2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-7281-6212-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/IROS45743.2020.9341770

Go-CHART: A miniature remotely accessible self-driving car robot

Shenbagaraj Kannapiran and Spring Berman

***Abstract*— The Go-CHART is a four-wheel, skid-steer robot that resembles a 1:28 scale standard commercial sedan. It is equipped with an onboard sensor suite and both onboard and external computers that replicate many of the sensing and computation capabilities of a full-size autonomous vehicle. The Go-CHART can autonomously navigate a small-scale traffic testbed, responding to its sensor input with programmed controllers. Alternatively, it can be remotely driven by a user who views the testbed through the robot's four camera feeds, which facilitates safe, controlled experiments on driver interactions with driverless vehicles. We demonstrate the Go-CHART's ability to perform lane tracking and detection of traffic signs, traffic signals, and other Go-CHARTs in real-time, utilizing an external GPU that runs computationally intensive computer vision and deep learning algorithms.**

I. Introduction

In this paper, we propose a miniature mobile robot that can emulate sensing and computation capabilities of a full-size autonomous vehicle, with the aid of an external GPU for computationally intensive tasks. The Go-CHART can also be driven remotely by a user with first-person views of the robot's environment through video feeds from its cameras. The Go-CHART can be used to conduct studies on conditions and challenges that are likely to be experienced by real autonomous vehicles, but are too difficult to realistically replicate in a driving simulator (e.g., crash scenarios with complex vehicle dynamics and impacts on surroundings, dynamic lighting and weather conditions) and too risky to physically implement at full scale. The Go-CHART can be used to investigate interactions between human drivers and driverless vehicles; perform studies on cooperative driving strategies; and test computer vision, deep learning, and control algorithms for autonomous vehicles. It can also be used to collect a wide range of data sets in scenarios that are difficult to reproduce on full-size vehicles for training deep learning algorithms.

Deep learning and computer vision play a crucial role in controlling autonomous vehicles and are essential for the vehicle to exhibit higher levels of autonomy. However, existing miniature driving testbeds lack the capabilities to support an external GPU. In this work, our contributions include the design and development of the Go-CHART, a small four-wheel skid-steer robot that resembles a standard commercial sedan. It is equipped with an ultra-wide night vision front camera, side-view and rear-view cameras, an ultrasonic sensor, and a 2D LiDAR sensor. The Go-CHART is modular and can be upgraded with updated microcontrollers and microprocessors, as well as stereo cameras. It utilizes a system architecture that supports an external GPU capable of handling computationally intensive deep learning and machine learning algorithms. In autonomous mode, the Go-CHART is controlled by on-board Raspberry Pi computers and Teensy microcontrollers, and external computers that replicate many of the sensing and computation capabilities of a full-size AV; for example, lane tracking and object detection using deep learning algorithms. We have also developed a graphical user interface that enables a user to remotely drive the Go-CHART using a steering wheel and responsive pedals. Additionally, we have developed a small-scale testbed that resembles a U.S. driving environment, including roads, traffic lights, signs, and other miniature scenery such as trees, which not only increase the realism of the testbed for human-robot interaction experiments, but also provide feature points in the environment for performing experiments on monocular simultaneous localization and mapping (SLAM). In this paper, we experimentally demonstrate the lane-tracking, object detection, and decision-making capabilities of the Go-CHART on this testbed.

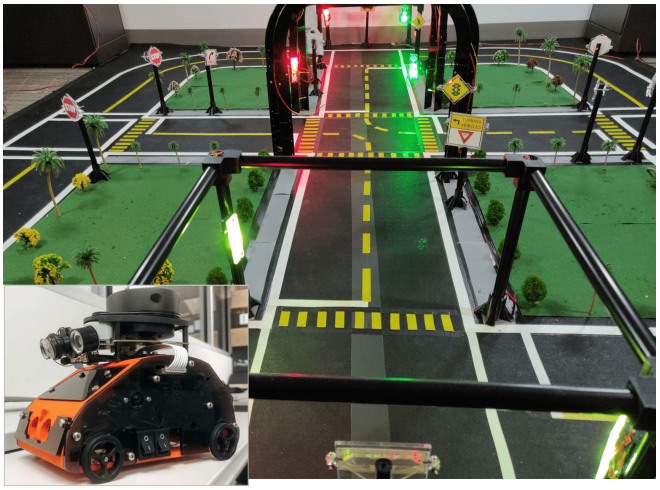


Fig. 1. Small-scale driving testbed and a Go-CHART robot (inset).

II. Existing Platforms

The recently accelerated development of self-driving cars has motivated the need to validate controllers for these vehicles on experimental testbeds. Several full-scale testbeds with 5G-enabled infrastructure have been implemented for testing connected and autonomous vehicles, including University of Michigan's Mcity [18], South Korea's K-City [19], and UK's Millbrook Proving Ground [20]. In addition, Google

This work was supported by the ASU Global Security Initiative.

Shenbagaraj Kannapiran and Spring Berman are with the School for Engineering of Matter, Transport and Energy, Arizona State University (ASU), Tempe, AZ 85281, USA {shenbagaraj, spring.berman}@asu.edu

has created a miniature Google Street View in tiny realistic reproductions of various international sites and cities [17].

The paper [1] includes a list of robots costing less than US $300 that were developed within the last 10 years from the date of its publication. The work proposed a small educational robot that can drive autonomously through a traffic testbed. However, the robot lacks the sensors and processing capabilities that facilitate the implementation of decentralized robot controllers and computation-intensive deep learning algorithms.

The University of Delaware Scaled Smart City [8] is a small-scale testbed that has been used to replicate real-life traffic scenarios and optimize control strategies on mobile systems. The testbed uses a Vicon motion capture system to localize and coordinate 35 vehicles simultaneously in a centralized manner.

The paper [3] proposes a traffic testbed with a fleet of miniature vehicles with Ackermann steering. The testbed supports cooperative driving studies and multi-vehicle navigation and trajectory planning. This paper includes a list of miniature robots that also use Ackermann steering. However, the robotic platform used in [3] does not incorporate vision or range sensors and lacks the processing capacity to run computationally intensive algorithms.

Several open-source and commercially available miniature autonomous vehicle robots with onboard GPUs and vision-based control have been developed in recent years. The MIT Racecar [5], MuSHR [21], and Donkey Car [23] are open-source self-driving race car platforms with a range of onboard processors and sensors. The AWS DeepRacer [22] and NVIDIA Jetbot [24] are commercially available robots with Ackermann and differential-drive steering, respectively. However, all of these platforms lack 360° camera views, and their restricted onboard processing capabilities preclude the implementation of complex deep learning techniques.

In [9], it was demonstrated that small differential-drive robots [2] can exhibit certain autonomous functions on a driving testbed such as lane tracking and traffic light detection. However, the limited onboard processing power of the robot caused delays in image processing, resulting in errors in navigation. Other differential-drive robots such as [6], [7] can also emulate particular functions of autonomous vehicles, but lack the processing power necessary for real-time navigation and object detection.

III. Go-CHART Design and Capabilities

A. Mechanical design and circuit boards

Figures 2 and 3 show SolidWorks renderings of the Go-CHART, with its components labeled in the exploded view in Fig. 2. The Go-CHART is built from both off-the-shelf and custom-built components such as custom PCBs, 3D printed parts, and laser cut acrylic parts. As described in Section III-D, we use the GPU on the NVIDIA Jetson TX2 module, which costs around US $500. However, the choice of GPU is flexible, depending on the application. Table I lists the primary internal components of the robot and their costs, which total to about US $264. Table II lists the external hardware components of the Go-CHART, which cost approximately US $513 in total (excluding the network router, which can be used with multiple robots).

TABLE I
Internal Components of Go-CHART Robot

Components	Number of Units	Total Cost (US$)
Raspberry Pi 3B	1	32
Raspberry Pi Zero	3	15
RPLIDAR A1 2D LiDAR	1	99
Raspberry Pi camera	3	15
Raspberry Pi ultra-wide angle night vision camera	1	11
Teensy 3.2	1	20
TB6612 Adafruit motor driver	2	10
Polulu mini motors with encoders	4	36
SD cards (16GB)	4	16
Miscellaneous components	-	10

TABLE II
External Components of Go-CHART Robot

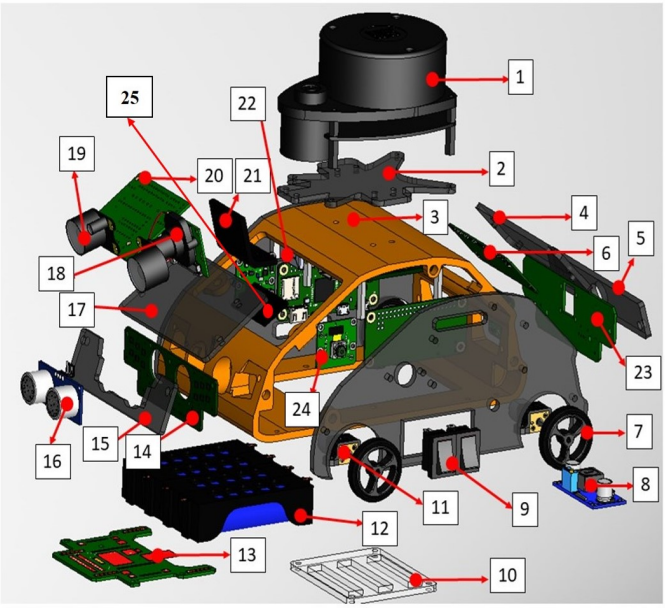
Components	Number of Units	Total Cost (US$)
External GPU (1 per robot)	1	500
LAN cables (2 ft) (1 per robot)	1	3
Miscellaneous components (per robot)	-	10
Dual-band router (1 per ∼60 robots)	1	200

The body of the Go-CHART is a 1:28 model of a standard commercial sedan and features a modular design, allowing for easy replacement and addition of components as necessary. The main body, labeled 3 in Fig. 2, is a single 3D-printed piece, while the remaining parts of the frame are made from laser cut acrylic.

Custom printed circuit boards (PCBs) were designed in order to densely populate all the circuitry within the limited volume of the Go-CHART. The PCBs provide several voltage levels (3.3V, 5V, and 12V) that can accommodate a wide variety of sensors, microcontrollers, and microprocessors. The PCBs have open ports for additional sensors, motor drivers, and microcontrollers as needed. Four custom PCBs were fabricated: (1) a power supply board, which includes the voltage regulators and the microcontroller board; (2) a motor driver board, which includes the motor drivers and all sensors; and (3) front and (4) back LED boards, which control the headlights, tail lights, and turn indicator lights.

B. Drive mechanism

The Go-CHART uses a four-wheel skid steering system, which is powered by four standard micro metal gear motors with a 51.45:1 gear ratio. The motors drive 32-mm-diameter wheels and have Hall effect encoders with a linear resolution of 0.163 mm/tick, enabling the Go-CHART to move at speeds between 4 cm/s and 42 cm/s. Two Adafruit TB6612 motor drivers control each motor separately, using a PD controller to limit the overshoot and dampen the oscillations in the robot's heading angle.



Serial No.	Component	Serial No.	Component
1	2D LiDAR	14	Front LED PCB
2	LiDAR and camera holder	15	Front light glass
3	Go-CHART main body	16	Ultrasonic sensor
4	Rear window glass	17	Front window glass
5	Rear light glass	18	Ultra-wide view camera (night vision)
6	RPi zero for rear and side-view cameras	19	UV blaster
7	Wheels	20	Power board
8	Buck converter	21	Camera holder
9	Rocker switches	22	RPi3B
10	Battery holder plate	23	Rear LED PCB
11	51.45:1 mini motor with encoder	24	RPi camera- rear and side
12	18650 battery	25	Teensy 3.2
13	Motor driver PCB		

Fig. 2. Exploded SolidWorks rendering of Go-CHART, with components labeled.

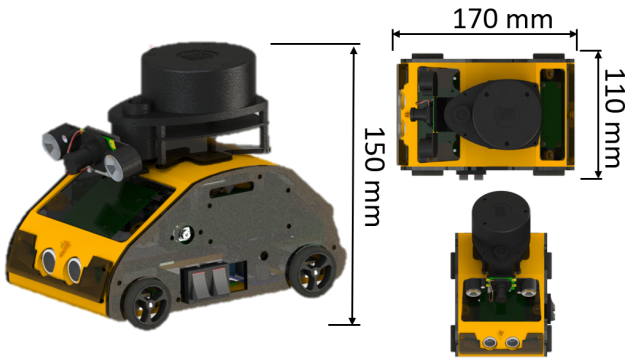


Fig. 3. SolidWorks renderings of isometric view, top view, and front view of Go-CHART.

C. Power supply

The Go-CHART has a custom-built battery pack with four 18650 batteries that deliver 7.4V at 7000 mAh, which provides enough power to drive the robot for almost an hour. The battery pack circuit is composed of four batteries, a 2S battery management system (BMS), the battery protection circuit, voltage regulators, a buck converter, fuses, and 3.5 mm barrel connectors.

D. Computational capabilities

The Go-CHART is equipped with commercially available microprocessors and microcontrollers that are supported by extensive online resources. High-level control and front-view camera video streaming are performed by one Raspberry Pi 3 Model B (RPi3B) computer (1.2 GHz quad-core ARM Cortex-A53 with 1GB LPDDR2) onboard the robot and three RPi Zero W computers (1 GHz single-core CPU, 512 MB RAM) dedicated for video streaming from the side and rear-view cameras. The RPi3B runs Ubuntu MATE with the Robot Operating System (ROS) and the OpenCV library [16] installed. The RPi Zero boards run the Raspbian OS and are upgraded with external dual band USB wireless network adapters, soldered directly to the motherboard, to enhance video streaming performance. Low-level control of actuation and sensor data processing are performed by a Teensy 3.2 microcontroller (MK20DX256VLH7 processor with ARM Cortex-M4 core running at 96 MHz with 64kB RAM), which can accommodate a large number of PWM and digital pins.

In addition to these onboard components, an external GPU is used to train and implement the neural network models for the deep learning algorithms described in Section V. The external GPU's capacity can be varied as needed, based on the testing requirements. We used a GeForce RTX 2080 Ti GPU to train the neural network models. To keep the system low-cost and portable, we used a low-power NVIDIA Jetson TX2 module with 256-core Pascal™ GPU architecture to execute the deep learning algorithms using the trained model.

E. Communication capabilities

Experiments on cooperative driving and multi-robot control will often require ad hoc communication between pairs of Go-CHARTs. To implement this communication, the RPi3B onboard the Go-CHART can act as a Wifi router node, enabling limited-bandwidth, two-way information transfer with a nearby robot. Since this is a short-range weak signal with limited bandwidth, usage of the external GPU is not possible (higher bandwidth is required for video streaming with minimal latency), which prevents the implementation of deep learning algorithms that require the GPU. The Go-CHART can also operate independently, without communicating with other Go-CHARTs. In this case, the robot must have access to the external GPU in order to run controllers that utilize deep learning algorithms.

In addition, multiple Go-CHARTs can be connected to a common dual band router (multiple routers in the case of a large testbed environment), which are in turn connected to the same LAN network and operate at different channels between 2.4 GHz and 5 GHz so as to reduce interference between the routers and increase bandwidth to accommodate multiple video streams from the Go-CHARTs. The external Jetson TX2 GPU (one for each Go-CHART) is manually connected to the routers through the LAN to reduce wireless network traffic, and the video stream is processed on the external GPU.

F. Onboard sensors

The Go-CHART integrates a variety of vision and range sensors, shown in Fig. 4, to enable lane tracking, object detection, collision avoidance, and mapping. There are four RPi cameras (Sony IMX219 8-megapixel sensor) on the robot: one ultra-wide angle front-view camera with night vision, one camera on each side of the robot, and one rear-view camera. The front-view camera, which has a viewing angle of 160°, is mounted 9 cm above the bottom of the Go-CHART main body and is tilted downward by an angle of 30°. This placement was chosen so that the camera's field of view contains the entirety of a 30-cm-tall traffic light when the robot is at least 20 cm from the light, so that the robot can still detect the light when it drives forward about one-third of the way into an intersection on the testbed. An HC-SR04 ultrasonic distance sensor with a range of 2–400 cm and a resolution of 0.3 cm is mounted on the front of the robot, and a 2D 360° LiDAR (RPLIDAR A1) is mounted on top. The measurements from the ultrasonic sensor and LiDAR are used to detect nearby obstacles and initiate collision avoidance maneuvers, and the LiDAR also enables 2D mapping of the local environment. In addition, the Go-CHART includes an LDR (Light Dependent Resistor) sensor for brightness correction of the front-view camera images, an IMU (Inertial Measurement Unit) with a 3D accelerometer, 3D compass, and 3D gyroscope, and Hall effect encoders on each motor for odometry.

IV. Driving Modes and Control Architecture

A. Driving modes

The Go-CHART can drive in two modes: *Autonomous* and *Remote-Control*. In *Autonomous* mode, the robot navigates through the testbed environment without any human intervention by tracking lanes and recognizing and responding to particular objects that it detects with its sensors. In *Remote-Control* mode, a user remotely drives the Go-CHART around the testbed using the driving station shown in Fig. 6, which includes a Logitech G920 steering wheel with an accelerator and brake pedals. The monitor in the driving station displays the video feeds from all four cameras on the Go-CHART, providing a nearly 360° view of the environment in real-time with low latency (as low as 50 μs). The monitor also displays odometry readings, which are scaled down to mph. The deviations of the steering wheel and pedals from their neutral positions are transmitted remotely using socket communication protocol to the RPi3B onboard the Go-CHART, which controls the Teensy board to steer the robot in real-time (see next section). A demonstration of driving the Go-CHART in *Remote-Control* mode is shown in the supplementary video, which can also be viewed at [24].

B. Control architecture

The control architecture of the Go-CHART is illustrated in Fig. 5. The RPi3B and Teensy boards transfer information wirelessly using a two-way USB serial communication protocol. The Teensy board controls the two motor drivers, each of which drives two motors. All sensors are connected to the Teensy board, which transmits the sensor data in real-time to the RPi3B. The steering angle values obtained by the external computer, either from the lane-tracking algorithm in *Autonomous* mode or the steering wheel inputs in *Remote-Control* mode, are sent wirelessly to the RPi3B, which transmits the commands to the Teensy board to drive the motors. The RPi3B wirelessly transmits sensor data in real-time, including the camera feeds, to the external GPU for processing. The GPU runs the deep learning algorithms in Section V for lane tracking and object detection. The GPU then reports the commanded Go-CHART heading and identified objects of interest in the environment to the RPi3B for further processing. This is done by employing ROS running on the RPi3B with ROS publishers and subscribers.

V. Navigation and Object Identification

This section describes the algorithms used by the Go-CHART for lane tracking and object detection. The outer control loop of the robot performs line and obstacle detection using techniques from the OpenCV library. The 2D LiDAR and ultrasonic sensors detect obstacles, and if an obstacle is detected, the Go-CHART stops immediately and yields until the obstacle moves away. The robot also detects lines on the ground in front of it by defining a region of interest in its front-view camera image and performing color masking and blob detection. If a line is detected, the robot moves according to the class of a detected object nearby, such as a right-only sign or a stop sign.

A. Lane tracking algorithms

The Go-CHART can use two methods for lane tracking: (1) techniques from the OpenCV library [16] that apply Gaussian blur, canny edge detection, color masking, and the Hough transform to a region of interest in the front-view camera image; or (2) end-to-end behavioral cloning (NVIDIA neural network model) [12], which imitates human driving behaviors. The second method is more computationally intensive than the first, but more robust to environmental changes if it is trained with sufficiently diverse data sets.

In the first method, each camera on the Go-CHART is initially calibrated individually to obtain its intrinsic and extrinsic parameters. In addition, LDR sensor readings are obtained to perform brightness correction, making the system robust to changing lighting conditions (particularly a change in brightness, which affects OpenCV edge detection

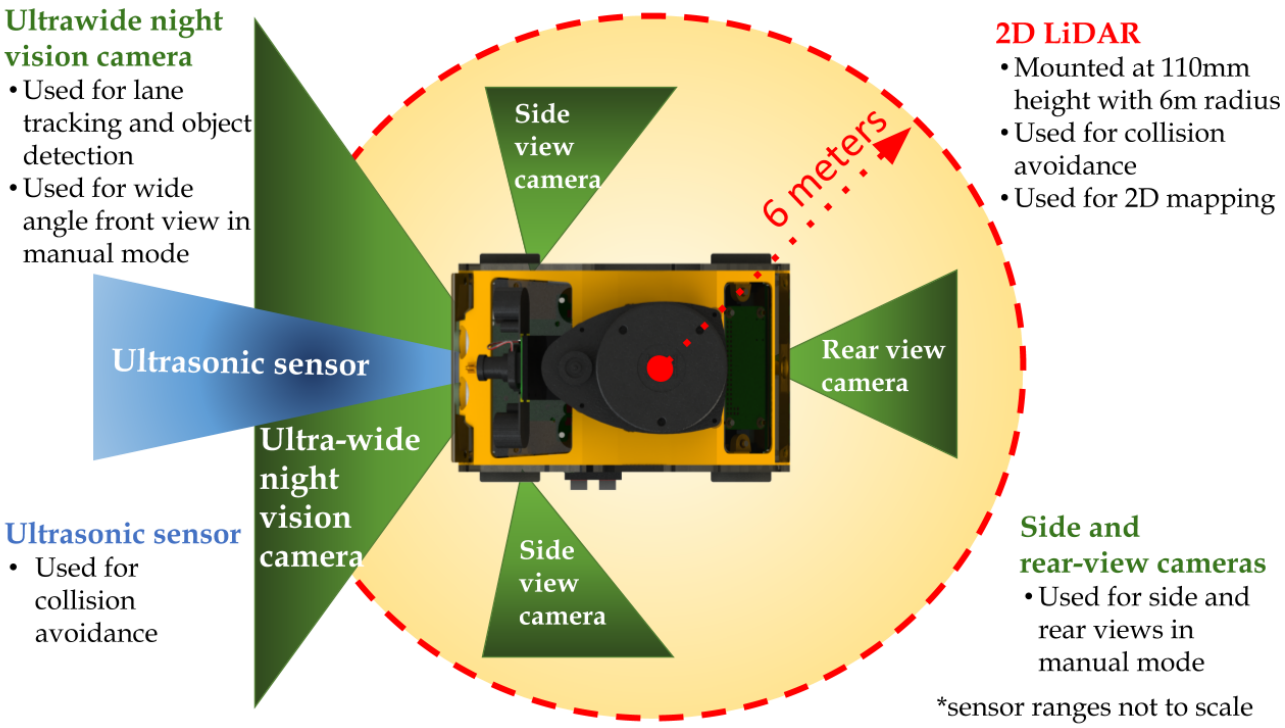


Fig. 4. Vision and range sensors on the Go-CHART.

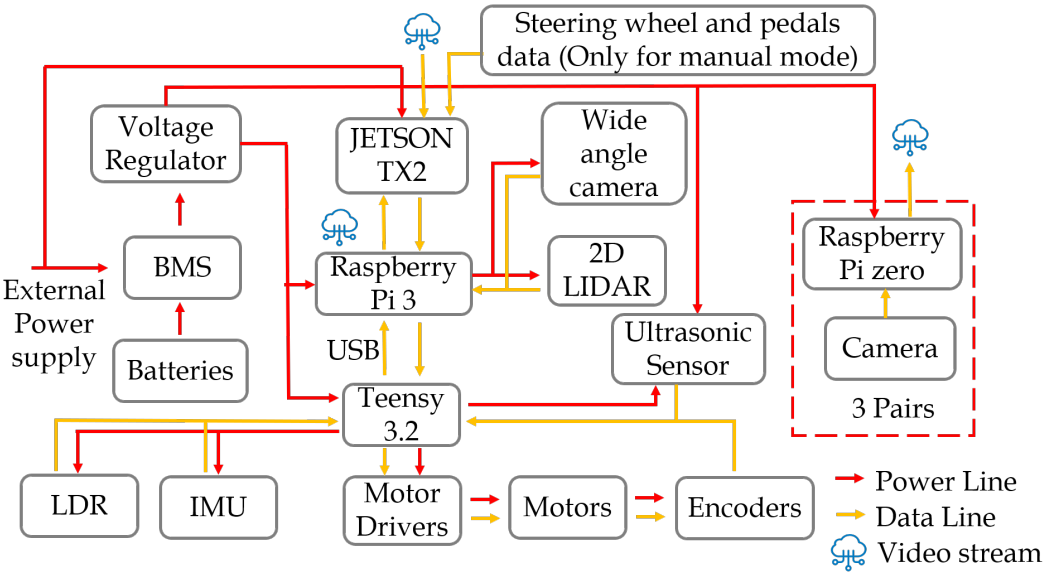


Fig. 5. Go-CHART control architecture.

techniques). The image obtained from the front-view camera is converted to gray-scale, and a Gaussian blur of kernel size 5×5 is performed on a trapezium-shaped region of interest, defined on the lower half of the image. Canny edge detection is applied to this region to detect sharp changes in adjacent pixel intensity. Following this step, the Hough transform is performed to accurately detect the boundaries of the lanes in the image, which are used to compute the steering angle required for the Go-CHART to stay within the lanes.

In the second method, a deep learning technique called end-to-end behavioral cloning (NVIDIA model) is used to perform lane tracking. The Go-CHART is remotely driven around the testbed, and the front-view camera images and corresponding steering wheel error values are recorded. The neural network is trained with these error values as inputs. Once trained, the network can replicate the same input behavior exhibited by the driver. Then, for an input camera image, the network outputs an error value that is used in the

Fig. 6. *Top:* Driving station setup for *Remote-Control* driving mode. *Bottom:* Monitor in the driving station displaying video streams from the front-view, side-view, and rear-view cameras onboard the Go-CHART.

PD steering controller (see Section III-B).

B. Object detection algorithms

To enable the Go-CHART to perform real-time detection of objects in the testbed, such as other Go-CHARTs and traffic signs and signals, we tested YOLOv3 [10], Tiny YOLO [11], ResNet [15], and Faster RCNN [13]. Ultimately, YOLOv3 [10] and Tiny YOLO [11] were selected since they detected objects with high accuracy at considerably higher frame rates than the other algorithms, given limited GPU processing power (128 CUDA cores), and performed well on low-resolution live video streams with varying latency. This enabled their use for real-time object detection, which was critical for the *Autonomous* driving mode. Low detection rates and low identification accuracy can cause the robot to miss critical environmental features, potentially causing accidents.

VI. Experiments

In this section, we demonstrate the capabilities of the Go-CHART on the small-scale driving testbed shown in Fig. 1 and evaluate the performance of its lane tracking and object identification algorithms.

A. Lane tracking

Demonstrations of the two lane-tracking methods described in Section V-A are shown in the supplementary video. When the Go-CHART uses the first lane-tracking method, the blue lines in the video indicate the lane sections that the robot tracks. This method performs lane tracking in real-time at 30 FPS without any latency. To implement the second lane-tracking method, the neural network was trained as described in Section V-A. Each camera image in the training data set was preprocessed as follows in order to meet the requirements of the NVIDIA neural network. First, a trapezium-shaped region of interest is defined in the image, and this region is converted to YUV color space. The trapezium image is stretched and resized into a rectangular image. Three regions of dimensions 352 $\times$ 288 pixels are selected from this rectangular image and then are translated by 25 pixels and rotated by 1° with respect to each other. Around 3000 images and steering error values were collected; 80% were utilized as training data, and the remaining 20% were used as validation data. Since the available data set was limited, the images were subjected to data augmentation techniques to improve learning and significantly reduce the training loss. Using the RTX 2080 Ti GPU, the neural network required around 8 min to train per epoch. The network was trained for a total of 300 epochs. The trained model can be used to predict the Go-CHART steering angle for the given input image using the external GPU. When implemented on the Jetson TX2, the network predicts the steering angle at a rate of approximately 5-7 angle values per second for the given input image size.

B. Object detection

In order to run both the YOLOv3 and Tiny YOLO object detection algorithms, the video feed from the front-view camera (frame size 320 $\times$ 240 pixels) was wirelessly streamed at 30 FPS over the wireless network to the GPU on which the trained neural network model is executed to predict the object classes of interest.

To detect U.S. traffic signs and signals and other Go-CHARTs, we needed to create a custom data set of images. Both YOLOv3 and Tiny YOLO were trained on custom data by transfer learning from the pre-trained COCO data set [14], which consists of around 200,000 labeled images and around 91 different object classes. We trained the neural network on 10 custom object classes, shown at the bottom of Fig. 7: *speed limit sign*, *stop sign*, *pedestrian crossing sign*, *traffic light sign*, *traffic signal (red, yellow, and green)*, *T intersection sign*, *turning vehicles yield sign*, and *Go-CHART robot*. The custom data set was collected from the front-view camera wireless video stream, stored in an external computer, and processed using techniques such as resizing

Fig. 7. YOLOv3 object detection results for the video feed from the Go-CHART front-view camera, with bounding boxes and class confidence levels shown. The neural network was trained on the object classes given in the bottom row. *Note:* The traffic signal includes 3 different object classes: red, yellow and green.

and annotation. It is important to record the images from the wireless stream, although some images might be blurred and low-resolution, so that the network is trained to be robust to such images. This eases the strain on the network bandwidth due to video streams from multiple Go-CHARTs, and also enables the YOLO network to produce faster results. Around 300 different images per class were obtained (a total of around 3000 images), and bounding boxes were manually marked on the regions of interest in all the images with their corresponding class labels. The training took around 30 min per epoch and required at least 20 epochs to obtain accurate predictions. However, we trained for 100 epochs in order to obtain higher confidence values for the detected classes. The number of epochs required for training decreases as the number of classes and size of the training data set for each class increase.

After the model was trained, it was executed on the Jetson TX2. The model is loaded after an initial boot-up delay of up to 90 s. Once loaded, object detection is performed considerably quickly, at around 3-5 FPS for YOLOv3 and up to 18 FPS for Tiny YOLO. The major trade-off between the two algorithms is that YOLOv3 provides accurate results but requires higher GPU usage, while the opposite is true for Tiny YOLO. Table III lists the inference speeds and mAP (Mean Average Precision) of both the YOLOv3 and Tiny YOLO algorithms, tested for four image sizes with 10 input images of each size.

TABLE III
INFERENCE SPEED AND MEAN AVERAGE PRECISION OF YOLOV3 AND TINY YOLO, TESTED ON JETSON TX2

Algorithm	Input image size in pixels	Inference speed (FPS)	mAP (Mean Average Precision)
YOLOv3	640 × 480	0.8	76.39
	480 × 360	1.9	75.37
	320 × 240	4.7	68.96
	160 × 120	9.1	22.21
Tiny YOLO	640 × 480	8.3	54.62
	480 × 360	12.6	41.72
	320 × 240	18.3	34.38
	160 × 120	19.9	16.79

Figure 7 and the supplementary video show object detection results using the YOLOv3 algorithm. If one or multiple object classes of interest are present in a particular frame from the video feed, the algorithm displays the detected object class(es), the prediction accuracy for each class, and the bounding box dimension and position of the bounding box. The prediction accuracy is set at a threshold value of 40% to ignore false positives. This threshold is relatively low because the prediction accuracy can decrease significantly for blurry, low-resolution images from the camera on a moving Go-CHART. Despite this, Figure 7 and the supplementary video show that the algorithm is generally effective at identifying objects in the testbed.

The Go-CHART decides on its next action based on the

type(s) of objects that it currently detects and the relevant traffic rules (e.g., stopping when a red light or stop sign is detected). When the Go-CHART detects multiple traffic lights at an intersection, it bases its next action on the traffic light with the largest bounding box, since that light is likely to be the closest one to the robot. The supplementary video shows scenarios in which (1) the Go-CHART stops at a red light in an intersection and resumes driving when the light turns green, and (2) the robot stops at a stop sign and waits for another Go-CHART with the right-of-way to cross in front of it before driving forward.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented the design, capabilities, and control architecture of the Go-CHART, a low-cost miniature robot that emulates a self-driving car. The Go-CHART can also be controlled remotely by a user who views the environment through the robot's four onboard cameras from a driving station. This capability will enable us to conduct experiments on interactions between human-driven and self-driving vehicles in a safe, controlled environment. We describe lane-tracking and object detection algorithms and demonstrate their implementation on a Go-CHART that autonomously navigates a miniature driving testbed, with its processing power augmented by an external GPU. We plan to expand the driving testbed to include additional traffic signs, dynamic lighting conditions, reconfigurable buildings, and media projections to customize the environment. We also plan to add artificial fog, which will enable us to develop algorithms for real-time defogging using only the robot's onboard monocular cameras. In addition, we will replace the robot's front-view camera with an RGBD camera to enable tests of real-time visual SLAM algorithms.

ACKNOWLEDGMENTS

The authors thank Sreenithy Chandran (ASU) for helping to implement the deep learning techniques and Sangeet Ulhas (ASU), Rakshith Subramanyam (ASU), Karthik Ganesan (ASU), and Sritanay Vedartham (BASIS Scottsdale) for helping build the testbed.

REFERENCES

[1] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, et al. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1497-1504. IEEE, 2017.

[2] Sean Wilson, Ruben Gameros, Michael Sheely, Matthew Lin, Kathryn Dover, Robert Gevorkyan, Matt Haberland, Andrea Bertozzi, and Spring Berman. Pheeno, a versatile swarm robotic research and education platform. *IEEE Robotics and Automation Letters* vol. 1, no. 2, pp. 884-891, July 2016.

[3] Nicholas Hyldmar, Yijun He, and Amanda Prorok. A fleet of miniature cars for experiments in cooperative driving. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3238-3244. IEEE, 2019.

[4] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods* 36(5):628-647, 2015.

[5] Sertac Karaman, Ariel Anders, Michael Boulet, Jane Connor, Kenneth Gregson, Winter Guerra, Owen Guldner, et al. Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at MIT. In *2017 IEEE Integrated STEM Education Conference (ISEC)*, pp. 195-203. IEEE, 2017.

[6] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The Robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1699-1706. IEEE, 2017.

[7] Joseph Betthauser, Daniel Benavides, Jeff Schornick, Neal O'Hara, Jimit Patel, Jeremy Cole, and Edgar Lobaton. WolfBot: A distributed mobile sensing platform for research and education. In *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education*, pp. 1-8. IEEE, 2014.

[8] Adam Stager, Luke Bhan, Andreas Malikopoulos, and Liuhui Zhao. A scaled smart city for experimental validation of connected and automated vehicles. *arXiv preprint* arXiv:1710.11408 (2017).

[9] Rakshith Subramanyam (2018). CHARTOPOLIS: A Self Driving Car Test Bed. *Master's Thesis in Electrical Engineering, Arizona State University*.

[10] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint* arXiv:1804.02767 (2018).

[11] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers. In *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2503-2510. IEEE, 2018.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel et al. End to end learning for self-driving cars. *arXiv preprint* arXiv:1604.07316 (2016).

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pp. 91-99. 2015.

[14] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO captions: Data collection and evaluation server. *arXiv preprint* arXiv:1504.00325 (2015).

[15] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in Resnet: Generalizing residual architectures. *arXiv preprint* arXiv:1603.08029 (2016).

[16] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 120–126, 2000.

[17] Google Maps: Miniatur Wunderland. Accessed: 2020-07-31. [Online] Available: https://www.google.com/maps/about/behind-the-scenes/streetview/treks/miniatur-wunderland/

[18] University of Michigan Mcity. Accessed: 2020-07-31. [Online] Available: https://mcity.umich.edu/

[19] Karen Hao. "The latest fake town built for self-driving cars has opened in South Korea." *Quartz*, Nov. 6, 2017. Accessed: 2020-07-31. [Online] Available: https://qz.com/1121372/south-korea-opens-k-city-the-latest-fake-town-built-for-self-driving-cars/

[20] Millbrook: Connected and Autonomous Vehicle Testing. Accessed: 2020-07-31. [Online] Available: https://www.millbrook.us/services/connected-and-autonomous-vehicle-testing/

[21] Srinivasa, S.S., Lancaster, P., Michalove, J., Schmittle, M., Rockett, C.S.M., Smith, J.R., Choudhury, S., Mavrogiannis, C. and Sadeghi, F. MuSHR: A low-cost, open-source robotic racecar for education and research. *arXiv preprint* arXiv:1908.08031 (2019).

[22] Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B. and Calleja, E. DeepRacer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning. *arXiv preprint* arXiv:1911.01562 (2019).

[23] W. Roscoe. Donkey Car: An opensource DIY self driving platform for small scale cars. Accessed: 2020-07-31. [Online] Available: https://www.donkeycar.com

[24] NVIDIA Autonomous Machines. Accessed: 2020-07-31. [Online] Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetbot-ai-robot-kit/

[25] "Go-CHART: A miniature remotely accessible self-driving car robot," Autonomous Collective Systems Laboratory Youtube channel, https://www.youtube.com/watch?v=pAa61VpF6oQ