Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research

Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, Andrea Censi

Abstract—Duckietown is an open, inexpensive and flexible platform for autonomy education and research. The platform comprises small autonomous vehicles (“Duckiebots”) built from off-the-shelf components, and cities (“Duckietowns”) complete with roads, signage, traffic lights, obstacles, and citizens (duckies) in need of transportation. The Duckietown platform offers a wide range of functionalities at a low cost. Duckiebots sense the world with only one monocular camera and perform all processing onboard with a Raspberry Pi 2, yet are able to: follow lanes while avoiding obstacles, pedestrians (duckies) and other Duckiebots, localize within a global map, navigate a city, and coordinate with other Duckiebots to avoid collisions. Duckietown is a useful tool since educators and researchers can save money and time by not having to develop all of the necessary supporting infrastructure and capabilities. All materials are available as open source, and the hope is that others in the community will adopt the platform for education and research.

I. INTRODUCTION

Self-driving vehicles are poised to become one of the most pervasive and impactful applications of autonomy. However, difficult challenges still remain before their widespread deployment, many of which concern the system as a whole, rather than single components in isolation. Examples include the co-design of hardware components and algorithms, the coupled interactions between perception and control, the optimal allocation of finite computational resources to concurrent processes, and safe multi-agent behaviors.

A modern curriculum in autonomy should train students in the individual components and the system-level interactions alike. This poses several challenges. First, building a full-scale vehicle, let alone a fleet of vehicles, is very costly and also imposes significant logistical and safety-related problems. Second, the time required to develop all of the components and infrastructure is significant, and much of it is spent on tasks not directly related to the desired subject matter.

To address these issues, we propose Duckietown (Fig. 1), an open-source platform for autonomy education and research. It includes autonomous vehicles called “Duckiebots.” The minimal configuration uses only a Raspberry Pi 2 for all computation and a single monocular camera for sensing, yet Duckiebots are capable of fairly complex single-robot and multi-robot behaviors. Duckiebots live in “Duckietowns,” colorful miniature environments that are assembled from modular tiles. Duckietowns and Duckiebots are easily reproducible and inexpensive, costing approximately $150 per vehicle and $2/m² for the environment.

Duckietowns are carefully designed to allow a sliding scale of difficulty in perception, inference and control tasks that makes the platform usable in a wide range of applications, from undergraduate-level education to research-level problems. For example, one solitary Duckiebot can successfully traverse the environment using only line detection and reactive control, while successful point-to-point navigation requires recognizing street signs. In turn, sign detections can be “simulated” either by using fiducials (AprilTags [1]) affixed to each sign, or it can be implemented using “real” object detection. Realizing more complex behaviors, such as vision-based decentralized multi-robot coordination, poses research-level challenges, especially considering resource constraints.

A major advantage of the Duckietown platform is derived from the complex software architecture provided, which contains components such as sensor calibration, configuration, low-level perception, object recognition, nonlinear relative estimation, global localization, high-level planning and decentralized coordination. Our goal was to provide a complete “textbook” architecture that is comparable in complexity with full-scale implementations (e.g. [2]), while still being understandable by beginners.

The main intended use of Duckietown is as support for

The authors are affiliated with the Massachusetts Institute of Technology. A. Censi and E. Frazzoli are also affiliated with ETH Zurich. A. Censi, D. Yershov, S. Liu, E. Frazzoli are also affiliated with no anonymity. They report no conflict of interest. Corresponding author email: lpaull@csail.mit.edu. All materials related to the project are released under open-source license and available at duckietown.mit.edu.
TABLE I
MINIMAL AUTONOMY CONFIGURATION COMPONENTS

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Item</th>
<th>OTS cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Raspberry Pi 2 B + 8GB SD card</td>
<td>$40</td>
</tr>
<tr>
<td>Actuation</td>
<td>Chassis + 2DC motors</td>
<td>$15</td>
</tr>
<tr>
<td>Motor Controller</td>
<td>Adafruit DC motor hat</td>
<td>$20</td>
</tr>
<tr>
<td>Sensing</td>
<td>Camera with fish-eye lens</td>
<td>$25</td>
</tr>
<tr>
<td>Power</td>
<td>38.4Wh Battery</td>
<td>$15</td>
</tr>
<tr>
<td>Communication</td>
<td>Wireless dongle</td>
<td>$10</td>
</tr>
<tr>
<td>Misc.</td>
<td>Cables, Screws &amp; Nuts</td>
<td>$10</td>
</tr>
</tbody>
</table>

Prices rounded to the closest $5. Exact items and links to vendors are available at the website http://duckietown.mit.edu.

We present three configurations:

1) Minimal Autonomy Configuration:

Duckiebots are autonomous vehicles designed with the objectives of affordability, modularity, and ease of construction. We present three configurations:

- **The minimal autonomy** configuration is sufficient to support all single-robot behaviors implemented.
- **The extended** configuration adds some “luxury” features that are convenient for development, such as a joystick and an on-board wireless access point.
- **The fleet** configuration includes light emitting diodes (LEDs) as a means of inter-Duckiebot communication and enables the multi-robot coordination behaviors.

   1) **Minimal Autonomy Configuration:** The components of a Duckiebot in the minimal configuration are summarized in Table I. These are all off-the-shelf components that are easily replaceable. Each robot requires about 15 minutes of soldering work and 0.5-1.5 hours of assembly, depending on skill.

   - **Computation:** All computation is performed on a Raspberry Pi 2 (RasPi), which provides four 900MHz ARM cores and 1 GB RAM.
   - **Actuation:** The chassis of the Duckiebot can be any of the many available kits that can be found online that use two DC motors in a differential drive configuration. Since we do not use odometers, the chassis is the most fungible part of the design. The motors are controlled through an Adafruit DC Motor Hat that attaches as a “shield” on the RasPi.
   - **Sensing:** The only sensor used is a monocular camera with a fish-eye lens. The camera is connected to the RasPi through a dedicated parallel connection.
   - **Communication:** Access (for setup, debug and optional manual control through keyboard) is provided through a WiFi dongle or Ethernet port on the RasPi.
   - **Coordination:** This version of the Duckiebot is equipped with five variable color LEDs, and an Adafruit pulse-width modulation (PWM) hat to drive them. The LEDs are employed for communicating with other vehicles by signaling the Duckiebot’s status and intent. The introduction of these LEDs increases the platform cost by $75.

2) Extended Configuration:

For teaching and research setups with large team deployments, the best solution to network saturation is to use an access point onboard each Duckiebot. These mobile hotspots create a dedicated 5GHz network for each robot and connect directly to the RasPi through Ethernet. Additionally, a wireless joystick with USB dongle enables more convenient manual control. Finally, a 32GB USB drive can be employed to store larger amounts of data logs. This configuration adds an additional $75.

3) Fleet Configuration:

This version of the Duckiebot is equipped with five variable color LEDs, and an Adafruit pulse-width modulation (PWM) hat to drive them. The LEDs are employed for communicating with other vehicles by signaling the Duckiebot’s status and intent. The introduction of these LEDs increases the platform cost by $30 over the base model.

We compare the base model Duckiebot with some other popular robots used for education and outreach in Table II.
Of particularly note is how few low-priced available robots use vision as the primary sensor. The choice of a camera, as opposed to a proximity or infrared (IR) sensor makes our system a much more realistic representation of a full-sized platform.

III. LANE FOLLOWING

The most basic behavior of the Duckiebot is lane following. This behavior is implemented using a realistic computer vision pipeline (Fig. 2) that contains these steps:

- Illumination variability compensation.
- Detection of road markings.
- Re-projection from image space to world frame, based on extrinsic and intrinsic calibration.
- Lane localization with a nonparametric Bayes filter.
- Lane controller.

![Camera image](image.png)

**Fig. 2.** The lane following pipeline runs on-board at 10Hz with a resolution of 320x240 and a latency of 110ms. The purple text indicates prior information.

A. Infrastructure - The Duckietown Road Layer

The design of Duckietowns is an easily-understandable example of a formal specification: if the environment satisfies the specification, then the Duckiebot is guaranteed to be able to navigate it.

Duckietowns have two layers: one for the road and one for the signals. Lane following only depends on the road layer. The road layer is constructed by arranging the five types of interlocking tiles (Fig. 3). The precise color and positioning of the road markings are part of the specification and constitute a strong prior that can be used by perception.

![Tape types](image.png)

**Fig. 3.** Duckietowns are modular assemblies of five different tile types: (a) straight road, (b) a three-way intersection, (c) a four-way intersection, (d) non-road, and (e) turn. These tiles can be arranged to obtain arbitrary topologies. The tapes on the right hand side of a lane are white and solid, the tapes on the left hand side of a lane are yellow and dashed, and the stop lines are red and solid.

![Camera image before and after illumination compensation](image.png)

**Fig. 4.** Camera image before and after illumination compensation. Pixel distribution (blue dots), representative main colors (large colored blobs) and the transformation performed (white arrows) in RGB color space.

expresses our prior belief on illumination conditions and road marker colors. The transformations allowed are channel-separate:

\[
I_{\text{obs}}^{(i)} = a_i I_{\text{orig}}^{(i)} + b_i + n, \quad i \in \{R, G, B\}, \tag{1}
\]

where \(n\) is Gaussian measurement noise, and, for regularization, we assume a Gaussian prior for the parameters \(a_i, b_i\). The resulting 6 × 6 linear system of equations is

\[
\begin{pmatrix}
A_{\text{data}}, b_{\text{data}}
\end{pmatrix}
\begin{pmatrix}
p_I
\end{pmatrix}
= \begin{pmatrix}
b_{\text{reg}}
\end{pmatrix},
\tag{2}
\]

where \(A_{\text{data}}, b_{\text{data}}\) result from the observation model of (1), and \(A_{\text{reg}}, b_{\text{reg}}\) result from the regularization. The vector \(p_I\) is the vector of illumination parameters. The squared residual error in (2) gives an estimation of fit quality and allows the system to detect failure.

B. Illumination Compensation

Illumination variability is one of the challenges of computer vision. We use this procedure to teach:

- How machine learning can be used “in the loop” for autonomous robots.
- The importance of using prior information; in this case, the prior on the colors is given by the Duckietowns road layer specification.

In the baseline implementation of this functionality, we use the k-means clustering algorithm over a subsample of the sensor pixels in order to detect the main clusters on the road, and we match them to the expected clusters of red, yellow, white, and gray, according to the prior. We then fit an affine transformation in RGB colorspace between the detected clusters and their color-balanced version. Regularization
Dilation Canny edge detection
HSV color thresholding Dilation
Probabilistic Hough transform
AND camera image
lines 
segment detection

Fig. 5. Pipeline of the line segment detection algorithm.

Extrinsic camera calibration. The right image is the setup for the extrinsic camera calibration. The left image is an undistorted image captured via the camera on the Duckiebot. Once a homography $\mathbf{H}$ is estimated, it is possible to map from a point in image $x_i$ to a corresponding point on the ground $x_g$. The coordinate frame on the right image represents the camera coordinate frame of the Duckiebot.

D. Image Plane $\rightarrow$ Road Plane Homography

The next step consists in transforming the oriented line segments detections in image space to oriented points in 3D in body coordinates. Using the prior of a planar environment, it is possible to create a 1-to-1 map from image space to 3D space. First, the points are undistorted using the camera’s intrinsic calibration. Then the map is represented by a homography, $x_i \sim \mathbf{H}x_g$, where $x_i$ and $x_g$ are the points in the image and ground planes respectively, $\mathbf{H} \in \mathbb{R}^{3 \times 3}$ is the homography matrix, and $\sim$ represents equivalence up to scale.

Duckietown includes a calibration tool that allows one to estimate the camera’s intrinsics and extrinsics (Fig. 6).

E. Lane-Relative Estimation

To execute the lane following behavior, we must obtain an estimate of the Duckiebot's lateral position and orientation relative to the lane ($d$ and $\phi$ as indicated in Fig. 7). However, we do not need to know the longitudinal coordinate. This is an example of a “minimal sufficient statistics” for performing a control task.

A standard parametric approach to the estimation problem (such as a Kalman filter which employs a Gaussian assumption) would likely fail due to the presence of such a potentially high percentage of outliers and the nonlinearity of the process model. As a result, in the baseline implementation we employ a nonlinear non-parametric histogram filter [23].

The state of the car in the lane at time $t$ is represented by the reduced-dimension state: $x_t = (d_t, \phi_t)$, where $d_t \in [d_{\min}, d_{\max}]$ is the lateral displacement in the lane (with the $d = 0$ line being defined as the center of the lane) and $\phi_t \in [\phi_{\min}, \phi_{\max}]$ is the angle relative to the center axis as shown in Fig. 7. From the specification, we know the width of the lane, $w$, and the widths of the right (white) and left (yellow), $l_W$ and $l_L$ respectively, and we can use this information together to generate a unique hypothesis of the state from each segment detected by the road marking detector.

Lane Filtering. Left: Coordinate system for the lane estimation and control with some line detections and the specified dimensions of the lane and lane markings indicated. Right: The measurement likelihood as a result of processing one list of segments, such as the ones shown on the left, which is used in the measurement update step of the filter. Each green dot corresponds to a vote generated by an individual segment detected in the image.

Each incoming list of segments constitutes a single measurement. We use each segment to produce a “vote”. The votes are binned in a histogram and the entire histogram represents the measurement likelihood (Fig. 7).

Once all of the segments have been processed, the histogram is normalized and used to perform the measurement update in the Bayes filter.

F. Lane Controller

Once we have an estimate of the position and orientation of the Duckiebot in the lane, we use it to generate the control signals to drive down the lane. We have designed the curves explicitly such that the proportional error as the Duckiebot performs a turn (Fig. 3-(e)) is small enough such that it is still within the basin of convergence of a simple linear tracking controller. This alleviates the need to do any parameterization of the reference trajectory.

A proportional-derivative (PD) controller is employed, in which the control command takes the form $u(t) = k_d d(t) + k_\phi \phi(t)$ (the $k_\phi$ terms acts like a derivative).

We evaluated the performance of the lane following behavior on a square track with rounded corners with errors shown in Fig. 8. Under nominal lighting conditions, the reliability of the lane following behavior of the Duckiebot is very high, with a mean time to failure greater than 30 min and a cross track error within 0.15m at all times.

Fig. 8. Lane following performance: The lateral displacement and the relative angle of the vehicle in the lane during a loop traversal.
IV. NAVIGATION

The basic lane following pipeline acts as a nested inner loop to enable more complex and interesting behaviors. As an example, an overview of the navigation pipeline is shown in Fig. 9.

A. Infrastructure - Layer 2: The Signal Layer

The second layer of Duckietown is made of signals, such as signs and traffic lights. Signs are present in two varieties: (i) traffic signs; and (ii) street names. The traffic signs can indicate the traversability of an intersection, the type of an intersection (traffic light or stop sign), or some other important road information, as shown in Fig. 10. These signs alone contain all necessary information for localization and navigation. Each sign is additionally equipped with an AprilTag [1] to enable the parallel development of the algorithms that detect the signs from the components of the system that use these detections. We constrain the signs to be placed at one of eight fixed poses on a tile to: (a) guarantee that the signs are in the camera field of view of a Duckiebot at a stop line; and (b) enable the automatic generation of the metric feature map.

B. Map Representation

Since Duckietowns are composed of modular tiles, a map can be completely specified by a matrix of tile types (shown in Fig. 3). Additionally, we can specify the presence, type and position of any signs on these tiles. This matrix can be used to automatically generate two different map representations: a metric feature map used for localization, and a topological network graph used for planning. These two maps are shown in Fig. 11.

IV. NAVIGATION

The basic lane following pipeline acts as a nested inner loop to enable more complex and interesting behaviors. As an example, an overview of the navigation pipeline is shown in Fig. 9.

A. Infrastructure - Layer 2: The Signal Layer

The second layer of Duckietown is made of signals, such as signs and traffic lights. Signs are present in two varieties: (i) traffic signs; and (ii) street names. The traffic signs can indicate the traversability of an intersection, the type of an intersection (traffic light or stop sign), or some other important road information, as shown in Fig. 10. These signs alone contain all necessary information for localization and navigation. Each sign is additionally equipped with an AprilTag [1] to enable the parallel development of the algorithms that detect the signs from the components of the system that use these detections. We constrain the signs to be placed at one of eight fixed poses on a tile to: (a) guarantee that the signs are in the camera field of view of a Duckiebot at a stop line; and (b) enable the automatic generation of the metric feature map.

B. Map Representation

Since Duckietowns are composed of modular tiles, a map can be completely specified by a matrix of tile types (shown in Fig. 3). Additionally, we can specify the presence, type and position of any signs on these tiles. This matrix can be used to automatically generate two different map representations: a metric feature map used for localization, and a topological network graph used for planning. These two maps are shown in Fig. 11.

IV. NAVIGATION

The basic lane following pipeline acts as a nested inner loop to enable more complex and interesting behaviors. As an example, an overview of the navigation pipeline is shown in Fig. 9.

A. Infrastructure - Layer 2: The Signal Layer

The second layer of Duckietown is made of signals, such as signs and traffic lights. Signs are present in two varieties: (i) traffic signs; and (ii) street names. The traffic signs can indicate the traversability of an intersection, the type of an intersection (traffic light or stop sign), or some other important road information, as shown in Fig. 10. These signs alone contain all necessary information for localization and navigation. Each sign is additionally equipped with an AprilTag [1] to enable the parallel development of the algorithms that detect the signs from the components of the system that use these detections. We constrain the signs to be placed at one of eight fixed poses on a tile to: (a) guarantee that the signs are in the camera field of view of a Duckiebot at a stop line; and (b) enable the automatic generation of the metric feature map.

B. Map Representation

Since Duckietowns are composed of modular tiles, a map can be completely specified by a matrix of tile types (shown in Fig. 3). Additionally, we can specify the presence, type and position of any signs on these tiles. This matrix can be used to automatically generate two different map representations: a metric feature map used for localization, and a topological network graph used for planning. These two maps are shown in Fig. 11.

IV. NAVIGATION

The basic lane following pipeline acts as a nested inner loop to enable more complex and interesting behaviors. As an example, an overview of the navigation pipeline is shown in Fig. 9.

A. Infrastructure - Layer 2: The Signal Layer

The second layer of Duckietown is made of signals, such as signs and traffic lights. Signs are present in two varieties: (i) traffic signs; and (ii) street names. The traffic signs can indicate the traversability of an intersection, the type of an intersection (traffic light or stop sign), or some other important road information, as shown in Fig. 10. These signs alone contain all necessary information for localization and navigation. Each sign is additionally equipped with an AprilTag [1] to enable the parallel development of the algorithms that detect the signs from the components of the system that use these detections. We constrain the signs to be placed at one of eight fixed poses on a tile to: (a) guarantee that the signs are in the camera field of view of a Duckiebot at a stop line; and (b) enable the automatic generation of the metric feature map.

B. Map Representation

Since Duckietowns are composed of modular tiles, a map can be completely specified by a matrix of tile types (shown in Fig. 3). Additionally, we can specify the presence, type and position of any signs on these tiles. This matrix can be used to automatically generate two different map representations: a metric feature map used for localization, and a topological network graph used for planning. These two maps are shown in Fig. 11.

IV. NAVIGATION

The basic lane following pipeline acts as a nested inner loop to enable more complex and interesting behaviors. As an example, an overview of the navigation pipeline is shown in Fig. 9.

A. Infrastructure - Layer 2: The Signal Layer

The second layer of Duckietown is made of signals, such as signs and traffic lights. Signs are present in two varieties: (i) traffic signs; and (ii) street names. The traffic signs can indicate the traversability of an intersection, the type of an intersection (traffic light or stop sign), or some other important road information, as shown in Fig. 10. These signs alone contain all necessary information for localization and navigation. Each sign is additionally equipped with an AprilTag [1] to enable the parallel development of the algorithms that detect the signs from the components of the system that use these detections. We constrain the signs to be placed at one of eight fixed poses on a tile to: (a) guarantee that the signs are in the camera field of view of a Duckiebot at a stop line; and (b) enable the automatic generation of the metric feature map.

B. Map Representation

Since Duckietowns are composed of modular tiles, a map can be completely specified by a matrix of tile types (shown in Fig. 3). Additionally, we can specify the presence, type and position of any signs on these tiles. This matrix can be used to automatically generate two different map representations: a metric feature map used for localization, and a topological network graph used for planning. These two maps are shown in Fig. 11.
Fig. 13. A simplified version of the finite state machine used to control the vehicle. To see the full version, please visit our github repository.

Fig. 14. The Duckiebot as a hybrid system. A bank of controllers are continuously active. The active motor controller at any instant is determined by the FSM mode.

would be to plan the paths of multiple Duckiebots simultaneously simulating a fleet management system.

E. Finite State Machine

The macro level operation of the Duckiebot is regulated through a finite state machine (FSM), a simplified version of which is shown in (Fig 13). The transitions in the FSM are generated by asynchronous perception-based events, such as detecting a stop line. The mode is used to control the Duckiebot in a hybrid (discrete-continuous) system configuration [24]. A bank of controllers are developed corresponding to the different desired actions, and the mode from the FSM in Fig. 13 is used to select which controller is active and connects that controller to the wheels driver (Fig. 14).

F. Route Traversal

Once a plan has been generated, following it requires executing the correct sequence of turns at each intersection. For the example shown in Fig. 11-(b) this sequence corresponds to [s, f, s, r, f, s, f], where f, s, r, and l correspond to follow lane, go straight, turn right, and turn left, respectively.

The traversal of any individual intersection is achieved by executing the sequence states “LANE_FOLLOWING” → “INTERSECTION_TRAVERSAL” in a loop until the final destination is reached (assuming single-Duckiebot behavior at this stage we bypass the “COORDINATION” behavior and assume that the intersection is always free). The transition from LANE_FOLLOWING to INTERSECTION_TRAVERSAL is triggered by the arrival at the stop line. The transition from INTERSECTION_TRAVERSAL back to LANE_FOLLOWING is triggered by the lane estimate reporting convergence as measured by an entropy threshold.

Fig. 15. Intersections. (a): Traffic light intersection with four Duckiebots. (b): A stop sign intersection. In both cases the yellow shaded area corresponds to the bottom vehicle’s sensor field of view.

Fig. 16. Communication based on LED frequency detection.

V. MULTIROBOT BEHAVIORS

The most advanced behaviors involve the interaction of many Duckiebots navigating in the same Duckietown. In this scenario, the Duckiebots must coordinate to share the roads and intersections.

We consider two types of intersections: traffic lights, and stop signs, as shown in Fig. 15. Traffic light intersections are a simpler and centralized solution to intersection negotiation. Stop signs require inter-Duckiebot communication. Communication is decentralized and perception-based, employing LEDs to signal intentions. The situation is complicated by the fact that the Duckiebot “to the left” is outside of the camera field of view when the Duckiebot is at the stop line, as shown in Fig. 15.

A. Infrastructure - Traffic Lights and LEDs

The traffic lights are equipped with LEDs facing each incoming road, and are constructed in the same way as the Duckiebots, but without a chassis.

B. Blinking LED Detection and Interpretation

The communication system based on LEDs has two decoupled components: the detector, which captures a camera stream and determines positions and frequencies of all LEDs present in the scene; and the interpreter, which receives this information and labels each detection with a physical object (e.g. vehicle or traffic light) and a coordination message.

An overview of the LED communication scheme is shown in Fig. 16. The details of the LED detector and interpreter are shown in Fig 17.

The interpreter leverages the assumption that the LEDs detected above the horizon belong to traffic lights, and those below correspond to Duckiebot communication LEDs.

C. Coordination Behaviors - Traffic Lights

Traffic lights at intersections sequentially communicate a "go" signal to one of the incoming roads while signalling
D. Coordination Behaviors - Stop Sign Intersections

Stop sign intersections do not have any centralized infrastructure, so the Duckiebots communicate with one another to coordinate their turns through the intersection using LEDs mounted on each Duckiebot. As shown in Fig. 15, a Duckiebot at an intersection is not visible to a Duckiebot on its left due to the finite sensor field of view. The detection of the signal emitted by the other Duckiebots takes approximately 2s and the detection phases of individual Duckiebots are not synchronized. Intuitively, the decentralized negotiation protocol observes the following rules: (a) A Duckiebot yields to a Duckiebot on its right (since it sits outside of the field of view of the sensor of the Duckiebot to the right), (b) when two mutually opposing Duckiebots arrive at an intersection, the one that arrived earlier goes first, and (c) when the two opposing Duckiebots arrive simultaneously with respect to the detection precision, the Duckiebots resolve the ambiguity by waiting for a random amount of time before their next attempt to negotiate with the other Duckiebots.

The coordination scheme progresses through a series of states as shown in Fig. 13. In AT_STOP_CLEARING, the Duckiebot is waiting a predetermined time to guarantee that the intersection is free, in AT_STOP_CLEAR a Duckiebot is waiting with no other Duckiebots in the intersection to guarantee that it is clear to go, in RESERVING the Duckiebot is signaling (through its LED) that it will attempt to traverse, in CONFLICT two Duckiebots have attempted to reserve the intersection simultaneously, and finally in GO the Duckiebot starts to navigate the intersection.

E. Evaluation

We tested the coordination behavior on both types of intersections and demonstrated the performance for several hours. When the vehicles stopped at the stop line oriented along the lane, the coordination behavior was able to reliably schedule the vehicles to pass through the intersection and avoid collisions. The result of the coordination behavior at a stop sign intersection is shown in Figure 18.
TABLE III
MODE-BASED RESOURCE ALLOCATION

<table>
<thead>
<tr>
<th>Mode</th>
<th>Line Detector</th>
<th>Lane Filter</th>
<th>Stop Line Filter</th>
<th>Vehicle Detector</th>
<th>Sign</th>
<th>LED Decoder</th>
<th>LED Encoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOYSTICK_CONTROL LOCALIZE</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LANE_FOLLOWING COORDINATION</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERSECTION_TRAVERSE AVOID VEHICLE</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Due to the limited computational resource of the platform, applications (nodes) utilize an event-based and rate-limited processing and publication scheme: a node only publishes when the output is significantly different from the last published output and when the limit on publication rate is preserved.

2) Mode-Driven Perception: The most computationally intensive tasks tend to be those related to perception, particularly since we are using vision as the only sensing modality. The most complex multi-robot behaviors require the robot to perform a number of perceptual tasks (line detection, lane filtering, stop line filtering, sign detection, LED detection, LED decoding, vehicle detection) but not simultaneously. Therefore, we impose a template whereby upon mode transitions a set of switches are published. These switches are used by all nodes and allow them to determine the necessary input (images) at a given time. An overview of which perceptual modules are active in each of the FSM modes is given in Table III. We also control the resolution of the camera imagery based on the requirements of the given task, in a similar way.

VII. CONCLUSION

We have presented “Duckietown,” a flexible platform for autonomy education and research. We have leveraged precise specification and resource management in developing the system to enable a sliding scale of realism. We have targeted an autonomous driving application here, however we believe this model, with augmented capabilities, can be extended to autonomy in other less structured domains, such as air, sea, and perhaps space robotics.

All materials are available under open source/free software licenses; pointers to the materials can be found at the website duckietown.mit.edu. Our hope is that others in the robotics community will adopt the platform and contribute to its growth.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation, with National Robotics Initiative award IIS-1405259, and with Robust Intelligence award IIS-1318392. Additional support was given by the Toyota Research Institute (“TRI”) and the Ford Motor Company. However, note that this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. We would also like to acknowledge the contributions made by the students during the Spring 2016 semester of the MIT 2.166 class. Finally, we would like to thank Kirsten Bowser, the Duckietown Engineering Co. human resources coordinator.

REFERENCES