Team NCTU: Toward AI-Driving for Autonomous Surface Vehicles - From Duckietown to RobotX

Yi-Wei Huang[1], Tzu-Kuan Chuang[1], Ni-Ching Lin[1] Tony Hsiao[1], David Chen[1], CT Hung[2], Sean Lu[1], Sam Liu[1], Hao-Wei Hsu[1], Wen-Yang Liu[2], Simon Hsieh, Kim-Boon Lua[3], Chia-Hung Dylan Tsai[3], Chi-Fang Chen[2], and Hsueh-Cheng Wang*[,1]

***Abstract*— Robotic software and hardware systems of autonomous surface vehicles have been developed in transportation, military, and ocean researches for decades. Previous efforts in RobotX Challenges 2014 and 2016 facilitates the developments for important tasks such as obstacle avoidance and docking. Team NCTU is motivated by the AI Driving Olympics (AI-DO) developed by the Duckietown community, and adopts the principles to RobotX challenge. With the containerization (Docker) and uniformed AI agent (with observations and actions), we could better 1) integrate solutions developed in different middlewares (ROS and MOOS), 2) develop essential functionalities of from simulation (Gazebo) to real robots (either miniaturized or full-sized WAM-V), and 3) compare different approaches either from classic model-based or learning-based. Finally, we setup an outdoor on-surface platform with localization services for evaluation. Some of the preliminary results will be presented for the Team NCTU participations of the RobotX competition in Hawaii in 2018.**

I. Introduction

RobotX competition has brought together the top schools among the nations around oceans, and enhanced the developments of both hardware designs and software algorithms for perceptions and propulsions of unmanned surface vehicles. Perceptions play an important role together with state estimation and motion planning for autonomous vehicles. The participated teams have been working toward robustness to degradation caused by motion, scale and perspective transformation from different viewing positions, warp and occlusion, and variants of color from light condition, and speed and accuracy to support real-time decision-making.

Nevertheless, Building an autonomous unmanned marine system is challenging in many aspects. The hardware should be robust enough to maintain functionalities in different weather conditions. Waterproof design, compact electronics, together with cooling system are crucial for weather conditions of heavy rain, strong winds and the blistering sun. Due to the resource constrained of power and computation, realtime algorithms should be adapted for overall performance. Those challenges remain due to the uncertainties of the marine environments, and the lack of baselines and standardized evaluations because benchmarking in marine environments is hard. More importantly, there are still big questions arisen from the previous RobotX competitions.

[1]Department of Electrical and Computer Engineering, National Chiao Tung University, Taiwan. Corresponding author email: hchengwang@g2.nctu.edu.tw

[2]Department of Engineering Science and Ocean Engineering, National Taiwan University, Taiwan.

[3]Department of Mechanical Engineering, National Chiao Tung University, Taiwan.

Fig. 1: We proposed an autonomous surface/underwater system with modular design and is compatible both with ROS and MOOS. The system is capable of running in real robots, minitured robots and simulations. The capability of running in multiple platforms provide us better tools to develop deep learning algorithms.

1) What are the basic principles to manage multiple tasks and to advance the developments, instead of just a set of clever hacks on individual tasks?
2) Many of the tasks have some hand-tuned parameters that were optimized for the tasks during the competition. How could the developed system/techniques be generalized to other scenarios, on other robots, or in other environments?
3) Can we use simulation environments to facilitate the developments? What are the gaps between real and virtual environments and how to bring the gaps closer?

Team NCTU's technical approaches are motivated from the challenges and big questions above. The recent mega trends of AI have fostered researchers in robotics, machine learning, and other fields together. In particular, we wish to adopt the principles of the AI Driving Olympics (AI-DO) [1], [2] hosted in NIPS 2018 into RobotX competition. The AI-DO is developed by the Duckietown community. The project was initiated in MIT in 2016 and is now offered as university courses in ETH Zurich, University of Montreal, and Toyota Technological Institute at Chicago, and National Chiao Tung University (NCTU) in 2017. Duckietown [3] is an open, reproducible, and inexpensive robotic education and research platform. A team of vehicles are built upon Robot Operation System (ROS) and include an onboard monocular camera and an embedded computer. A miniaturized city (Duckietown) with roads, signage, and obstacles is designed to tackle the problems of autonomy. The Duckietown platform started to

embrace Docker and deep learning in 2018 and host the competitions of a few tasks via learning approaches such as Convolution Neural Network (CNN) and Deep Reinforcement Learning. Here we wish to transform such methodologies into the domain of unmanned surface vehicles.

We summarize our contributions as follows:

1) With the principles of containerization (Docker), we designed and built an autonomous unmanned marine system with the hardware and software that is compatible for two commonly-used middlewares: ROS and MOOS. The containerization allows to a) develop under different middlewares, b) deploy on real and simulation environments, and c) modularize and compare algorithms for the RobotX tasks.
2) The uniformed AI agent framework with observations and actions facilitate comparisons between classic and learning-based algorithms. In particular, we tackle the problems of a) the placard detection and 3D object recognition carried out in classic feature-based vs. learning-based methods, b) obstacle avoidance via classical methods vs. deep reinforcement learning approaches.
3) We built an outdoor on-surface motion capture system based on 3D LiDAR and 2D vision-based approaches as benchmarks for further research purposes.

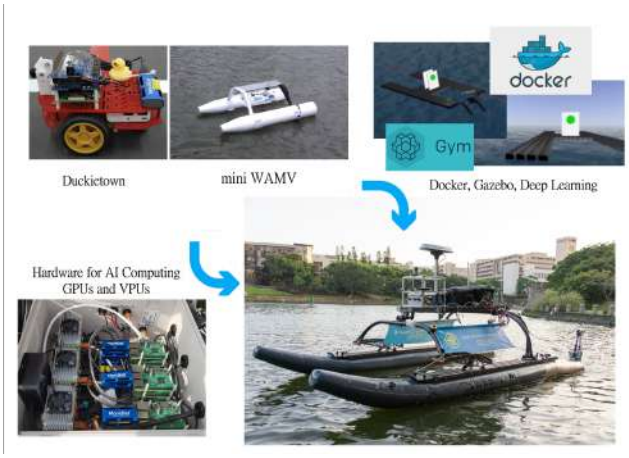

Fig. 2: From duckietown to RobotX

II. Literature Review

A. RobotX 2014 and 2016

RobotX has already been held for 2 times in 2014 and 2016. The 2014 championship team MIT-Olin's [4] uses IvP-helm [5] to achieve multi-objective optimization, such as transit to target waypoint while avoiding obstacles. The MOOS [6] framework operates the speed and heading which are generated by IvP Helm and also process data, for instance, GPS, object map, PID controller, and so on. The 2016 championship team Team NaviGator AMS [7] utilize ROS [8] as middleware. That year added a new mission about underwater shape identification. The team designed underwater vehicle Anglerfish [9] by themselves which can be controlled by the NaviGator ASV via a 30-metered tether providing power and ethernet. Most of the teams in the past two competitions apply the algorithms about detection and motion without using machine learning methods.

B. Learning Approaches for Mobile Robots

Recently, deep Convolutional Neural Networks (CNN) have been used to achieve autonomous trail or lane following. Giusti et al. [10] tackled autonomous forest or mountain trail-following using a single monocular camera mounted on a mobile robot, such as a micro-aerial vehicle. Unlike the previous literature, they focused on trail segmentation and used low-level features to develop a supervised learning approach using a deep CNN classifier. The trained CNN classifier was shown to follow unseen trails using a quadrotor. Deep driving [11] categorizes the autonomous driving work into three paradigms. *Behavior Reflex* is known as a low-level approach for constructing a direct mapping from the image/sensory inputs to produce a steering motion. This is done by means of a deep CNN trained by labels generated from human driving along a road or in virtual environments. *Mediated perception* is the recognition of driving-relevant objects, e.g., lanes, traffic signs, traffic lights, cars, or pedestrians. The recognition results are then combined into a consistent world representation of the cars and immediate surroundings. *Direct perception* falls between mediated perception and behavior reflex. It proposes to learn a mapping from an image to estimate several meaningful states of the road situation, such as the angle of the car relative to the road and the lateral distance to lane markings. With the state estimation, other filters or FSMs and controllers can be applied. The forest-trail-following vehicle in [10] belongs to behavior reflex, whereas the Duckietown falls into the direct perception paradigm.

In the context of deep learning, the most commonly used method for simulation to real environment is transfer learning [12], [13]. By gathering data from the target domain in addition to the source domain closes the gap between simulation and the reality. [14], [15] uses data alignment to tackle with simulation to real problems for robotics arm pose estimation. [16], [17] proved that doing simple but precise tasks in simulations can be transferred to real world robots for more complex and general tasks. [18] connected the layers of deep learning models trained with simulation data and real world data together. It resulted features trained in virtual environments usable in real world scenarios.

III. Design Strategy

A. Enabling Hardware Systems for AI Computing

For the first time building such a sophisticated system, it is very crucial to do every possible evaluation beforehand. It just happens that we've been working on a project called "Duckietown", a miniaturized AI self-driving car platform that runs on ROS. The cars a.k.a "duckiebots" use AI to navigate around and communicate with other duckiebots. We think it would be interesting to apply the AI technology to our Autonomous Maritime System(AMS). The duckiebots could provide a miniaturized environment for testing our algorithms.

B. From Simulation to Real Environments

In addition to the duckietown platform as an simulation, using software simulation is also important for our proof of

Fig. 3: Hardware

concept. We use ROS as middleware, so Gazebo [19] is the obvious solution due to the compatibility to ROS. Using the advantage of ROS, we could simply interface with simultation and real robots by publishing and subsribing corresponding messages. A few parameters should be adjusted to fit the real environment.

C. Containerized Algorithms for Deployable Softwares

To deploy a variety of different algorithms on various environments, software dependencies may be troubling. A decent solution for it is to containerize algorithms. By using Docker, every algorithm is like a building block, which is easy to switch to one another. The plug and play feature on docker containers provide us with simple deployment on both simulation and real environments.

D. Comparing Deep Learning to Classic Approaches

Deep learning has influenced the robotic research in the past few years. The challenge has always been proving deep learning is better than classic approaches. We believe that both are good but in different aspects. Classic approaches may deal with a problem with a really great performance, on the other hand deep learning methods might be more reliable to unexpected environment changes. Our goal is to compare the two and discuss the pros and cons of each method.

IV. Vehicle Design

The WAM-V system has one multi-enclosures on payload which can be seen in Fig. 3, which split into two power enclosures, one sensor enclosure, one controller enclosure, and one hydronphone enclosure.

A. Propulsion

Two forward thrusts are used for the differential drive system. Each motor is capable of 80 lbs of thrust. It is the simplest to implement with the least amount of cost. We once considered holonomic drive which enables the mobility of the USV, but it requires at least three thrusters and it has much more complicated control motion. In addition, we think there must be some reason that no holonomic drive USV is currently working in the ocean.

B. Sensor System

The sensor system is build as an sensor tower and an indipendent sensor for underwater ocustic array for the competition. The sensor tower has 4 level and mounted with different sensors show in Fig. 3. To sense the environment, there are a 3D LiDAR, three depth cameras, an IMU, and a GPS be used. These sensor could localized our USV and target objects. There is also an underwater hydronphone array for detecting underwater pinger task.

C. Computation

This paper presents a multi-computing units system. There is an Industrial Personal Computer(IPC) communicate with other computing units, such as NVidia Jeson TX2, and Raspberry Pi 3 for different purposes. The NVidia Jeson TX2 is used to get depth camera information and sent it to the IPC. The Raspberry Pi 3 is used to control motors. The system is connected by ethernet and communicate with ROS.

TABLE I: Computation Units

Computation Units	Number	Processor	Spec/Used
ASUS Laptop (GX501)	1	GPU	GeForce GTX 1050 Ti
NVidia Jeson TX2	3	GPU	Pascal 256 CUDA cores
Raspberry Pi 3 (Neural Compute Stick)	9	VPU	Myriad 2 Vision Processing Unit
Raspberry Pi 3	1		Propulsion
Raspberry Pi 3	1		Visual Feedback
Raspberry Pi 3	1		Launching and Recovery
Raspberry Pi 3	1		Detect and Deliver

V. Software System Design

A. Overall System

The software is built upon the ROS interface, several nodes are built for distinct functionalites and they communicate with each other via ROS messages and services. The ROS package tree contains nodes shown in Figure 4

There are nodes that deal with path planning, localization, control, perception, classification, etc. Some of the ROS nodes are contained in docker for easier deployment.

B. AI Agent

To generalize the usage of this system, we introduce the concept of AI agent. The idea is to build it so its compatible for OpenAI gym [20].

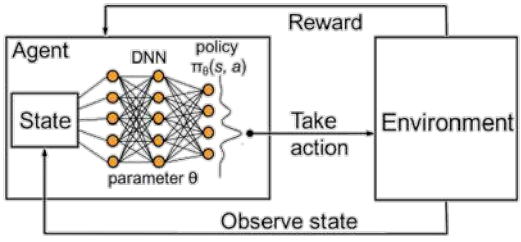

Fig. 5: The relationship between the AI agent and the environment

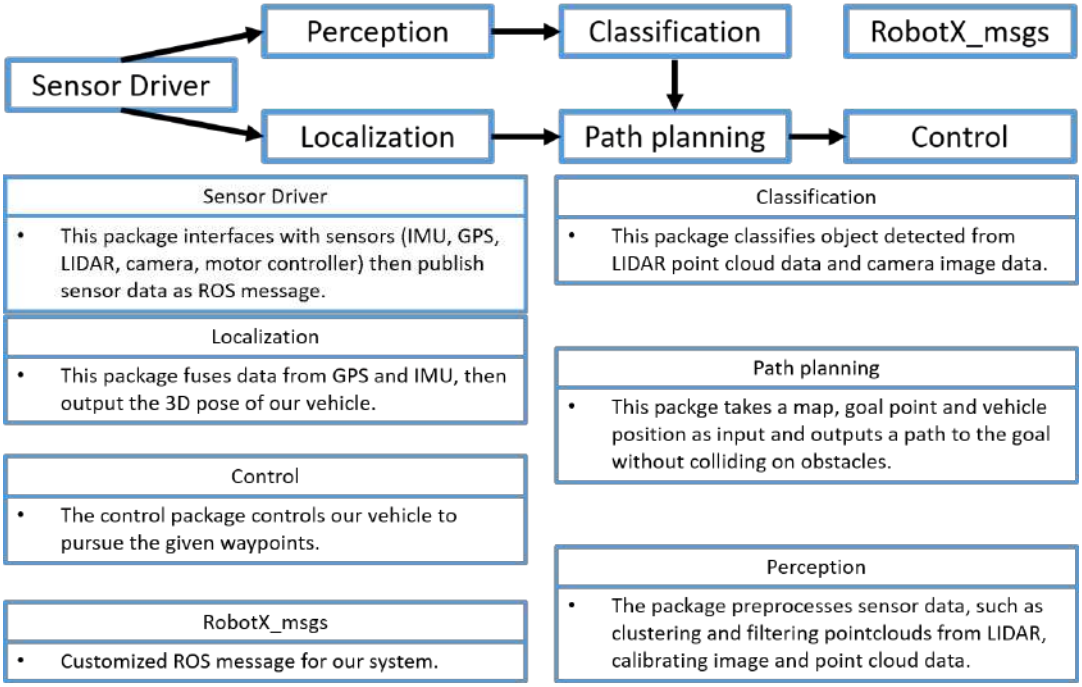

Fig. 4: The software architecture

Figure 5 shows that the agent should output observations and rewards for each input action. The design of the agent should output higher rewards for correct actions. Then the algorithm tries to interact with the agent and learn what the AI agent wants it to learn. So for each task there is at least one AI agent and each agent could have different observation from various of sensors.

C. MOOS and ROS

The Mission Oriented Operating System (MOOS) [6] and the Robot Operating System (ROS) [8] are both robotic frameworks which provide the communication between different robots and components.

ROS is an open source project which is compatible of using C++ and Python as development languages. In ROS, processing units are called ”Nodes” and they communicate with each other by ”Messages” and ”Services”. A central monitor program named ”ROS Master” controls all the status of nodes and handles all the ”messages” and ”services” exchanged among the ROS ”nodes”. Since there are majority of users and thousands of modules have been developed, ROS already became the most commonly-used middleware for robotics. Lots of robots such as PR2, Atlas, UR5, Turtlebot use this open source software as their software framework.

MOOS-IvP is the combination of two open source projects: MOOS and Ivp Helm. MOOS is developed by the University of Oxford which and is designed to be the core of autonomy middleware. While IvP Helm is developed by MIT used for multi-objective optimization between competing behaviors. Most of the MOOS-IvP projects are applied in the field of unmanned surface vehicle and underwater acoustics and large influence in the field of marine robots for years.

This paper [21] from Georgia Tech presented some advantages and disadvantages of these two middlewares. MOOS is more oriented towards onboard publish-subscribe architecture by using its community database ”MOOSDB” and more lightweight. In addition, there are hundreds of executable behaviors, simulations, and MOOS Apps developed by the marine robotics community. This leads to easy development of your own algorithms for USVs. On the other hand, ROS is more general and provides multi-platform services which is less painful for system integration. Moreover, it has many of the low-level device control interfaces and components int the hardware abstraction layer. ROS also included Gazebo, a 3D simulation for general robotics application, on the contrary MOOS only offers 2D simulation. However, development in Gazebo is much complicated than in MOOS.

We chose to use both frameworks for taking advantages from each. In order to cooperate the two frameworks, we use MOOS-ROS Bridge [22] to communicate these two robotics middlewares. We modified and added a few features for better performance in applications needed by our system.

D. Localization

One of the critical things to do is to obtain the vessel’s pose. Without the poses, the vessel wouldn’t know where it is not to mention navigate to a position. We implement the localization with a Hector GPS and a microstrain 9 DoF IMU. GPS datum and IMU datum are fused together using a Gaussian filter. This filter stablizes the localizaton output. Posistion and Orientation are calculated separately.

E. Control and Navigation

Another crucial feature is to control the vessel and to navigate from one point to another. We designed our WAM-V’s with a differential drive motion model, therefore linear and angular could be controlled separately. We use a PID controller for heading control and a cascade PID controller for

the position. We chose the cascade PID controller to control position because with both position and velocity feedback, it could provide better performance for tasks i.e. station keeping.

For navigation, we implement the pure pursuit algorithm for waypoint navigation. The pure pursuit algorith makes the trajectory smoother by reducing sharp turns. Figure 6 shows the trajectory for our WAM-V executing waypoint navigation. The waypoints are a the points of a square with a 15 meter edge. It show the turns on the corners are smoothed out.

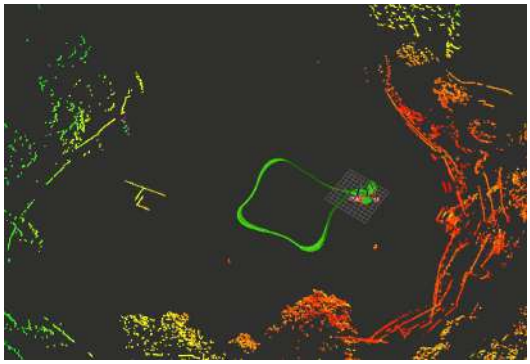

Fig. 6: The green line shows trajectory of the vessel executing a square movement. Other colored points are the pointcloud from the LiDAR data.

F. Object Classification

Due to the precarious weather and changeable lighting conditions which can strongly impact the color of the object, we chose to use the pointclouds gathered from the LIDAR for object classification. A brief introduction of the algorithm pipeline is as follows. First, for the preprocessing stage, we apply RANSAC and noise filter to remove the points from the sea level and random noises, leaving the objects remaining in the pointcloud. After the preprocessing process, we apply a clustering algorithm to separate each object for each another. Then we project each object's pointcloud to X-Y, Y-Z and X-Z planes (according to the LIDAR's coordinate frame) and saved each plane projection to one of the RGB channels of an image. (Figure 7) We now obtain the RGB image and named it as the "flattened pointcloud". This "flattened pointcloud" remains some of the 3D information and at the same time is a more compact datatpe compared to the raw pointcloud.

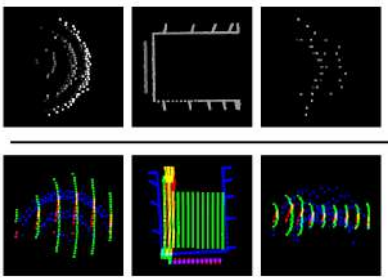

Fig. 7: Each target object's pointcloud is projected to the X-Y, Y-Z and X-Z planes then the three flattened images saved to the R, G, B channels of a 2D-image

However, the projection is related to the LiDAR's orientation, different rotations of the WAM-V cause the "flattened pointcloud" of the same object to differ a lot. This may lead to bad performance for classification afterwards. To cope with this problem, we simply transform the coordinate frame to the object's frame which sets the origin to the center of the object and the x axes. This resulted the flatted image be consistent as the WAM-V change the orientation. Figure 8.

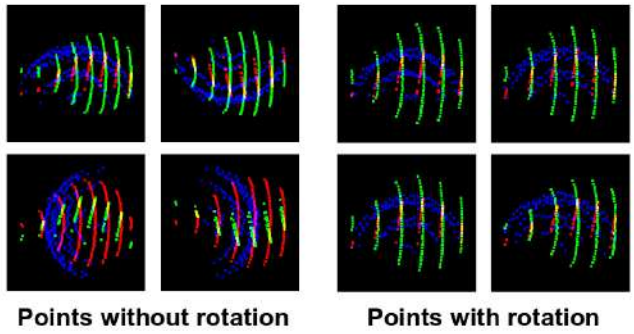

Fig. 8: With the rotation technique, the flattened pointcloud from the same object but 4 different viewing angles looks almost identical compared to those with out the rotation process. This improves our classification performance.

For each class we gather approximately 700 "flatten pointcloud" images, with 500 images from Gazebo virtual environment and 200 images from the real world (Bamboo lake in NCTU, Taiwan). We have 5 different classes, which are obstacle buoys, totem buoys, the dock and the box for the detect and deliver task.

After collecting our dataset, we train them with CaffeNet and got an accuracy of 95.4% in gazebo testing dataset Figure 9 and 87.7% in real world testing dataset. The decrease of accuracy in the real world testing dataset is mainly because the object is too far from the LIDAR. The obtained sparse pointcloud leads to worse performance. But overall, it is still a decent algorithm for the tasks in RobotX.

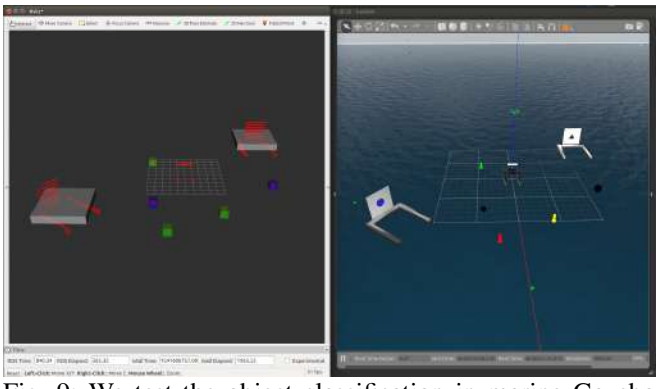

Fig. 9: We test the object classification in marine Gazebo environment, and get an well performance

G. Placard Detection

For the placard detection algorithm in the task identify the dock, most of the traditional methods based on feature matching suffered from illumination variance, perspective transformation, and occlusion. And the methods were limited

to scaling up with more types of placards. To overcome those challenges, a deep CNN classifier was utilized to identify the placards. The training data was collected from virtual and semi-realistic environments which are more accessible. We collect data by driving the vehicle in different trajectories shown in Fig. 10, the data contained different perspective transformation, and some with occlusion. Our CNN model is based on CaffeNet [23] but using network surgery techniques to reconfigure it for having 10 output classess. The 10 classes represent nine different types of placards and one background. After training from virtual and semi-realistic environments, the CNN classifier is applied to perform placard identification with MSER region proposals input in real-world environment. As shown in Fig. 11, the green, black, and white boxes represent the green circle class, background class, and predictions under threshold respectively.

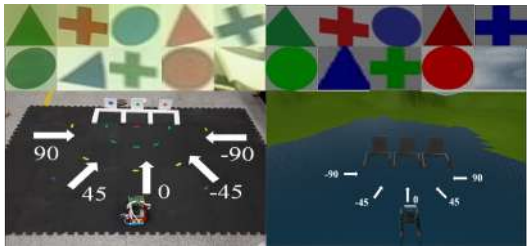

Fig. 10: The training data for CNN placard classifier was collected from virtual and real environments by driving vehicle following different trajectories. The data contains data with a variety of perspective transformation, with occlusion, and illumination variance. Left: semi-realistic environment Duckietown [24]. Right: virtual environment Gazebo.

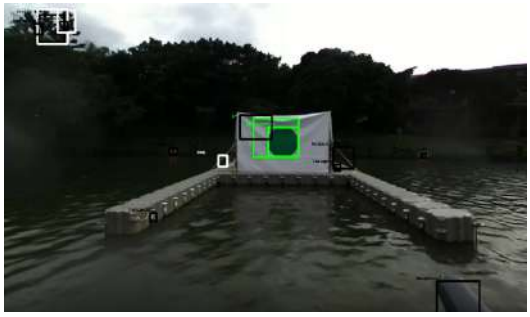

Fig. 11: The CNN classifier trained from virtual and semi-realistic environments is applied to identify placard in real-world environments. Green boxes: green circle class. White boxes: predictions under threshold. Black boxes: background class.

H. Docking Motion

Inspired by the previous work [25], an end-to-end deep CNN model was deployed to predict three motion probabilities with single RGB image input. The imitation learning process is to gather training data by contolling a vehicle mounted with three different heading cameras torwards a docking bay. Shown in the left of Fig. 12, the data collected from three cameras was automatically labelled into three classes: turn left, go straight, turn right. The probabilities of three motion classes outputs were then calculated to the motor commands of the differential-driven vessel. The right Fig. 12 showed the images from a single camera mounted on ASV with different heading angles to the docking bay in real-world environments. And the images could be predictied into correct motion classes for the following docking motion control.

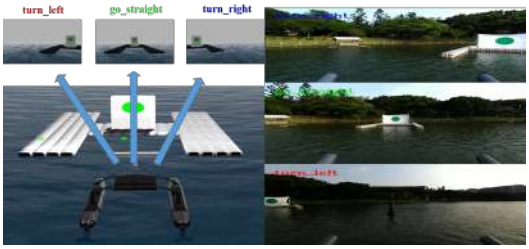

Fig. 12: An end-to-end deep CNN model was used to predict motion classes for the docking process with a single RGB image as input. Left: training data collected by three cameras with different mounting angles. The datum is automatically labeled into three motion classes. Right: real-world motion class predictions with images of a single camera equipped on the ASV.

I. Totem Circling

For the totem circling task, we implemented it in the classical way. Totem detection is achieved by the object detection discussed in previous subsection therefore we could get the spatial information of the totems. Then we formulate the circling problem as figure 13. The two variables d and phi are used to describe the vehicle's position. If the vehicle is performing a circulating action, d will be the rotating radius R and phi would be zero. Therefore, by using 2 PID controllers to control d and phi, the circulation motion is done.

This algorithm is implemented in both the Duckietown and Gazebo simulations(Figurefigure 13). Both of them are robust enough to complete at least 30 rounds.

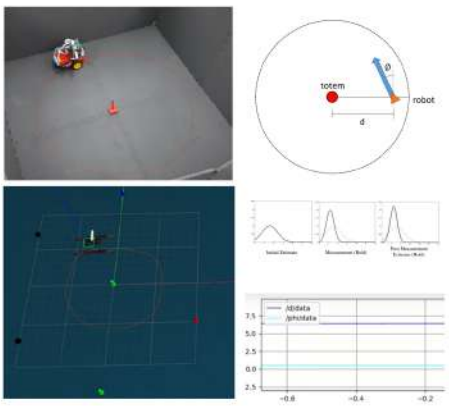

Fig. 13: Robots doing totem circling in two simulation environments. The red lines show their trajectories Top Left: Duckietown. Lower left: Gazebo. Top right: the problem formulation of robot state: d and phi. Lower right: The d and phi through time for the gazbebo simulation.

J. Obstacle Avoidance

Obstacle avoidance is one of the basic features for mobile robots and there are plenty of different algorithms to solve this problem. In this section, we are discussing three algorithms for this problem: MOOS obstacle manager, minimum angle real-time path planning and deep reinforcement learning. Different techniques may all solve the problem, but the meaning behind it could be quite different. The first method is the obstacle avoidance feature included in MOOS. It is mainly composed of three parts. The pFeatureTracker App manages objects' convex polygon, position, and map. The pObstacleMgr App sends the OBSTACLE_ALERT to the behavior. The BHV_AvoidObstacle is the behavior function in MOOS-IvP Helm which maneuvers the vehicle to avoid the obstacles. In response to different situations, the user has many configuration parameters to adjust for the AvoidObstacle Behavior. This method is based on a priority policy to avoid obstacles which may be useful in many scenarios.

The second method we implemented the minimum angle real-time path planning, which is based on [26]. Basically, the algorithm forms a line segment by connecting the start point and the goal point, then checks whether this line collides with objects or not. If collision occurs, then it finds the points on the right side and left side of the obstacle as candidates. Then the turning angles of both the candidates are calculated. The candidate with a smaller angle will be picked and considered as a new starting point. Then we repeat the process of checking the new line segment that is connecting the new starting point and goal point. By doing this iteratively until the starting point is close to the goal point, we could find a path that connects the original starting point to the goal point without hitting obstacles.

The last method we implemented for obstacle avoidance by Reinforcement learning. This is a well known problem for reinforcement learning. Work [27] has been done to solve this problem using a 2D LiDAR as input with two classic reinforcement learning algorithms: Q learning and SARSA. Results are very promising in simply built simulation environments. However, with a more complex sensor input, it is common that there are thousands or even millions of robot states. This resulted in the traditional reinforcement learning almost impossible to train. Deep reinforcement learning overcomes this problem by replacing policy tables with Deep Neural Networks(DNN). We've designed our software as an AI agent(mentioned previously), so it is simple to implement it for deep reinforcement learning. For this task, first we set the action space as $\langle$*go_straight*, *turn_left*, *turn_right* $\rangle$. Then we downsampled the LiDAR data as the observation. Finally, we set the reward as r a function of observation o and action a.The design of the reward function is simply penalize collisions and rewarding more for straight movement compared to turns for preventing the agent spinning on the same spot. By training iteratively, the DNN learns the policy. On an average of 100 episodes the agent learns to avoid obstacles in 100 steps. While this method seems workable and simple. However, some strange behaviors sometimes occur on some agents, for example, some agents only turn right to avoid obstacles. The agents only learn from the reward functions, so it makes total sense it would learn this result. Therefore it is important to carefully design the reward function.

K. Underwater Manipulation

For the underwater manipulation task we modified the inspector 2 from Seadrone Inc as our AUV. The AUV is equipped with 5 thrusters which provided 3 DoF in translation and 1 dof in orientation(yaw) movement. Onboard sensors include a pressure sensor, leak sensors, IMU and a camera with extra led lighting. We added an underwater gripper from Blue Robotics for grasping rings for the task.

Using classic image processing algorithms, we are able to detect the pose of the ring, steer the AUV to grasping point using PID control and retrieve the ring to the surface.

Fig. 14: AUV detecting and retrieving ring autonomously

VI. Experimental Results

All the tests and trials of our AMS are done in the lake located in our campus. This is our first year doing this so we did a lot to get our WAM-V in the lake. We managed to build a launching platform which consists of a slope with a winching system. Neither of us had any experience so it was a trial-and-error process. We also built the equipments from totems buoys to docks by hand.

After the WAM-V is launched into the lake, we did some test for basic functionalities(e.g. localization, motion control and perception), the problem we encountered is that we couldn't do a quantitative analysis for all the experiments related to localization. We could only do qualitative analysis and just come up to a more general conclusion. For instance, we did a trial of 10 rounds for the WAM-V performing obstacle avoidance and it completed 9 rounds.

Therefore, later we decided to build a vision-based localization system Figure. 15 by placing apriltags [28] on floating pontoon cubes anchored with concrete blocks. This could provide us more quantitative results for our AMS localization, giving us more prove whether our algorithms work better.

VII. CONCLUSIONS

Our development of the whole system for the RobotX 2018 competition is meaningful for us being part of the unmanned autonomous vessel research. We focus not only on making our system work in the competition but also building a foundation for further researches. Our final goal is to bring a better platform and modularized software for easier development and deployment that every developer could use. This could bring the community closer together by sharing work based on the same framework and also let researchers focus on

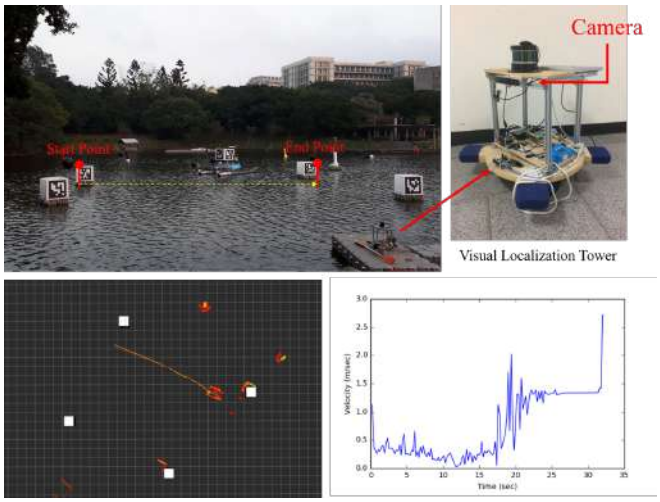

Fig. 15: The system to localize marine robotics using RGB camera with AprilTag [28]. Top left: Vision-based localization system in real environment with our WAM-V. Top right: Vision-based localization tower with and RGB camera. Lower left: Trajectory in real environment using vision-based localization system. Lower right: WAM-V velocity calculate from the vision-based localization system.

parts they care about. In our case we brought some deep learning / deep reinforcement learning algorithms and hope to bring more cutting-edge deep learning technologies to the field of marine robotics.

ACKNOWLEDGMENT

The research was supported by National Chiao Tung University Innovative Creative Technology (NCTU-ICT), Ministry of Science and Technology, Taiwan (grant numbers 107-2623-E-009 -005 -D and 105-2511-S-009-017-MY3), . We are also grateful for the help by Santani Teng, Robert Katzschmann, Ilenia Tinnirello, and Daniele Croce.

REFERENCES

[1] The AI Driving Olympics. [Online]. Available: https://nips.cc/Conferences/2018/CompetitionTrack

[2] The AI Driving Olympics. [Online]. Available: https://AI-DO.duckietown.org

[3] Duckietown MIT. [Online]. Available: http://duckietown.mit.edu/

[4] A. Anderson, E. Fischell, T. Howe, T. Miller, A. Parrales-Salinas, N. Rypkema, D. Barrett, M. Benjamin, A. Brennen, M. DeFillipo, *et al.*, "An overview of mit-olin's approach in the auvsi robotx competition," in *Field and Service Robotics*. Springer, 2016, pp. 61–80.

[5] M. R. Benjamin, "Interval programming: A multi-objective optimization model for autonomous vehicle control." 2003.

[6] M. R. Benjamin, J. J. Leonard, H. Schmidt, and P. M. Newman, "An overview of moos-ivp and a brief users guide to the ivp helm autonomy software," 2009.

[7] D. Frank, A. Gray, K. Allen, T. Bianchi, K. Cohen, D. Dugger, J. Easterling, M. Griessler, S. Hyman, M. Langford, *et al.*, "University of florida: Team navigator ams," in *RobotX Forum*, 2016.

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[9] A. Gray and E. Schwartz, "Anglerfish: an asv controlled rov," in *29th Florida Conference on Recent Advances in Robotics (FCRAR), Miami, FL*, 2016.

[10] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[11] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[12] X. Peng, B. Sun, K. Ali, and K. Saenko, "Learning deep object detectors from 3d models," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1278–1286.

[13] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2686–2694.

[14] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," *CoRR, abs/1511.07111*, 2015.

[15] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4068–4076.

[16] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *arXiv preprint arXiv:1511.03791*, 2015.

[17] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3357–3364.

[18] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[19] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator." in *IROS*, vol. 4. Citeseer, 2004, pp. 2149–2154.

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[21] M. E. West, T. R. Collins, J. R. Bogle, A. Melim, and M. Novitzky, "An overview of autonomous underwater vehicle systems and sensors at georgia tech," 2011.

[22] K. DeMarco, M. E. West, and T. R. Collins, "An implementation of ros on the yellowfin autonomous underwater vehicle (auv)," in *OCEANS 2011*. IEEE, 2011, pp. 1–7.

[23] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*. ACM, 2014, pp. 675–678.

[24] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S.-Y. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H.-C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. D. Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017.

[25] T.-K. Chuang, N.-C. Lin, J.-S. Chen, C.-H. Hung, Y.-W. Huang, C. Tengl, H. Huang, L.-F. Yu, L. Giarré, and H.-C. Wang, "Deep trail-following robotic guide dog in pedestrian environments for people who are blind and visually impaired-learning from virtual and real worlds," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–7.

[26] H.-Z. Zhuang, S.-X. Du, and T.-J. Wu, "Real-time path planning for mobile robots," in *2005 International Conference on Machine Learning and Cybernetics*, vol. 1, Aug 2005, pp. 526–531.

[27] P. F. Lucas Manuelli, "Reinforcement learning for autonomous driving obstacle avoidance using lidar." [Online]. Available: http://www.peteflorence.com/ReinforcementLearningAutonomousDriving.pdf

[28] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3400–3407.