Proceedings of the 2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Auckland, New Zealand, July 9-12, 2018

A Low-Cost Hardware-in-the-Loop Agent-Based Simulation Testbed for Autonomous Vehicles

Ray Barker[1], Aaron Hurst[1], Ridge Shrubsall[1], Ghulam Mubashar Hassan[2] and Tim French[2]

***Abstract*— We have developed a flexible and low-cost hardware testbed for autonomous vehicle research and education. The testbed provides the ability to autonomously control multiple small vehicles within a 1200x900 mm environment and is entirely open source with regard to both hardware and software, making it easily reproducible. An agent-based approach is used for vehicle control, meaning that different agent models can be applied to each vehicle as desired for simulation. GPS-like position information is provided to the controller in real-time using a global computer vision system.**

Using our figure-of-eight test environment, we have demonstrated that our system can support six cars driving smoothly around the track whilst avoiding collisions with each other.

I. INTRODUCTION

Autonomous and semi-autonomous vehicle technologies are expected to establish a major position in the automobile market within the next few decades [1]–[3]. This will result in significant changes to driving patterns, vehicle usage and government policy. Hence, significant research and public engagement will be necessary to ensure a smooth and successful transition to this new technology.

Traditionally, modelling and analysis of transport systems has been done using software simulations [4]–[6]. Widely used programs include MATsim [7], Aimsun [8] and VISSIM [9]. Aimsun, for example, has been used to model complex intersection layouts, delivering intersection design improvements based on its simulation results [8].

Additionally, traffic simulations are also used in education. In [10], it was demonstrated that simulation enabled experiential learning in the field of transportation engineering. The authors also provided qualitative evidence that students were engaged in the educational content, particularly the simulation elements. In [11], the significant barrier to entry posed by the complexity of simulation software commonly available was recognised. To resolve this, an intuitive web-based traffic simulation tool was developed to allow students to understand and implement different control strategies and layouts.

However, adequately detailed software simulations for modelling and analysis tasks have inherent complexity, both in usage and data interpretation. Even with such complexity there is the danger of idealising salient real-world phenomena. Further, some programs do not allow for flexible simulation of customised agent-based vehicle behaviour algorithms.

Numerous hardware-based testbeds have also been used to good effect and address some of the challenges of software simulations. The advantages of such testbeds are their intuitive tangible output and native inclusion of real-world phenomena.

One of the more well-developed hardware testbeds that has been through multiple iterations was created by researchers at the University of California, Los Angeles [12]–[15]. Early iterations of this testbed consisted of a fixed driving area and multiple vehicles, cameras and computers [12]. While this testbed did deliver a small overall footprint (1.5x2.0 m) and small vehicle size (64x34 mm) – increasing its usability and making it somewhat portable – the need for multiple desktop computers increased cost, complexity and bulkiness.

Later iterations have increased the complexity of the cars by adding distributed computing and multiple sensors and reduced or eliminated reliance on external computing and sensors [15]. This has improved portability and flexibility, but at the cost of complexity and price.

Other researchers have used much larger robotic vehicles for similar projects, which require room-sized spaces [16], [17]. One project used Zen-Wheels micro-cars (58x30 mm) in a 1.2x2.4 m testbed [18]. This project was not completed, however its aims included portability and providing an easily reconfigurable testbed environment.

Some work in robotics research has also involved constructing testbeds for mobile robot applications. Typically these are more specialised in nature. For example: sensor integration within a rugged mobile robot for use in an outdoor test environment for planetary exploration [19], novel wheel design and drive control for autonomous robots in extreme environments [20] and testing of a control system for aggressive manoeuvres by miniature autonomous vehicles [21].

Overall, tangible representations such as physical testbeds can both facilitate research in the field of traffic modelling and autonomous and semi-autonomous vehicles. They are useful tools for stimulating public engagement with research and autonomous vehicles.

The objective of this project was to develop a tool for this purpose. That is, to develop a tabletop autonomous and semi-autonomous vehicle simulation system which allows for the testing, demonstration and education of autonomous vehicle algorithms and behaviours under real-world conditions. Design requirements were to develop a reproducible, low

[1]Final year students at The University of Western Australia, 35 Stirling Highway, Crawley, WA 6009, Australia

[2]Tim French (tim.french@uwa.edu.au) and Ghulam Mubashar Hassan (ghulam.hassan@uwa.edu.au) are with the Department of Computer Science and Software Engineering, The University of Western Australia, 35 Stirling Highway, Crawley, WA 6009, Australia

978-1-5386-1854-7/18/$31.00 ©2018 IEEE

cost, easily transportable and configurable system which can simulate a range of environments and up to six vehicles in real time.

The remainder of this paper consists of a description of the architecture of the testbed system that has been developed (Sec. II), a description of the design verification experiments undertaken (Sec. III) and conclusions with a discussion of current development work (Sec. IV).

II. TESTBED ARCHITECTURE

A block diagram illustrating the sub-systems and interfaces in our testbed is shown in Fig. 1. Beginning with the camera, the control loop is as follows. First, images of the driving surface are captured and sent to the computer vision system. This software system, written in C++ and running on a Raspberry Pi 3, processes each image and determines the kinematic state of each vehicle (position, velocity and orientation).

This vehicle state information is passed to the controller program – written in Python and running on the same Raspberry Pi computer – which in turn passes a description of the environment to agent models for the vehicles. The agents determine appropriate actions for each vehicle, given their state, environment and objective. Actions are then returned to the controller program.

Finally, the controller translates these actions into driving commands and transmits these to each vehicle via Bluetooth. Simultaneously, the controller also sends the driving environment as an image to the projector, which displays this over the driving surface for visual representation of the environment. A photograph of the testbed as constructed is shown in Fig. 2.

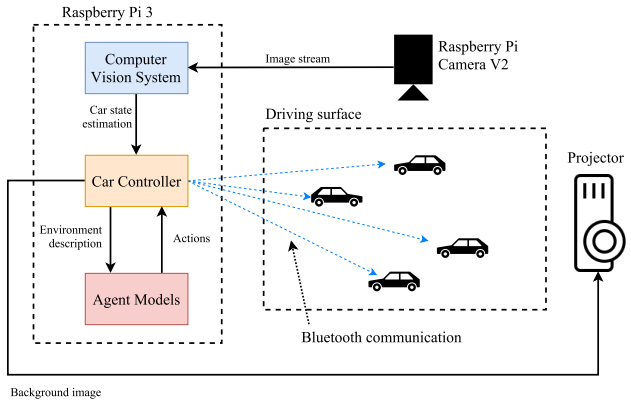

Fig. 1. Testbed system architecture for the testbed showing key sub-systems and interfaces.

A. Hardware

The hardware for the system consists of the following components:

1) Zen-Wheels Bluetooth Micro Cars,
2) Raspberry Pi 3,
3) Raspberry Pi Camera V2,
4) Short-throw projector, and

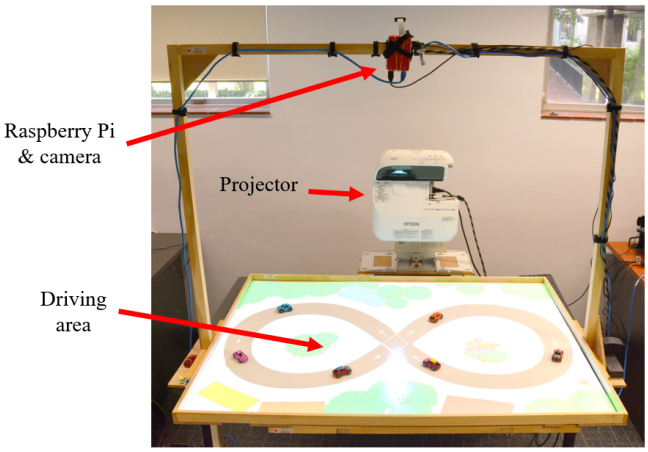

Fig. 2. Testbed as constructed.

5) Custom-built frame.

Zen-Wheels cars measure approximately 60x30 mm. They are sold as remote control cars for smart phones. The communication protocol of the cars has been adapted so that they can be controlled using a Raspberry Pi instead of a smart phone.

The Raspberry Pi 3 runs all of the software for this system. A Raspberry Pi Camera is directly connected to the Pi for the computer vision system. The benefits of using a Raspberry Pi are its small size, low cost and ubiquity.

The projector is used to display an environment on the driving surface that can be easily changed between simulations. This environment may consists of roads, trees and buildings and is necessary to make the system look visually appealing. Using a short-throw projector reduces the size required for the system, compared to using a traditional projector.

The frame connects the driving surface, projector and camera/computer mount. By interconnecting these components, complex alignment issues of the camera and projector are eliminated, thereby reducing the set-up time of the system. The design of the frame is shown in Fig. 3. Detailed design drawings of the frame are available on the project GitHub [22].

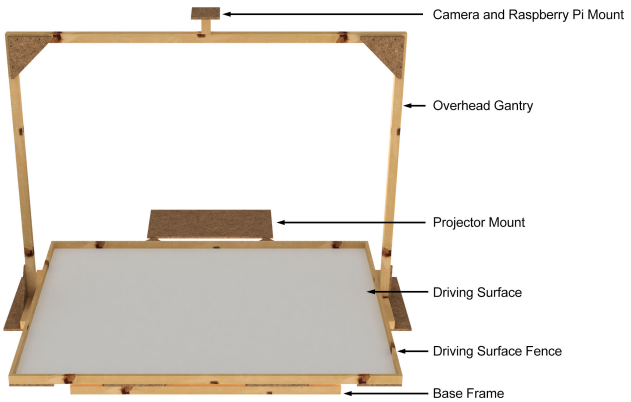

Fig. 3. Render of the custom frame design

The size of the driving surface is 1.2x0.9 m. This size has an aspect ratio of 4:3 which maximises utilisation of the camera's field of view. The driving surface is white melamine, which provides a good surface finish for projection, driving, and computer vision. A fence around the driving surface protects cars from damage by preventing them from falling off the edge.

B. Computer Vision System

Tracking of the vehicles is accomplished by analysing images from a single Raspberry Pi Camera. Images are captured at 640x480 pixel resolution. With the custom frame the camera's field of view extends slightly beyond the driving surface. Hence, each pixel represents approximately 1.95x1.95 mm.

The software for tracking the vehicles is written in C++ and relies heavily on the OpenCV library [23]. The algorithm used for tracking is based on histogram comparison [24] combined with background subtraction and operates as follows:

1) Obtain new image.
2) Compute *difference image* as absolute difference between current image and a reference *background image*.
3) Generate a binary mask that highlights all cars present by converting difference image to grayscale and applying a threshold operation to select only those pixels with high intensity (i.e. large difference).
4) Apply dilation once using a 3x3 kernel.
5) Calculate colour (i.e. hue) histograms over each connected region that matches the known size of the vehicles.
6) Compare these *observed histograms* to *prototype histograms* for each known vehicle using the χ^2 distance [25] as a comparison metric.
7) Identify vehicles by associating a detected region (or none) with each vehicle in use based on χ^2 distance measurements.
8) Calculate vehicle state (found/not found, position, velocity and orientation).
9) Send vehicle state information to controller program.

The *background image* must be obtained before tracking commences and should not include any of the vehicles. Prototype histograms, like the observed histograms, are histograms of the colour (hue) values of the individual vehicles. They are generated prior to use by using a separate program. Dilation is required to 'fill in' gaps within semi-connected regions that should be joined to represent a single car.

The identification step (No. 7) is accomplished by interpreting the lowest χ^2 distance (most similar pairing of observed and prototype histograms) below a minimum threshold for matching as a positive detection. This is repeated for each vehicle (without replacement) until all regions are either associated with a vehicle or deemed not to represent a vehicle (no match below χ^2 distance threshold).

C. Controller

The controller's primary functions are to provide an environment model, perform closed loop control of the cars and interface the computer vision data to the agents. The controller is also responsible for communicating with the hardware cars.

The software has been written in Python 3, and is available at the project GitHub [22]. Performance of the software permits six cars to be simulated reliably, with a total system delay of approximately 150 ms.

A description of the major controller components follows.

1) Environment Model: The model of the environment captures the position of all vehicles, roads and intersections. The road and intersection layout is defined in advance using a configuration file, whilst the vehicle positions are continuously updated from the computer vision data. The environment model is distributed to the agents as their primary percept for making driving decisions.

A visual representation of the environment is projected onto the driving surface. The visual representation primarily serves to make the system visually engaging.

The environment is customisable, meaning that our simulation platform could simulate a range of environments.

2) Closed Loop Control: Closed loop control of the cars is essential as the cars do not drive perfectly as instructed. A control loop which controls the orientation of the car using computer vision has been developed, and is shown in Fig. 4. Essentially, the observed orientation is used to control the steering to make the cars drive in a desired orientation.

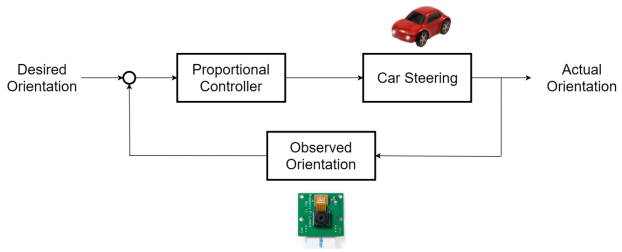

Fig. 4. Diagram of closed loop control system for vehicle steering

The equation for a generic proportional controller is

$$P_{out} = k_p * e_p + p_0$$

where P_{out} is the output of the controller, k_p is the proportional gain, e_p is the instantaneous error and p_0 is the output of the controller with no error.

To steer the cars, a proportional controller on car orientation is used. Driving the cars along arbitrary paths is achieved by targeting points along the track in series.

The current orientation of the car is observed from the computer vision whilst the required orientation is calculated as the orientation of the vector from the observed position to the target position. The difference between these orientations is the error angle as shown in Fig. 5 and denoted θ_{error}. This term is the instantaneous error e_p for the controller.

The proportional controller was tuned experimentally, with a proportional gain of $k_p = 0.3$ found to be most effective. Additionally, $p_0 = 0$ because no steering is required if the car has the correct orientation. The final equation for the orientation proportional controller is then

$$Steering = 0.3 * \theta_{error}$$

Proportional control was found to be sufficient for this purpose. It was not necessary to use a PI or PID controller; however, it is likely that this would have slightly reduced path tracking average error and overshoot.

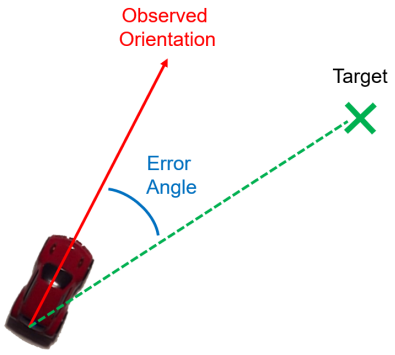

Fig. 5. Determination of error angle for orientation P controller

Once the car is within 5 cm of the target point, the car will then target the next point along the track. This is so that the cars do not have to reach the exact coordinates of the target point, leading to endless circling around the target point.

D. Agent Models

Bespoke agent models can be developed and simulated using the system. To facilitate this, an interface between the car controller and agent model has been specified.

On each loop, the *update_percepts()* function is called by the core thread which updates the observed state of the agent. The *decide_actions()* function is then called. In this function, the logic of the agent is programmed to use the available percepts and internal state to determine a list of actions which it wishes to perform. The list of actions is returned and then sent to the cars via Bluetooth by the core thread at the next available opportunity.

Our project has implemented one example agent from which other agents can be developed. The example agent has the simple objective of travelling around the track whilst avoiding collisions with the other cars. To do this, it uses its observed position to determine its desired position which is the next point on the track. A list of instructions to reach this position are returned and queued for transmission to the car. Collision avoidance is performed by issuing a stop command when the agent sees another car within the collision avoidance radius.

III. Experiments

A. Design Evaluation

To be an effective traffic simulation tool suitable for use in classroom or research situations, this testbed was designed to be reproducible, low-cost, transportable, configurable and real-time.

To ensure the system is reproducible, all system code is available on the computer vision and controller Github repositories, [26] and [22] respectively. Detailed drawings of the custom frame are also available on the controller Github repository.

The custom frame described in Sec. II-A combined with the choice of a low-cost and light-weight central controller (Raspberry Pi) addresses the need for transportability and low cost. The following cost breakdown in Table I demonstrates that the overall system cost is relatively low, especially if the optional projector is omitted.

TABLE I
Cost breakdown of testbed

Item	Approx. cost (AUD)
Zen-Wheels Cars (x6)	450
Raspberry Pi	50
Raspberry Pi Camera	10
Custom frame	100
Projector	1,400
Total	**2,010**

Configurability of the testbed is provided through the agent-based approach to car control described previously as well as the ability to simulate a custom environment using the projector.

To provide a real-time simulation system, the two key performance measures are:

1) Car detection accuracy, and
2) System delay.

Car detection accuracy is important since the observed positions over time are used to calculate the error term for the proportional controller. Additionally, the system delay must be minimised so that the observed position has not changed significantly by the time that the car performs the corrective steering action. Overall, this leads to visually realistic and smooth driving behaviour.

B. Car Detection Accuracy

The accuracy of the computer vision system is dependent on external lighting conditions and proper calibration. However, once prototype histograms and difference image thresholds are properly calibrated, the system can achieve detection accuracy above 95% under reasonable indoor lighting conditions.

For example, the computer vision system achieved an average accuracy of 97% over a test set of 6,000 image frames with six cars using the projected environment shown in Fig. 6. If no projected environment is used, this figure increases slightly.

There is some variation in detection performance between cars of different colours. However, this was limited to only 3 percentage points between the best and worst performing car in the above test case.

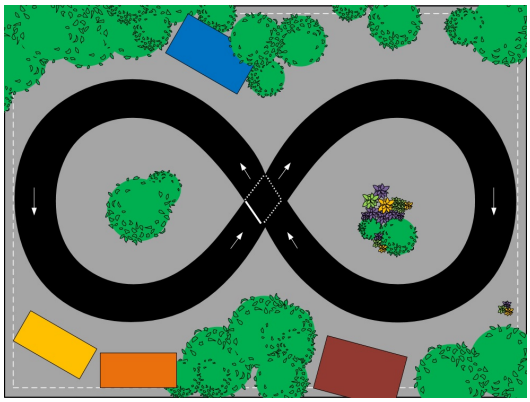

Fig. 6. Figure-of-eight test scenario

The precision of car detection as also measured. Position precision was measured by the standard deviation in position results for stationary cars, while speed precision was measured by their mean speed.

Overall, the system was found to provide position estimates to within ± 0.4 mm and speed estimates are within ± 3.9 mm/s. This was averaged across three tests of 1,000 image frames each with six cars.

To put this in context, the cars are approximately 60 mm in length, so this represents a mean error in position of less than 1% relative to car length. In terms of speed, typical car speeds are in the range 100-300 mm/s. Hence, this represents a mean error of 1-4%.

C. System Delay

The time performance of the computer vision system was analysed in detail due to computer vision often being a resource intensive and time consuming computing task. To this end, three trials of 1,000 frames each with between one and six cars were carried out for the algorithm described in Sec. II-B.

As shown in Fig. 7, the algorithm achieved frame rates of approximately 15 fps for each case, only declining slightly as more cars are added. This level of performance was more than satisfactory for use in this testbed.

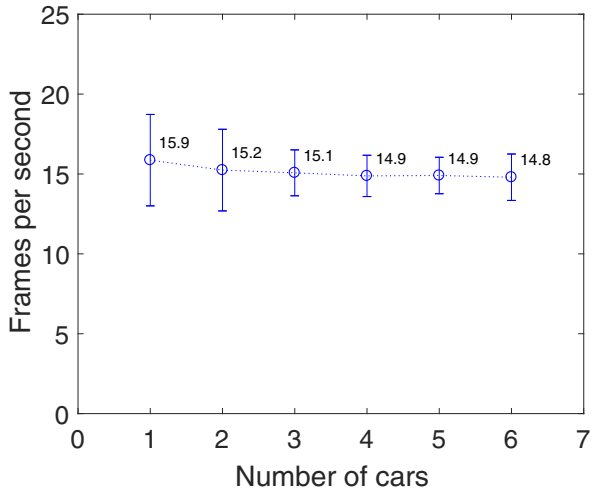

Fig. 7. Speed of the computer vision system is only slightly affected by the number of cars, remaining close to 15 fps for one to six cars. Average speeds measured over three trials 1,000 images with error bars representing one standard deviation either side of the mean.

IV. CONCLUSIONS

We have developed a flexible and versatile hardware-in-the-loop simulation system for autonomous vehicle research and education. By using physical cars and computer vision-based GPS-like position data, real-world phenomena are inherently incorporated into the simulation. Hence, car control algorithms and agents can be researched and tested in a quasi-real world environment.

Simulation of six cars in custom environments with custom agents is possible with our simulation system. The design is transportable, reproducible and relatively low-cost.

Development work currently in progress includes improving the ability of the computer vision system to handle varying lighting conditions and temporary occlusions, adding interactivity, and the ability to dynamically build an environment from roads drawn on the driving surface with a whiteboard marker, so as to provide a lower cost design variant.

Applications of the developed system include simulating real-world problems, such as logistics scheduling scenarios, autonomous mining equipment operating at a mine site or analysis of the interaction between human and autonomous drivers on public roads.

ACKNOWLEDGMENT

The authors would like to acknowledge Elizabeth Loreck, Mason Pearce, Amanda Quy and Theodore Vickery for their prior work developing the initial Python control software for the Zen-Wheels cars.

REFERENCES

[1] S. A. Bagloee, M. Tavana, M. Asadi, and T. Oliver, "Autonomous vehicles: challenges, opportunities, and future implications for transportation policies," *Journal of Modern Transportation*, vol. 24, no. 4, pp. 284–303, 2016.

[2] KPMG and Center for Automotive Research, "Self-driving cars: The next revolution," 2012. [Online]. Available: https://faculty.washington.edu/jbs/itrans/self_driving_cars%5B1%5D.pdf

[3] J. Manyika, M. Chui, J. Bughin, R. Dobbs, P. Bisson, and A. Marrs, "Disruptive technologies: Advances that will transform life, business, and the global economy," 2013.

[4] M. Bahram, Z. Ghandeharioun, P. Zahn, M. Baur, W. Huber, and F. Busch, "Microscopic traffic simulation based evaluation of highly automated driving on highways," in *Proceedings of the 17th International Conference on Intelligent Transportation Systems*, 2014, pp. 1752–1757.

[5] M. Pursula, "Simulation of Traffic Systems - An Overview," *Journal of Geographic Information and Decision Analysis*, vol. 3, no. 1, pp. 1–8, 2016. [Online]. Available: http://publish.uwo.ca/~jmalczew/gida_5/Pursula/Pursula.html

[6] E. Thonhofer, T. Palau, A. Kuhn, S. Jakubek, and M. Kozek, "Macroscopic traffic model for large scale urban traffic network design," *Simulation Modelling Practice and Theory*, vol. 80, pp. 32–49, 2018.

[7] M. Balmer, M. Rieser, K. Meister, D. Charypar, N. Lefebvre, and K. Nagel, "MATSim-T: Architecture and simulation times," in *Multi-Agent Systems for Traffic and Transportation Engineering*. IGI Global, 2009.

[8] K. Zhang, A. Excell, *et al.*, "Modelling a complex give-way situation-aimsun vs linsig," *Road & Transport Research: A Journal of Australian and New Zealand Research and Practice*, vol. 22, no. 2, p. 16, 2013.

[9] X. Y. Lu, J. Lee, D. Chen, J. Bared, D. Dailey, and S. E. Shladover, "Freeway micro-simulation calibration: Case study using aimsun and VISSIM with detailed field data," in *Proceedings of the 93rd Annual Meeting of the Transportation Research Board, Washington, DC*, 2014.

[10] F. C. Fang and D. Pines, "Enhancing transportation engineering education with computer simulation," *International Journal of Engineering Education*, vol. 23, no. 4, pp. 808–815, 2007.

[11] C.-F. Liao, T. Morris, and M. Donath, "Development of internet-based traffic simulation framework for transportation education and training," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1956, pp. 184–192, 2006.

[12] C. H. Hsieh, Y. L. Chuang, Y. Huang, K. K. Leung, A. L. Bertozzi, and E. Frazzoli, "An Economical Micro-Car Testbed for Validation of Cooperative Control Strategies," in *Proceedings of the American Control Conference*, 2006, pp. 1446–1451.

[13] K. K. Leung, C. H. Hsieh, Y. R. Huang, A. Joshi, V. Voroninski, and A. L. Bertozzi, "A second generation micro-vehicle testbed for cooperative control and sensing strategies," in *Proceedings of the American Control Conference*, 2007, pp. 1900–1907.

[14] M. Gonzalez, X. Huang, D. S. H. Martinez, C. H. Hsieh, Y. R. Huang, B. Irvine, M. B. Short, and A. L. Bertozzi, "A Third Generation Micro-vehicle Testbed for Cooperative Control and Sensing Strategies," in *Proceedings of the International Conference on Informatics in Control, Automation and Robotics*, 2011, pp. 14–20.

[15] D. H. Martinez, M. Gonzalez, X. Huang, B. Irvine, C. H. Hsieh, Y. R. Huang, M. B. Short, and A. L. Bertozzi, "An Economical Testbed for Cooperative Control and Sensing Strategies of Robotic Micro-Vehicles," in *Proceedings of the International Conference on Informatics in Control, Automation and Robotics*, 2013, pp. 65–75.

[16] R. A. Cortez, J. M. Luna, R. Fierro, and J. Wood, "A multi-vehicle testbed for multi-modal, decentralized sensing of the environment," in *Proceedings of the International Conference on Robotics and Automation*, 2010, pp. 1088–1089.

[17] C. Yan and Q. Zhan, "Real-time multiple mobile robots visual detection system," *Sensor Review*, vol. 31, no. 3, pp. 228–238, 2011.

[18] J. D. Suter, "Miniature Vehicle Testbed for Intelligent Transportation Systems," Honours Thesis, Ohio State University, 2016.

[19] A. Gauthier, M. Pelletier, A. Salerno, P. Allard, S. Gemme, T. Lamarche, and E. Dupuis, "Csa redesign of the mobile robotics test-bed (mrt)," in *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*. IEEE, 2010, pp. 489–496.

[20] K. Tanaka, Y. Okamoto, H. Ishii, D. Kuroiwa, J. Mitsuzuka, H. Yokoyama, S. Inoue, Q. Shi, S. Okabayashi, Y. Sugahara, *et al.*, "Hardware and control design considerations for a monitoring system of autonomous mobile robots in extreme environment," in *Advanced Intelligent Mechatronics (AIM), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1412–1417.

[21] A. Arab, K. Yu, J. Yi, and Y. Liu, "Motion control of autonomous aggressive vehicle maneuvers," in *Advanced Intelligent Mechatronics (AIM), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1663–1668.

[22] Ray Barker, "GitHub - Cars Controller Project," 2017. [Online]. Available: https://github.com/raybarker95/cars-controller

[23] OpenCV Team, "About," 2017. [Online]. Available: http://opencv.org/about.html

[24] M. J. Swain and D. H. Ballard, "Color Indexing," *International Journal of Computer Vision*, vol. 7, no. 1, pp. 11–31, 1991.

[25] O. Pele and M. Werman, "The Quadratic-Chi Histogram Distance Family," in *European Conference on Computer Vision*, 2010, pp. 749–762.

[26] Aaron Hurst, "GitHub - Computer vision for a Traffic and Autonomous Vehicle Testbed," 2017. [Online]. Available: https://github.com/aaron-hurst/testbed-tracking