This paper presents a detailed overview of the software architecture and algorithms employed in a 1-to-10 scale autonomous driving vehicle designed for the Bosch Future Mobility Challenge (BFMC). The competition focuses on undergraduate and master's students developing algorithms for autonomous vehicle control in a simulated city-like environment, using 1/10-scale model cars provided by the organizers.

Key algorithms discussed include PID control for speed and direction regulation, Hough transform and Histogram filter for lane detection, and odometry and Kalman filter for vehicle localization. Additionally, the YOLO object detection system identifies and tracks objects on the road. These algorithms contribute to various functionalities, such as lane following, path planning, sign detection, turning, and parking maneuvers.

The paper highlights the potential of self-driving technology to revolutionize the transportation industry, while acknowledging the challenges and limitations of self-driving car algorithms. The BFMC aims to promote innovation and encourage the development of new technologies related to autonomous driving. The model kit provided by Bosch consists of a pre-assembled 1:10 vehicle model with essential components for autonomous driving, as well as low-level code for control and brain unit boards.

The software architecture for the autonomous driving vehicle is built on the ROS framework, which facilitates communication, control, and coordination between different components of a robotic system. The software system is divided into four sub-components or layers: Sensing and Input, Perception and Scene Understanding, State Machine, and Vehicle Control and Actuation.

Sensing and Input: This layer collects data from various sensors installed on the vehicle. Key nodes include the Camera handler, which publishes raw and depth images, the IMU unit that provides orientation data, and the encoder unit, which publishes the velocity/RPM of the motor. Additional nodes for GPS, vehicle-to-vehicle communication, and traffic light detection are included for use during the competition.

Perception and Scene Understanding: This layer interprets the data collected by the Sensing and Input layer. Techniques such as Hough transform and histogram filter are applied to detect lane markings, intersections, and dotted lines. Traffic signs, pedestrians, and other vehicles are identified through a trained machine learning classifier model. Odometry-based localization is achieved using the IMU and encoder nodes, providing information about the vehicle's position and orientation.

State Machine: The state machine node subscribes to the four perception nodes and synchronizes them using an approximate time synchronizer. The driving task is broken down into several states with unique behavioral systems, which can switch between one another through event triggers such as traffic sign detection or intersection detection. The vehicle follows lanes, reacts to traffic signs, and makes appropriate path decisions at intersections.

Vehicle Control and Actuation: This layer is not detailed in the provided text, but it likely handles control commands and actuation of the vehicle's components based on information from the previous layers.

The primary procedure for the autonomous vehicle's movement is lane following, which relies on lane detection. Two methods for lane detection were tested: Hough Transform and Histogram Filter.

Hough Transform: This method involves converting the image to grayscale, blurring it, applying a Canny edge detector, masking the area of interest, and using Houghline transform to obtain line segment coordinates. The detected lines are classified into left lane, right lane, and stop line based on their slopes. This method's advantages include low inference time and adaptability to changes in environment lighting and image brightness. However, it struggles with large curves, dotted lanes, and pedestrian crossing areas.

Histogram Filter: This method starts similarly to Hough Transform but instead applies binary thresholding and sums up column pixel values to obtain a row matrix. Delimiters for lane markings are found where the weights switch from high to low. This method performs better than Hough Transform for most situations, but it may be sensitive to lighting changes. To mitigate this, binary thresholding value is varied according to the maximum pixel value of the processed image, resulting in a detection system independent of lighting conditions.

Lane following: Each frame of image data is processed to determine the lane center's position and whether a stop line was detected. The vehicle's error offset with respect to the lane center is calculated, and a PID controller is used to determine the steering command that will correct the car's position. Issues faced include fluctuating steering due to failure in lane detection, and camera position causing confusion between left and right lanes in sharp curves. A filter is applied on the detection result to ensure correct lane center output for most image frames.

The task of sign detection in traffic scenes involves identifying various objects, such as traffic signs, lights, pedestrians, and vehicles. A custom dataset was created to include these objects, with traffic signs obtained from the BFMC documentation page and vehicle images from the Stanford car dataset. A series of preprocessing techniques were employed to generate multiple images from each original image, applying fog, rain, snow, and shadow effects to enhance the dataset.

The dataset was used to simulate real-world traffic scenarios, appending random images to background images of highway roads. A total of 16,000 images were generated, containing various sign objects, traffic lights, pedestrians, and vehicles against different backgrounds and scales. The dataset was split into 80% training data, 10% validation, and 10% testing.

Using PyTorch and the YOLOV5 algorithm, an object detection algorithm that employs a convolutional neural network (CNN), the dataset was trained for real-time object detection in images. The algorithm divides the input image into smaller grid-like boxes, detects objects within each box, and draws bounding boxes around each object. The Intersection Over Union (IOU) metric is used to calculate the accuracy of the model's predictions.

The training results achieved a precision above 95% for all classes and an mAP@0.5:0.95 of 95.57%, indicating excellent performance. The mAP (mean Average Precision) metric considers both precision and recall, with the mAP score calculated by averaging the precision-recall curves over all classes and computing the area under the curve (AUC). The mAP@0.5:0.95 score is the mAP score computed at a range of IoU thresholds from 0.5 to 0.95 with a step size of 0.05.

VII. INTERSECTION MANEUVERS

A. Approach

The autonomous vehicle system detects signs indicating an upcoming intersection and transitions to the approaching intersection state. If a stop sign or red light is detected, the vehicle stops at the intersection for three seconds before entering the intersection maneuvering state. If a priority sign and green light are detected, the system enters the intersection maneuvering state directly.

B. Trajectory

The system fits a trajectory using an exponential function, considering the vehicle's initial orientation. Matrix multiplication between the vehicle's poses and a rotation matrix ensures the vehicle follows the correct path through the intersection.

C. Odometry

The vehicle's position and orientation are estimated using finite difference approximation and updated pose estimates. The estimated position is compared to the desired position, and the error is fed into a fine-tuned PID controller, adjusting the steering angle to keep the vehicle on the desired trajectory.

VIII. LOCALIZATION

A. Sensor Data

The vehicle is equipped with sensors (IMU and encoder) used for localization. Data is acquired and synchronized through ROS nodes.

B. Extended Kalman Filter

The ROS package robot localisation provides a built-in extended Kalman filter (EKF) to integrate sensor measurements with GPS data. The EKF estimates the state of the system by maintaining a Gaussian probability distribution over the state and iteratively updating the mean and covariance of the noise.

IX. NAVIGATION

A. Approach

The map configuration is modeled as a directed graph, with locations as nodes and possible paths as edges, to perform general map position localization and compute paths between locations using shortest path algorithms.

B. Map

Distinct sections or locations of the map are defined and grouped, with their general position coordinates and perimeter positions. Using NetworkX, nodes are initialized, and directed edges are added based on standard driving rules.

C. Path and Decision Points

Edges have length parameters and decision parameters (left, right, or straight). The path planner takes starting and destination nodes, using Dijkstra's shortest path algorithm to output a list of all node locations and decision points for the vehicle to follow.