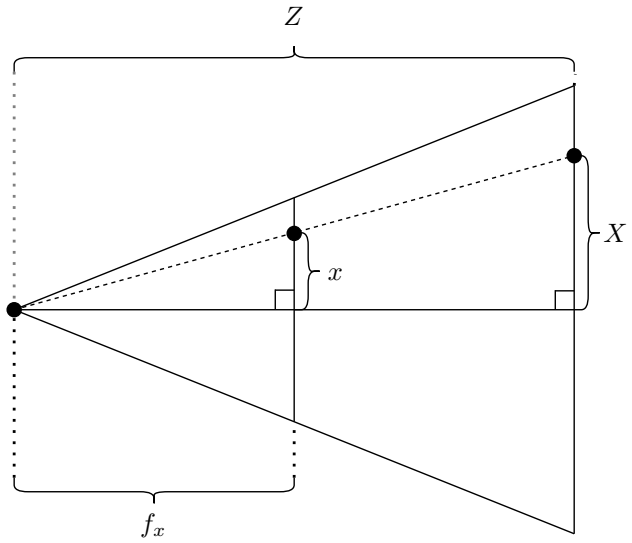

Fig. 10. Simple backprojection

This method produces X and Y values on a plane parallel to the imaging-plane, which would work fine if the camera is mounted normal to the surface of interest (for example the ground), but becomes problematic when the imaging plane is not parallel to the surface of interest. This non-parallel plane, or surface, is illustrated as a grey line in figure 11. This method was attempted in the same program as the extrinsic calibration, and the estimated position to the ”world origin”, displayed in figure 9, is approximately $[116mm, 19mm, 614mm]$. This seems plausible.

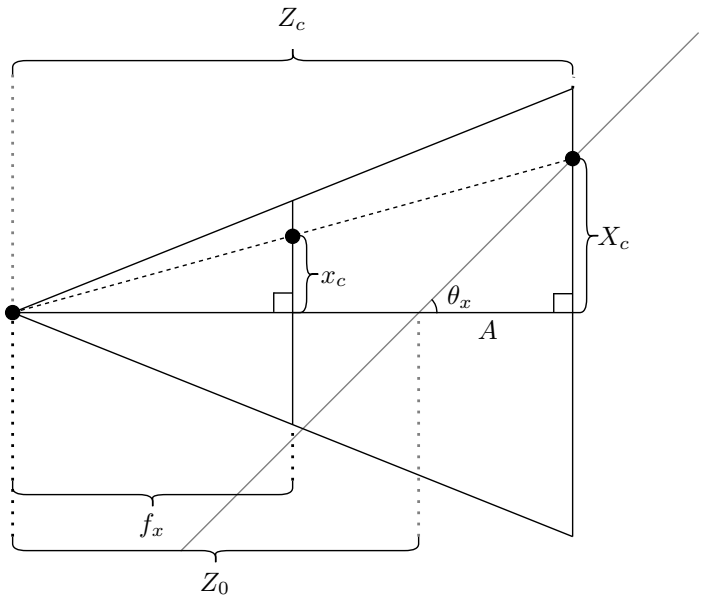

Fig. 11. Backprojection with non-parallel surface

Obtaining the value of Z_0 is of interest for later calculations. The following expression can be derived from figure 11:

$$Z_0 = Z_c - A \tag{6}$$

Note that this A is not the intrinsic matrix. A is found:

$$\tan(\theta_x) = \frac{X_c}{A} \quad \Rightarrow \quad A = \frac{X_c}{\tan(\theta_x)} \tag{7}$$

Equation 6 is rewritten:

$$Z_0 = Z_c - \frac{X_c}{\tan(\theta_x)} \tag{8}$$

It is desirable to obtain expressions for X and Z such that the point they describe ends up on the grey line, as illustrated by $X(x)$ and $Z(x)$ in figure 12. The grey point in this figure (Z_0, X_0) shows the point that would have been produced if equation 5 was applied.

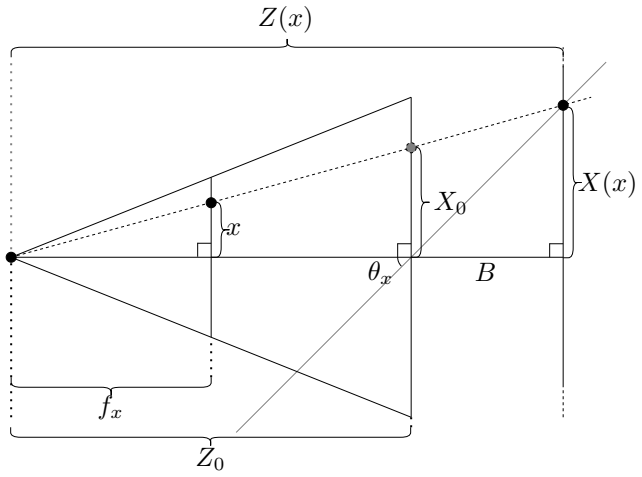

Fig. 12. Finding $Z(x)$ and $X(x)$

The following expression can be derived from figure 12:

$$Z(x) = Z_0 + B \tag{9}$$

B is found:

$$\tan(\theta_x) = \frac{X(x)}{B} \;\Rightarrow\; B = \frac{X(x)}{\tan(\theta_x)} \tag{10}$$

Formality of triangles is used:

$$\frac{X(x)}{Z(x)} = \frac{x}{f_x} \;\Rightarrow\; X(x) = Z(x)\frac{x}{f_x} \tag{11}$$

Equation 9, 10 and 11 is combined in order to find $Z(x)$:

$$\begin{aligned} Z(x) = Z_0 + \frac{X(x)}{\tan(\theta_x)} \;\Rightarrow\; Z(x) &= Z_0 + \frac{x}{f_x}\cdot\frac{Z(x)}{\tan(\theta_x)} \\ Z(x)(1-\frac{x}{f_x\tan(\theta_x)}) &= Z_0 \\ Z(x) = \frac{Z_0}{1-\frac{x}{f_x\tan(\theta_x)}} & \end{aligned} \tag{12}$$

Then equation 11 and 12 are combined to find $X(x)$:

$$X(x) = \frac{Z_0\frac{x}{f_x}}{1-\frac{x}{f_x\tan(\theta_x)}} \;\Rightarrow\; X(x) = \frac{Z_0}{\frac{x}{f_x}-\frac{1}{\tan(\theta_x)}} \tag{13}$$

The following property is assumed:

$$Z(x) = Z(y) \tag{14}$$

$Y(y)$ should have a similar structure to $X(x)$:

$$Y(y) = \frac{Z_0}{\frac{y}{f_y}-\frac{1}{\tan(\theta_y)}} \tag{15}$$

Lastly, equation 12, 13 and 15 are combined to:

$$\begin{bmatrix} X(x) \\ Y(y) \\ Z(x) \end{bmatrix} = \begin{bmatrix} (\frac{x}{f_x}-\frac{1}{\tan(\theta_x)})^{-1} \\ (\frac{y}{f_y}-\frac{1}{\tan(\theta_y)})^{-1} \\ (1-\frac{x}{f_x\tan(\theta_x)})^{-1} \end{bmatrix}\cdot Z_0 \tag{16}$$

2) Extracting Information from Depth Image: LiDAR and camera combination sensors are able to provide a depth image along with a normal colour image. This depth image is a 2D array of size $(n \times m)$ where each item of the array contains a value between 0 and 255 based on the spatial distance from the camera to the appropriate points in the surrounding room. It is possible to extract the distance from the camera lens to the real point, by relating an (x, y)-coordinate in the colour image to an (x, y)-coordinate in the depth image. The distance information extracted from this depth image can be used to calculate the real world coordinate of the point. The method for extracting this information is shown for x, but is assumed to be identical for y. Looking at figure 13, the depth information provided by the depth image is represented by Z.

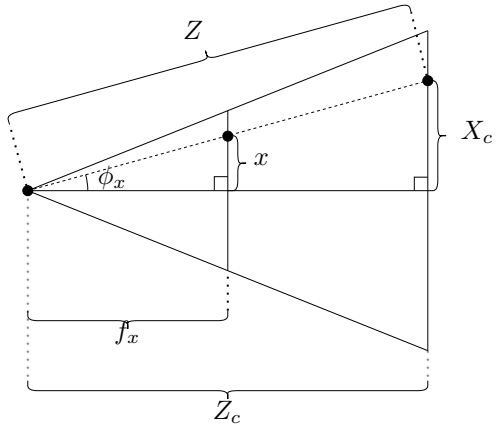

Fig. 13. Pinhole camera model illustrating the depth value Z provided by the depth image. Explanation: ϕ_x: angle between img center and line Z, x: coordinate of point in image relative to image centre. f_x: Focal length in x-direction, X_c: real world X coordinate of point relative to camera, Z_c: real world Z coordinate of point relative to camera.

Looking at figure 7, the coordinate x is found as $x = u - c_x$ where: u is pixel "x-direction" coordinate of point in image and c_x is the x coordinate of image centre. c_x is gathered from the intrinsic camera calibration described in section III-D1a. The angle ϕ_x is then found using equation 17:

$$\phi_x = \arccos\left(\frac{f_x}{\sqrt{f_x^2 + x^2}}\right) \tag{17}$$

As the distance Z is gathered from the depth image, coordinate X_c and Z_c is then found using equation 18:

$$\begin{bmatrix} Z_{c1} \\ X_c \end{bmatrix} = Z \cdot \begin{bmatrix} \cos\phi_x \\ \sin\phi_x \end{bmatrix} \tag{18}$$

Notice that the Z_c from equation 18 is denoted as Z_{c1}. This is due to the fact that Z_c is calculated for both $x(Z_{c1})$ and $y(Z_{c2})$. These two values are then combined to an average value Z_c. Doing this procedure for both x and y gives $[X_c, Y_c, Z_c]$ which is the coordinate of the given point relative to the center of the camera focal plane.

E. Actual Implementation Between Lane Detection and ROS2

Implementing the proposed camera calibration methods proved to be challenging. Furthermore, implementing this

lane detection with the Navigation2 system in ROS2 would require high knowledge around the Navigation2 API, and more time. This resulted in an alternative method for implementing lane detection and ROS2 in order to achieve autonomous lane keeping. This section describes how this implementation between the lane detection and ROS2 was done.

Assuming the lane detection algorithm returns an accurate image coordinate that describes the centre of the road lane, a simple algorithm can be made in order to keep this point in the centre of the vision systems perception. The UGV responds to a velocity command on the ROS2 topic $/cmd\_vel$. For a differential drive robot, the velocity command would consist of a forward velocity $X_{vel}[\frac{m}{s}]$ and an angular velocity about the z-axis $z_{ang}[\frac{rad}{s}]$. This method relies on providing the UGV's velocity controller a velocity command with a constant forward velocity of n $\frac{m}{s}$ and then varying the angular velocity based on the reported x position of the lane centre from the lane detection algorithm. The angular velocity is decided by equation 19, where $x[pix]$ is the x coordinate of the detected lane centre relative to image centre($x = u - c_x$), $x_{max}[pix]$ is the image width and $\omega_0[-]$ is a scaling factor.

$$z_{ang}(x) = \omega_0 \cdot \frac{x}{x_{max}} \tag{19}$$

The lane detection algorithm along with equation 19 is then implemented into algorithm 2, in order to communicate with the robot.

Algorithm 2 Algorithm for following lanes

Input: image $(n\ X\ m\ X\ 3)$
Output: $X_{vel}[m/s]$, $z_{ang}[rad/s]$
1: $img_{processed}, mass[u, v] = lane\_detect(image)$ (Alg. 1)
2: x = u - c_x (c_x = A(1,3), sec III-D1a)
3: X_{vel} = n
4: $z_{ang} = \omega_0 \cdot \frac{x}{x_{max}}$ (eq. 19)
5: vel_cmd.x_linear = x_{vel}
6: vel_cmd.z_angular = z_{ang}
7: Publish vel_cmd to UGV velocity controller
8: Show $img_{processed}$ on screen for visualisation.
9: **return** $X_{vel}[m/s]$, $z_{ang}[rad/s]$

IV. Results & Discussion

A. Lane Detection

1) CNN Lane Detection Model: Verification and testing is done by providing various images to the machine learning model and analysing the results. Figure 14 and 15 illustrates how the CNN model performs on roads with well defined markings and a newly paved road

Fig. 14. Input image to lane detection model. This image has a newly paved road with well defined markings.

Fig. 15. Output of lane detection model when presented with figure 14

Figure 16 and 17 shows how the model preforms on more worn down roads.

Fig. 16. Input image to lane detection model. This image has an older road.

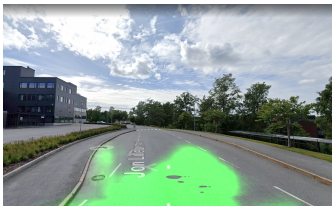

Fig. 17. Output of lane detection model when presented with figure 16

The figures above demonstrates the robustness of the model. It seems like the model preforms relatively well on roads with clearly defined markings, but struggles on more worn roads and roads where the lines are less predominant.

2) Lab Testing: A test was conducted where the CNN lane detection algorithm described in section 1 was implemented with ROS using the implementation described in section III-E. The test was done in a machine lab where the autonomous lane keeping system was deployed with the UGV standing between some floor markings. The UGV was able to follow lane markings autonomously to some extent, but would sometimes move outside the lane to the point of no return. The algorithm has no indication on the detection of a lane, so the UGV would keep going in an uncontrolled manner. Figure 18 is an image taken during testing. It is possible to see the green lane detected on the computer screen. At this instance, the UGV was autonomously navigating between the lines.

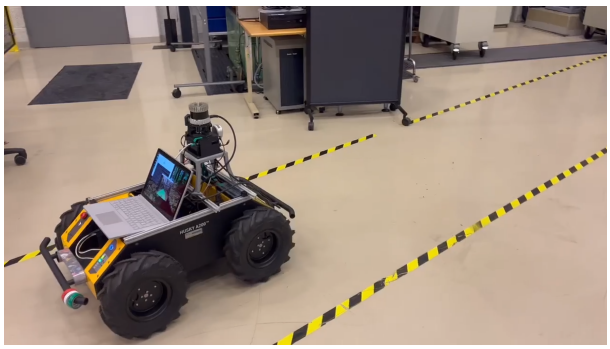

Fig. 18. Lab testing of autonomous lane keeping

The test was preformed with the UGV moving at $0.2[\frac{m}{s}]$ as it was thought that the probability of successful lane keeping would be higher for lower velocities. The lane detection model

is trained to detect road lanes and not markings on a floor, even though it was tested on floor markings. It seemed that the vision system struggled with reflections of the roof lighting on the floor as it was a fairly reflective surface compared to a real road. It would be preferable to do this test on a real road outside.

V. Conclusion

A Husky UGV has been set up to autonomously navigate an indoor environment with Robot operating system 2 (ROS2). The system uses SLAM Toolbox to localise and generate a 2D map of the environment, and Navigation2 to implement navigation algorithms. Two lane detection algorithms is described, one using classic machine vision methods and one method using a machine learning model. Camera calibration is done and two different methods of getting depth from the vision data is described. One method involves using a depth image from LiDAR data and converting this information to XYZ-coordinates. The other method estimates the image depth based on backprojection, which takes the distance and angles of the camera in to account. Implementation between Navigation2 and the Lane Detection algorithms was not established. Therefore, an alternative way to achieve autonomous lane keeping is proposed. The alternative method involves the UGV driving at a constant speed while running a CNN lane detection algorithm. As the algorithm detects lanes, the UGV will try to keep the centre of the lane in the centre of its vision by turning as it goes. This method proved to work fairly well as the UGV was able to stay within the markings on the floor in a machine lab to some extent.

VI. Future Work

A. Lane Detection

The lane detection algorithms could benefit from having some kind of indication of lane detection. As it stands, the algorithm has no way of deciding whether it sees a road or not and the systems will report coordinates no matter what.

1) Traditional Lane Detection: The traditional lane detection algorithm described in section III-C1, has not been tested on the UGV.

2) CNN Lane Detection: The CNN lane detection algorithm implements a pre-trained model with unknown training conditions. This means that it is not known what data this model is trained on nor the training parameters of the model. It is believed that more work could be done on this area to achieve better results from the CNN lane detection system.

B. Extracting Real Coordinates from Image

1) Rotation of Plane: Extracting 3D coordinates by rotating the imaging plane and thus generating a dynamic value for depth, as explained in section III-D1c, is a theoretical method that is yet to be tested. This method assumes that the ground visible to the vision system is flat relative to the UGV. In this case: that the ground in front of the UGV is parallel to the ground under the UGV. It is believed that this method would then be able to estimate the spatial coordinate of a point on the ground. It is also assumed that there would be inaccuracies, in this case, if the UGV is for example at the foot of a steep hill. This method could possibly be implemented with variable angles.

2) Estimating 3D coordinates from Depth Image: The 3D coordinate extraction method described in section III-D2 relies on accurate readings from a depth image and a fused camera and LiDAR setup. It is assumed that by fusing a powerful LiDAR with a camera, this method might be possible.

C. Interaction between Lane Detection Algorithm and UGV navigation system

Getting the lane detection algorithm to interact with Navigation2 is thought to bring some advantages. Compared to the implemented solution with a velocity command directly to the UGV velocity controller, Navigation2 would add global and local path planning, obstacle avoidance and mapping to the lane keeping system.

References

[1] Husky group documentation. https://mas514-husky_group.gitlab.io/mas-514-husky/. Accessed: 2022-10-31.

[2] Qiang Li, Ranyang Li, Kaifan Ji, and Wei Dai. Kalman filter and its application. In *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pages 74–77, 2015.

[3] Qin Zou, Qin Sun, Long Chen, Bu Nie, and Qingquan Li. A comparative analysis of lidar slam-based indoor navigation for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6907–6921, 2022.

[4] Illah Reza Nourbakhsh, Rob Powers, and Stan Birchfield. Dervish - an office-navigating robot. *AI Mag.*, 16:53–60, 1995.

[5] Wolfram Burgard, Armin Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. pages 11–18, 01 1998.

[6] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: a second-generation museum tour-guide robot. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 3, pages 1999–2005 vol.3, 1999.

[7] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. A navigation system for robots operating in crowded urban environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 3225–3232, 2013.

[8] Steven Macenski, Francisco Martin, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[9] Li Haixia and Li Xizhou. Flexible lane detection using cnns. In *2021 International Conference on Computer Technology and Media Convergence Design (CTMCD)*, pages 235–238, 2021.

[10] Li Haixia and Li Xizhou. Flexible lane detection using cnns. In *2021 International Conference on Computer Technology and Media Convergence Design (CTMCD)*, pages 235–238, 2021.

[11] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando A. Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving. *CoRR*, abs/1504.01716, 2015.

[12] Clearpath Robotics. Husky. Available at https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/ (2022/09/26).

[13] NVIDIA Corporation. Jetson agx xavier developer kit. Available at https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit (2022/09/26).

[14] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[15] Paul Bovbel, Chris Lalancette, Michel Hidalgo, Daisuke Nishimatsu, Rein Appeldoorn, Yoshimaru Tanaka, Mikael Arguedas, and Carlos Andres Álvarez Restrepo. Ros 2 pointcloud ¡-¿ laserscan converters. https://github.com/ros-perception/pointcloud_to_laserscan, 2022.

[16] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6:2783, 05 2021.
[17] Algobotics. Lane detection using a deep learning model — hands-on tutorial, July 2021.
[18] Open CV. Camera calibration and 3d reconstruction. Available at https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga61585db663d9da06b68e70cfbf6a1eac (17.11.2022), November 2022.
[19] Thomas Opsahl. The perspective camera model. Available at https://www.uio.no/studier/emner/matnat/its/TEK5030/v19/lect/lecture_1_2_the_perspective_camera_model.pdf (17.11.2022).