A Deep Vision Landmark Framework for Robot Navigation

Abhijith R. Puthussery, Karthik P. Haradi, Berat A. Erol, Patrick Benavidez, Paul Rad and Mo Jamshidi
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249, USA
Email: qaw164@my.utsa.edu, dxq821@my.utsa.edu, berat.erol@utsa.edu
patrick.benavidez@utsa.edu, paul.rad@utsa.edu, moj@wacong.org

***Abstract*—Robot navigation requires specific techniques for guiding a mobile robot to a desired destination. In general, a desired path is required in an environment described by different terrain and a set of distinct objects, such as obstacles and particular landmarks. In this paper, a new approach for autonomous navigation is presented using machine learning techniques such as Convolutional Neural Network to identify markers or objects from images and Robot Operating System and Object Position Discovery system to orient to the marker, calculate the distance and navigate towards these markers using depth camera.**

***Index Terms*-Autonomous Navigation, Robot Operating System, Machine learning, TensorFlow, Convolutional Neural Network, Differential drive robot.**

I. Introduction

Autonomous mobile robots are becoming prominent in commercial, industrial and scientific applications. Different kinds of mobile robots are being employed into many situations, such as logistics, self driving cars, search and rescue, space robotics, exploration of dangerous environments, agriculture, etc. Progress in developing efficient and effective autonomous vehicles is a challenge to researchers and developers.

One of the primary objectives in mobile robotics is to develop navigation routines for a robot to navigate through indoor or outdoor environments with precision and speed using sensory feedback. To do this, these robots require high computational power and powerful sensors. In order to achieve this, real-time decision making based on uninterrupted sensor data from the environment is required. For a robust navigation routine, a robot must be able to adapt to variations in the surroundings, typically using a system which collects visual and depth sensor data.

Navigation sensors provide information about the vehicle states (position, orientation, speed, etc.) and the objects in the surrounding environment. Different sensors such as Global Positioning System (GPS), vision, and laser range scanner are used in autonomous mobile robot systems as primary sensing systems. Ultrasonic sensors [1] and RFID [2] are also been used as primary sensors but are less common. Other sensors such as odometer, inertial measurement unit (IMU), digital compass, and gyroscope are typically used as secondary sensors to complement the primary sensing systems [3].

Global Positioning System (GPS) based guidance technology has been widely used in autonomous navigation. A GPS often combined with other sensors provide more accurate navigation information [4]. The most common problems with using GPS for navigation involve obstruction of line-of-sight to satellites, and interference from other RF sources. Also GPS based navigation in indoors is highly inaccurate.

Vision sensors have been widely used in mobile robot navigation because of the cost effectiveness of vision sensors and their capability to provide extensive information. Vision systems are becoming more common in navigation applications such as localization, map construction, path following and obstacle avoidance. One big disadvantage using vision sensors is the influence by varying ambient lightening conditions especially in outdoor environments. But nowadays vision sensors with better performance are used which can work flawlessly in outdoor environments [5].

Laser scanners have the benefits of high resolution and large field of view. The laser scanner is one of the most popular devices in outdoor applications. It determines the relative distance of objects in the surrounding area by measuring time of flight of laser pulses. One important advantage of laser sensors over visual systems is the ability of providing robust ranging data for object detection and localization [5]. This enables the robot to operate more reliably at different weather and ambient illumination conditions. The main disadvantage of laser scanners is that they are expensive.

In this paper we propose a vision-based navigation algorithm using machine learning to identify objects in the environment as markers in real-time. The robot will be able to navigate autonomously both indoors and outdoors via these marker locations from Object Position Discovery system. This method can be used in various situations such as logistics, search and rescue, hospitals and homes. The rest of the paper is structured as follows. Section II provides the system architecture. Section III presents details on the experimental setup. Section IV presents the experimental implementation.

* This work was supported by Grant number FA8750-15-2-0116 from Air Force Research Laboratory and OSD through a contract with North Carolina Agricultural and Technical State University.

II. Related Work

A framework for mobile navigation was proposed by Rasmussen which exploits the continuity of image sequences by tracking features in images to guide the robot and provide predictive information [6]. This image-based method provides an accurate motion without a precise geometric model of the surrounding. But a human guide is required to recognize the features from the images during the initial phase of this navigation process. Later, a new developed system introduced, landmark patterns are designed as bar codes, which can be detected in real-time images taken with robot's camera [7]. Pillai and Leonard developed a monocular SLAM-aware object recognition system, which localize and recognize different objects in an environment from images taken from multiple views [9].

Previous related work by the authors of this paper follows. A framework for navigation and target tracking system for mobile robot was presented using 3D depth image data and used color image recognition, depth camera data and fuzzy logic to control and navigate the robot [8]. Design of a testbed for Large-Scale autonomous system of vehicles was proposed for localization, navigation and control of multiple networked robotic platforms by using cloud computing in [10]. A real-time cloud-based VSLAM was provided in [19] with enhancements to reduce processing time and storage requirements for a mobile robot. A visual SLAM based cooperative mapping study with cloud back-end proposed the importance of the object identification for the mobile navigation and localization [11]. Furthermore, a cloud architecture for large scale systems of autonomous vehicles was presented in [18]. A foundation for deep neural network control was provided in [20]. The approach proposed in this paper does not require human guided scene calibration to identify objects initially, bar-codes or any special detection methods or move around the environment. This approach also does not require taking images from multiple angles before identifying the objects thereby saving time in recognizing the object from its surrounding and can be implemented real-time without much changes.

III. Architecture

A. Overview

The proposed autonomous navigation method is separated into two phases: marker detection and robot navigation.

1) Marker Detection Phase: Machine learning is used in the Marker Detection Phase of the navigational algorithm. TensorFlow and the Inception v3 Image Recognition Engine are used in this phase. The robot captures images of the environment around it and transfers these images to modified Inception v3 Image Classification Engine. Objects identified with the Inception V3 engine are classified with the classes found in the ImageNet database. This engine records the position and orientation of markers in a file for use by the controller in the robot navigation phase.

2) Robot Navigation Phase: In the navigation phase, the robot prompts the user to identify a marker that the robot should move to. When the user selects a desired marker, the robot locates the position and orientation of the target in its database. Then the robot calculates the pose of target relative to its current location and orients towards the target. When the robot has oriented towards the object, the robot moves towards it using on-board depth camera for feedback. A flowchart of the process is provided in Fig.1.

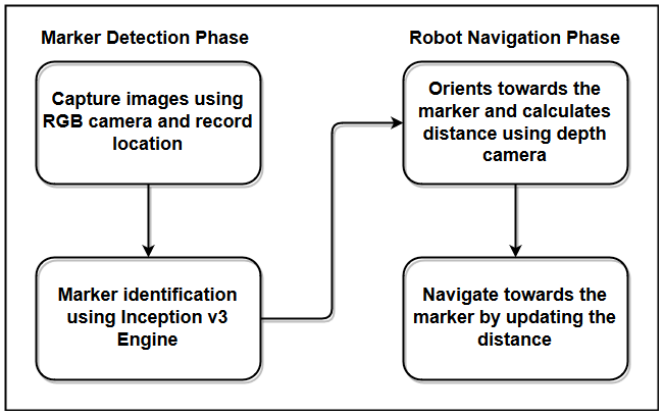


Fig. 1. Marker detection and navigation system overview

B. Algorithm

The block diagram of proposed algorithm for marker detection and robot navigation are provided in Figure 2 and Figure 3, respectively.

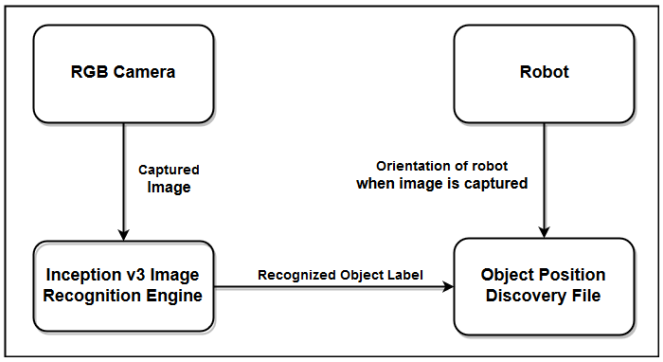


Fig. 2. Algorithm for marker detection phase

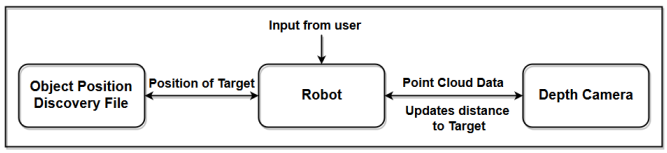


Fig. 3. Algorithm for robot navigation phase

1) Marker Detection Phase: Multiple images are captured using the Asus Xtion on-board camera while the robot rotates from a fixed position at constant speed. Simultaneously, the position and orientation of the robot is recorded when the images are captured and this data is stored in a Object Position Discovery (OPD) file. These images are passed through a custom TensorFlow Inception v3 image classification engine. The top 5 classes of classified objects from the image are listed by their match scores. From this list, the object with maximum score is written to OPD file along with the position of the robot when the image was captured. After detection is complete, the OPD file is cleaned using a python script which removes empty rows and columns and multiple labels of same class. The marker detection phase ends here.

2) Robot Navigation Phase: In the navigation phase, all the recognized markers are prompted to the user for a selection of a target to navigate to. Once user input for the desired object is acquired, the robot scans through the OPD file and identifies the user-selected object and its corresponding position. If multiple entries of the same object are present within a certain proximity of each other, an average of their positional data is calculated. Once the final position is calculated, the robot turns and orients itself towards the desired object. After orienting towards the object, the distance to the object from the robot is calculated using the on-board depth camera. The robot navigates towards the marker, constantly updating the distance to travel and stops when it reaches user-defined threshold distance to the object.

C. Inception v3 Image Recognition Engine

Inception v3 is a convolutional network developed by Google. Inception V3 builds an image classification system for selected image classes by applying a prior trained deep learning model. This model can classify an image across 1000 categories supplied by the ImageNet academic competition and achieves 5.64% top-5 error. Inception architecture of GoogLeNet was also designed to perform well even under strict constraints on memory and computational budget. The computational cost of Inception is also much lower than VGGNet.

The Inception v3 engine is written using TensorFlow. TensorFlow is an open source software library for machine learning developed by researchers and engineers working on the Google Brain Team. TensorFlow is defined as an interface for expressing machine learning algorithms and an implementation for executing such algorithms [12].

IV. Experimental Setup

A. Hardware

The robot used for this autonomous navigation method is a Kobuki TurtleBot 2 [15], a low-cost, open source differential drive robot by Yujin Robot. It has reliable odometry sensors, long battery life and provides power for an external sensor and actuators. It consists of a mobile base, Asus Xtion RGB-D camera, and an ODROID XU4 octacore microcomputer.

B. Test Bed

The test bed was designed for clear classification of defined markers against a plain background. Thus the amount of variables in the image classification algorithm is minimized with the plain background and the features can be more easily classified. The various markers are positioned at different locations within the test bed. A picture of test bed is provided in Fig.4.

Fig. 4. Experimental testbed consisting of a Kobuki Turtlebot2 and objects to classify

V. Software Implementation

A. Marker Detection Phase

ROS packages are used for the interface for the connectivity to the Kobuki Turtlebot2 and the ASUS Xtion RGB-D camera. The Kobuki ROS launch file *minimal.launch* is executed first to bring up the robot model and start the basic nodes of the robot. The OPENNI2 ROS launch file *openni2.launch* is used to connect to OpenNI-compliant devices such as the Asus Xtion. The robot keyboard controller launch file *keyop.launch* provides control for the initial stage to rotate the robot while taking pictures.

A general purpose consumer laptop is used to supplement the robot's on-board computer. The ODROID microcomputer can run the image recognition engine but adds a significant level of latency to the operations of the robot. ROS is used to facilitate the connectivity between the microcomputer and the laptop. Software developed using the ROS middleware only needs to be aware of the IP Address of the microcomputer to subscribe to the camera topic.

Image recognition is accomplished by the TensorFlow framework based Inception v3 engine. The default engine is designed such that it is not integrated with ROS. An interface was added to integrate Inception V3 with ROS. Positions of the markers are identified by the robot's orientation data topic and the yaw angle topic. The combined engine and ROS interface subscribes to odometry data and transforms the quaternion-based heading to an Euler angle.

A new ROS package, *kobuki_odom* was created and contains a new script *odom_listener* which subscribes to current

position of the robot (x, y, z) from the *nav_msgs/Odometry* type message published by the robot in Quaternion form. Another requirement is the current angle of the robot, which is collected from a *geometry_msgs/Quarternion* type message published from the robot. The angles in Quaternion form and converted to converted to Euler form. The conversion of the angle is included in the *odom_listener* script. The converted angles are published as a new node *odom_listener* with a *geometry_msgs/Vector3* message type. This node is used in many scripts for Kobuki navigation.

Once the position of robot is obtained, commands for starting image capture are executed. Once the live image feed from Asus Xtion camera is up and running, images can be saved at regular intervals. For saving images, the *image_saver* tool of *image_view* (a viewer for ROS image topics) is used which subscribes to image feed of camera. When the above *image_saver* code is executed, images with a resolution of 640x480 are saved whenever the "save" feature of the "image_saver" ROS service is called. The saved image is passed to *Inception v3* engine immediately via a bash script.

Inception v3 engine scans through the image and identify different objects using Convolutional Neural Network model along with the object label and score of top 5 probable objects. Out of these, the label with highest score are recorded in Object Position Discovery (OPD) system using JSON. The position is recorded in OPD when the image is captured and the label is added against the position after passing through the image recognition engine. There can be multiple labels for the same object in ImageNet, but only the first label is used. All other label entries are omitted. Also, all empty rows are deleted to make the OPD file light and readable. A python script *filecleanup* is executed to clean up the OPD file and write to an updated JSON file.

B. Autonomous Navigation Phase

The user is prompted with the recognized object labels via a python script *IdentifyLocation*. The purpose of the prompt is to determine which object the robot is supposed to move towards. After receiving input from user, the updated OPD file is scanned for the location of the target. The average location a selected label from the initial scanning phase is used to determine the object location. The average location of an object is used as the existence of a particular object in multiple images is likely the case during the scanning phase.

The algorithm for orienting the robot towards the target is shown in Figure 5. First, the position of object is read from OPD file and copied to a variable *required_angle*. The current yaw angle of the robot is collected from *odom_listener* ROS node. The difference of required angle and current yaw angle is determined and stored in a new variable, *error*. The algorithm will minimize the error by rotating the robot towards the object until the error is approximately 0, in which case the robot stops and is oriented with the target.

The distance from the robot to the target is calculated from measurements of the Asus Xtion depth camera. Due to lack of proper laser scanner, a ROS package *depthimage_to_lasercan*

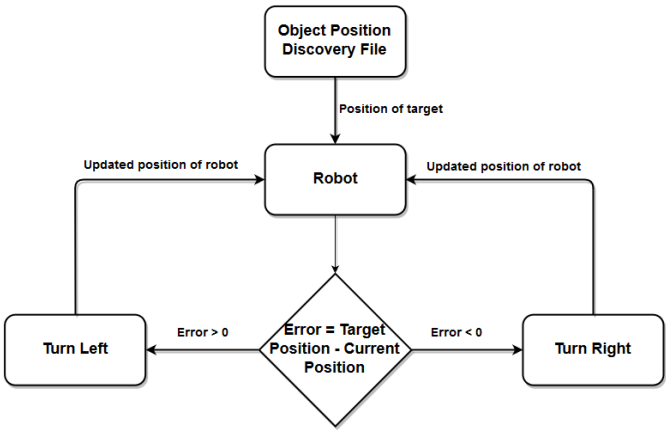


Fig. 5. Algorithm for orienting towards the target

is used to convert the point cloud obtained from the depth image to a 2D laser scan in a *sensor_msgs/LaserScan* type topic [17]. The stock parameters for the depthimage_to_lasercan node are modified for the application (*scan_time* is set to 0.5s, *range_min* is reduced from 0.45m to 0.10m so the robot can detect the object even if it is very close to it).

After *depthimage_to_laserscan* ROS package is up and running, a controller written in python (*move_kobuki*) is executed which subscribes to */scan(sensor_msgs/LaserScan)* type topics and odometry data from the Kobuki in the form of *Twist(geometry_msgs)* type messages. This script also publishes a ROS topic named *dist_range*. From the depth image values, the shortest distance value is selected and passed to a variable *minimum_distance*, which will be the distance for the robot to travel to the required object. Required distance to travel is calculated by taking the magnitude of the difference of *minimum_distance* and current position of Kobuki and stored in variable *required_distance*. Command velocity *cmd_vel* is passed to */mobile_base/commands/velocity* to move the Kobuki in forward direction. When the robot reaches near the required object, the *cmd_vel* is forced to 0. To avoid collision with the object, a safe distance is maintained between the robot and object. This threshold distance is set to 40cm.

VI. Experimental Results

When the codes are executed as mentioned above, the Kobuki was able to rotate, take multiple images and pass them to modified Inception v3 engine. Figure 6 shows the Kobuki rotating from its initial position, taking images at fixed intervals.

Figure 7(a) shows the terminal in which Inception v3 engine using TensorFlow framework identifies different objects from the image. In the terminal, the identified object along with its accuracy (score) can be seen.

Figure 7(b) shows the output of the *ImageIdentifier* script after cleaning up the OPD file using *FileCleanup*. All recognized objects were listed and ask the user to select the required object. Figure 7(b) also shows the terminal listing the objects and waiting for user input. The robot orients itself towards

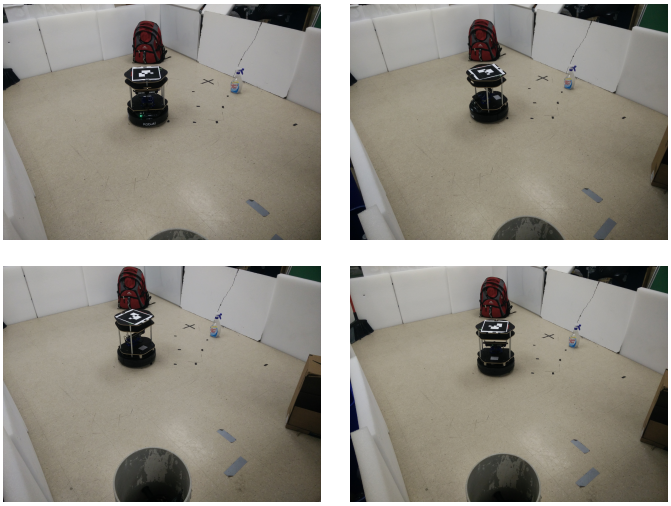
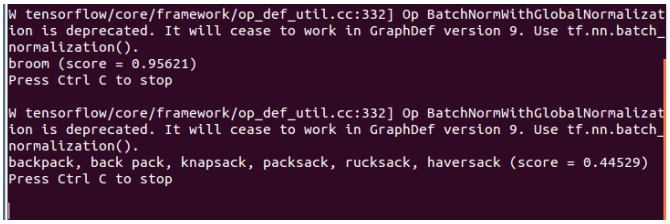
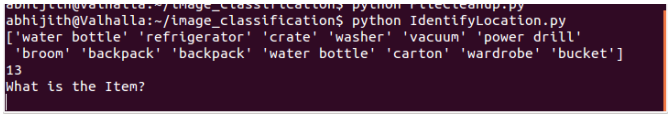


Fig. 6. Kobuki Turtlebot2 scanning the area environment with the vision sensor



(a)



(b)

Fig. 7. (a) Terminal window displaying output of Inception v3 engine. The displayed output shows a "backpack" and a "broom" were discovered in the last two images. (b) Terminal displaying the output of the "IdentifyLocation" script. The displayed output shows a prompt to the user asking for the name of the discovered object.

the required object once it gets an input from the user. At this point the *depthimage_to_laserscan* package was activated. Point cloud output and laser scan output can be seen in Fig. 8(a) and 8(b). The distance of the object is calculated from this data.

The engine classifies the image and the object recognized were registered in a Object Position Discovery (OPD) file. The yaw angle from the initial position of the robot to each object was written in same OPD file against the recognized object label as shown in Table I.

After finding the distance to move, the robot starts moving towards the object. Figure 9 shows the movement of robot taken at various instances. The Kobuki will stop in front of the object keeping a safe distance, shown in Figure 9 (c).

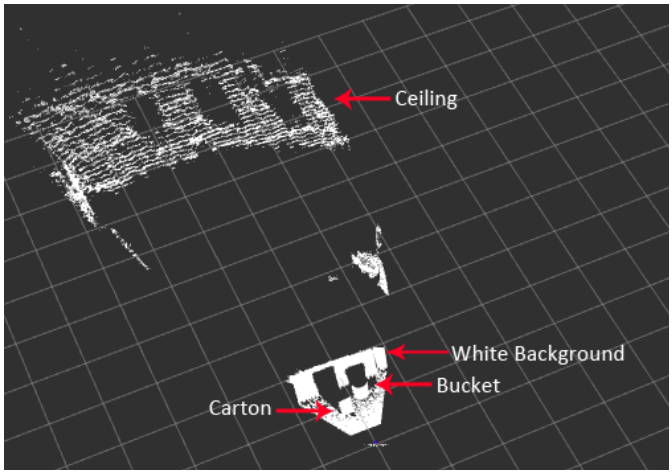
VII. Conclusion and Future Work

A deep vision framework for robot navigation was proposed in this article. The proposed method extracts markers from images captured by the robot and navigates to desired targets autonomously. Our algorithm for marker detection and robot navigation was successful in achieving the desired results. The custom image recognition engine works well with the developed ROS interface. The experimental results show the proposed algorithm works well with good accuracy.

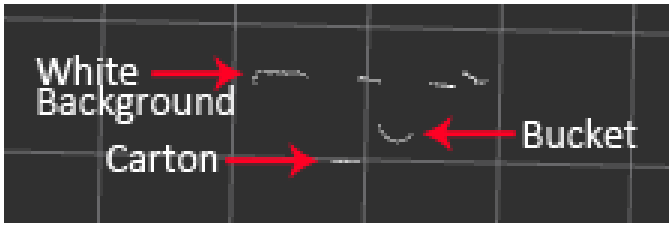
Improving the capability of the deep network to handle noise will reduce the reliance on using plain white backgrounds in the environment. This will allow for the robot to work in a variety of workspaces without much trouble.

TABLE I
Sample of a generated Object Position Discovery file containing each recognized object and the corresponding angles to the objects

Label	Position
backpack	-0.738972
water bottle	-1.579348
refrigerator	-2.237512
crate	-2.977357
washer	2.605951
vacuum	1.895428
power drill	1.23866
broom	0.535292
backpack	-0.342085
backpack	-1.067443
water bottle	-1.79734
carton	-2.469466
wardrobe	3.00396
bucket	2.257234



(a)



(b)

Fig. 8. (a) ROS sensor visualization output displaying the PointCloud image taken from the Asus Xtion RGB-D camera. (b) Top view of simulated laser scan ("/scan" ROS topic) from Asus Xtion

(a) (b) (c)

Fig. 9. Kobuki Turtlebot2 robot moving towards the ”bucket” object found in Figure 8(b)

File input and output operations of storing and recalling images, and image metadata were computationally time consuming. Optimizations to the operations of storing and recalling data in the system will greatly enhance the speed of many repetitive operations.

As accurate object detection within an image is not necessarily the focus of deep learning research, it is often least solved. In our case, we solved the problem by averaging the location of an object in an image over a 360 degree rotary scan of the robot about a certain point in the environment. A filtering approach will be developed for future versions of this system to get a more accurate pose of the object.

To better identify objects that should be in a given scene, a deep network can be tailored to a given environment. For instance, instead of 1000 different classes, we could use just 100 of the common items in a given scenario. Optimizing the Inception model further to the environment will also provide classifications faster with higher accuracy. Different datasets can also be applied to determine which one works with the highest accuracy in real life scenarios. In real-time situations, the scanning process can be executed again to improve the accuracy of object detection and navigation.

Navigation of the robot in this system can be improved via use of Simultaneous Localization and Mapping (SLAM). Scanned objects in a scene can be utilized as landmarks that are used for localization in SLAM. A system that identifies the temporal static-ness of certain objects would be useful in determining which of the detected markers would be a viable choice for use in mapping.

REFERENCES

[1] Masunga N., Arnould P., Mobile Robot Navigation in Indoor Environements by using the Odometer and Ultrasonic data, International Technical Conference on Circuits/Systems, Computers and Communications, 1999, pp. 2821-2828

[2] Srilakshmi S.,Venkata Phani Raja K., A Mobile Robot Navigation System Using RFID Technology, IOSR Journal of Electronics and Communication Engineering, Dec. 2012, Vol. 4,Issue. 3, pp. 15-20.

[3] Shalal, N. and Low, T. and McCarthy, C. and Hancock, N. (2013) A review of autonomous navigation systems in agricultural environments. In: SEAg 2013: Innovative Agricultural Technologies for a Sustainable Future, 22-25 Sept 2013, Barton, Western Australia.

[4] Sukkarieh, S. and Nebot, E.M. and Durrant-Whyte, H.F., High integrity IMU/GPS navigation loop for autonomous land vehicle applications, IEEE Transactions on Robotics and Automation, Jun 1999, Vol: 15, Issue: 3, pp 572-578.

[5] Bonin-Font, F., Ortiz, A. & Oliver, Visual Navigation for Mobile Robots: A Survey, Journal of Intelligent and Robotic Systems, 2008 Issue. 53, pp.263-296. DOI:10.1007/s10846-008-9235-4.

[6] Rasmussen, C. & Hager, G. Robot Navigation using Image Sequence, AAAI-96 Proceedings, 1996.

[7] Briggs, A., Scharstein, D., et al. Mobile Robot Navigation using Self-Similar Landmarks, IEEE International Conference on Robotics and Automation, 2000, pp.1428-1434.

[8] Benavidez, P. & Jamshidi, M., Mobile robot navigation and target tracking system. International Conference on System of Systems Engineering, June 2011.

[9] Pillai, S. & Leonard, J., Monocular SLAM Supported Object Recognition, Robotics: Science and Systems (RSS), 2015.

[10] Labrado, J. D., Erol, B. A., Ortiz, J., Benavidez, P., Jamshidi, M., & Champion, B. (2016, June). Proposed testbed for the modeling and control of a system of autonomous vehicles. In System of Systems Engineering Conference (SoSE), 2016 11th (pp. 1-6). IEEE.

[11] Erol, B.A., S. Vaishnav, et al. Cloud-based Control and vSLAM through cooperative Mapping and Localization, World Automation Congress, 2016, pp.1-6. DOI=10.1109/WAC.2016.7582999.

[12] TensorFlow https://www.tensorflow.org/

[13] Szegedy, C., Liu, W., Jia, Y., et al. Rethinking the Inception Architecture for Computer Vision, 2014, arXiv:1409.4842.

[14] Inception v3 schematic diagram, https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html

[15] Kobuki Turtlebot 2, http://kobuki.yujinrobot.com/about2/

[16] ROS Wiki. ROS Package, http://wiki.ros.org/Packages

[17] Biswas J., Veloso M., Depth camera based indoor mobile robot localization and navigation, IEEE International Conference on Robotics and Automation, 2012, DOI: 10.1109/ICRA.2012.6224766.

[18] Miratabzadeh, S. A., Gallardo, N., Gamez, N., Haradi, K., Puthussery, A. R., Rad, P., Jamshidi, M. (2016, July). Cloud robotics: A software architecture: For heterogeneous large-scale autonomous robots. In World Automation Congress (WAC), 2016 (pp. 1-6). IEEE.

[19] Benavidez, P., Muppidi, M., Rad, P., Prevost, J. J., Jamshidi, M. (2015, April). Cloud-based realtime robotic visual slam. In Systems Conference (SysCon), 2015 9th Annual IEEE International (pp. 773-777). IEEE.

[20] Mehdi, R., Rad, P., Jamshidi, M., Deep Learning Control for Complex and Large Scale Cloud Systems, Intelligent Automation and Soft Computing (AUTOSOFT), DOI: 10.1080/10798587.2017.1329245.