Autonomous Drone for Object Localization

Achiraya Klinpipat
Jiraphapa Jiravaraphan
Monrada Juycharoen

Bachelor of Engineering Program in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Academic Year 2017

KMITL-2018-IC-B-003-006

Thesis - Academic Year 2017

Bachelor of Engineering in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Title: Autonomous Drone for Object Localization

Authors:

1. Achiraya Klinpipat Student ID: 57090001
2. Jiraphapa Jiravaraphan Student ID: 57090005
3. Monrada Juycharoen Student ID: 57090009

Approved for Submission

.....

(Dr. Ukrit Watchareeruetai)

Advisor

Date 15 June 2018

Abstract

Achiraya Klinpipat 57090001
Jiraphapa Jiravaraphan 57090005
Monrada Juycharoen 57090009
Dr. Ukrit Watchareeruetai Advisor
Academic Year 2017

Keywords: drone, server, mobile application, localization, mapping, Simultaneous Localization and Mapping(SLAM), Unmanned Aerial Vehicle(UAV), Robot Operating System(ROS), navigation, convolutional neural network, image processing, image detection.

Nowadays as drone technology is being improved and used more in daily life, more advanced applications are being developed from drone systems. Thus in order to make drones smart enough to support further application, other technological aspects should be applied to drone systems.

This thesis proposed a system that applies different technological aspects including object recognition, path planning, indoor navigation, and control via mobile application to build a prototype project as a base for further drone-based applications. The goal is to develop a system that should be able to fly autonomously and localized object.

The system composed of three major components which are 1) drone controlled by Raspberry Pi 2) server, and 3) mobile application. Sensors are attached to the drone to retrieve data from the environment. Raspberry Pi handles these data along with drone control. These data are passed down to the server to be further processed and accomplished the process of object recognition, path planning, indoor navigation, and communicating with mobile application. The mobile application handles communication between user and system control.

Despite the minor challenges faced, the system stands as a proper prototype to be the base for further drone system extension needed for future applications as aimed.

Acknowledgments

During the development of the proposed system of Autonomous Drone for Object Localization, we had done many research and studies about technologies that could be used to develop drone system. There are many challenges faced during the project, including limitation of hardware resources and access, hardware interfacing, ROS systems, SLAM, and implementation learning curves. However, we were able to accomplished and gain a lot of knowledge from the project development process.

First of all, we would like to express our deepest appreciation and sincere gratitude to our advisor, Dr. Ukrit Watchareeruetai, for his professional guidance, expert advice, support, and encouragement along the development of the project, as well as his supervision that made this project came into existence.

We would like to especially give an acknowledgment to HiveGround Co., Ltd. This project would not have been possible without their kindness of supporting essential equipments for the project development, welcoming workspace, their provision of expertise in the knowledge of robotics & drones.

We are also thankful to our lecturers and committee members, including, Dr. Rutchanee Gullayanon from Engineering Faculty, Dr. Asst. Prof. Dr. Chaiwat Nuthong, Dr. Isara Anatavarasilp, Dr. Visit Hirankitti, and Dr. Nathhapong Jungteerapanich for their professional guidance, logistical support, and necessary equipments.

Lastly, we would like to express our gratitude to the Computer Vision lab, seniors, families, and colleagues for the suggestions, criticism, co-operation, motivations, and encouragement. Also International College, KMITL for providing suitable workspace and grant the essential equipments, which supports us for the project development.

May 15, 2018

Table of contents

List of figures

List of tables

with 36% to $4.5 billion increases in revenue.

As technology takes steps forward, so do drones. They have been put together with other technological aspects to serve wide variety of services. They quickly gain new physical, artificial intelligence, computer vision, GPS, and more to serve broad ranges of need. Today, more advanced applications can be developed and implemented to the drones. In agriculture, using advanced sensors and computer vision, the drone can help provide map of the crop for the farmer to use in identifying areas of crop variation and damage. Moreover it can also diagnoses problems, such as irrigation problems or pesticide, and prescribes appropriate solutions, such as variable rate nitrogen applications. Companies like Agremo, Aglytix, and Skymatics have been taken advantages of drone usage to come up with solutions that allow them to calculate for crop damage from counting and analyzing plants. In the field of mapping and surveying, drone is being used by the GIS professionals in several mapping projects to save time and cost on surveying. Sergio Lugo Serrato, the civil engineer of Skylab Industries, along with his team had received a project from the MexicoâĂŹs public government agency to map 1,000 kilometers of highway, and the whole project was being done using a drone. Construction as one of the biggest industry to adopt drone finds it very useful to use drone in keeping track of job site through video streams from the camera attached to the drones. The famous American construction company Brasfield & Gorrie uses drone-generated 3D model to compare actual earthwork to site plans. Swarms of drones learn to cover an area collectively and could build a 3D map of an entire building. Also in the late 2013, Amazon had come up with the delivery system using drone known as Amazon Prime Air. The service can deliver product of up to 5 pounds within 30 minutes and it can cover the distance up to 10 miles.

As mentioned earlier, the integration of drone with other technological aspects could lead to much more potential. Object recognition is one of the aspect. It would be beneficial if the drone can to recognize object, and that is when machine learning comes in. Knowing objects in the environment allow drone to be applied to many application, for example detection of traffic on the road. Indoor navigation is another aspect to be further explored. The indoor drone can be very helpful in search and rescue mission. In case of post-disasters, some hazardous area can not be reached by men such as area of dangerous chemical and high

temperature. In this kind of situation drone can be used to search for survivors or even locate the source of problems. Some additional technological aspects could be added to the drone system to support indoor navigation. A process called, *SLAM*, or Simultaneous Localization and Mapping has been introduced for this purpose. It is the process of incremental map construction for localization. Its main function is to aggregate observations obtained by sensors in order to obtain information and map the environment. Also by applying the SLAM output, path planning is also possible. Drone flight can be made autonomous and extended to even further application. This leads to the core of this project which is to extend the usage of drone by applying and integrating different technological aspects to the drone system.

1.2 Proposed system

Autonomous Drone for Object Localization (ADOL) is the system that apply different technological aspects to the drone system including object recognition, path planning, indoor navigation, and control via mobile application to build a prototype project as a base for further drone system extension for future application.

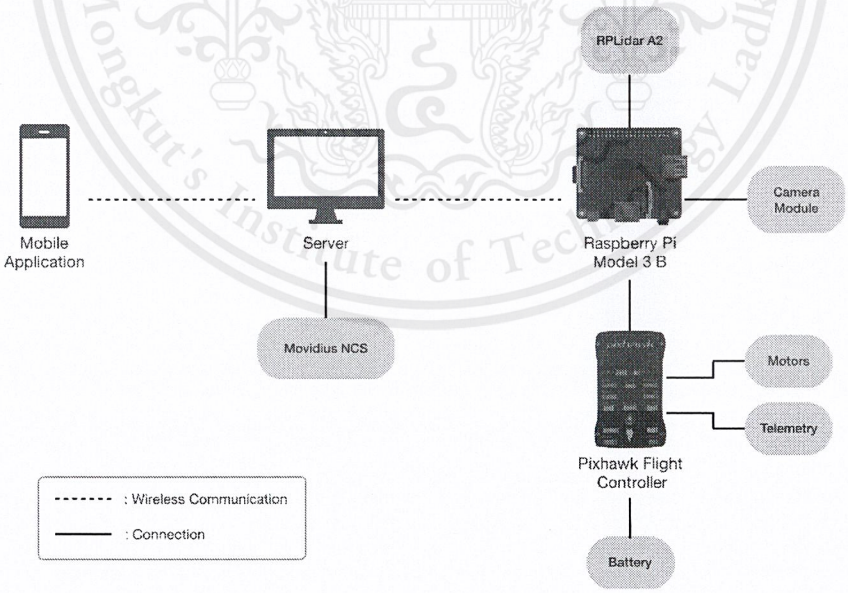


Figure 1-1. Overview of the proposed system.

As shown in Figure 1-1, the system will be composed of the interconnection between Raspberry Pi placed on the drone as the main board, pixhawk for flight controller, the server for computation and communication, and mobile application for user interaction. This project will be focusing on studying the technological aspects mentioned in the paragraph earlier of how it can be best applied to the drone system. The drone used will be self-assembled, along with attaching additional sensors. The SLAM technique used for localization and mapping will be adopted from the existed module called, Hector SLAM. Convolutional Neural Network (CNN) machine learning model is being used to yield object recognition. The navigation module will be adopted from texttt*navfn* library. Along with adopting the stated modules, the drone control system and server are to be implemented and the application use to communicate to the drone will be self-implemented as well. Mobile application based on the iOS will be implemented for user interaction with the system. Integrating together these technologies, the system aims at recognizing object while flying autonomously.

During the flight process, drone will map the surroundings to obtain a static map of the environment. Path planning will be made according to the output map while simultaneously localize the drone and display its location on the map. During the flight, the user can view map construction at real-time, the video live streaming of drone viewing, and the current position of the drone in the constructed map. In case of emergency, the abort mission command is used to stop the operation of the drone immediately at any time of the process. After the process has been finished, the summary page is provided for the user to view the images of the objects found by the drone, along with its location on the map.

This proposed system would provide a prototype model for a smarter and more efficient drone system. The system can be extended and applied to various applications, for example, traverse into areas that is hazardous or impossible for human reach, such as navigation in mines, rescuing, finding, and localizing object or people in dangerous situations.

1.3 Objectives

The objective of this project is to develop a prototype system focusing on developing, applying, and integrating different technological aspects including object recognition, path planning, indoor navigation, and control via mobile application to the drone system to be the base for further drone system extension for future application. To indicate the successfulness of the project, the following goals are set to be satisfied:

- The drone is able to fly autonomously.
- The system is able to map the indoor environment.
- The system is able to recognize objects to search for.
- The system can search for an object and return the photo and location of the object as 2D map output when found.
- The application should allow user interaction with the system.
- The application has video live streaming functionality.
- The application should display map construction at real time.

1.4 Scope of work

The proposed system is designed to work under the scope as follows:

- Platform
  - Python programming language is used to implement drone flight control, path planning, SLAM algorithm, object recognition, and communications between the drone and the iOS mobile application.
  - The microcontroller attached to the drone must be running on Raspbian operating system.
  - Swift 4.0 programming language will be used to develop the iOS mobile application.
  - The mobile application will be available only on iOS platform with version 11 or newer.
- The drone will fly in one specific level/height.
- The available object to be recognized will be limited to categories within MS-COCO datasets.
- The environments should be confined.
- Location on 2D map and photo of localized object is returned to the application when object found.
- Map of the surrounded environment along with drone's position is sent to an application to be displayed at real time.
- Battery shortage will not be considered in this project.
- Assume WiFi connection is always perfect.
- There will be only static obstacle or permanent landmark.

1.5 Thesis structure

The rest of the thesis are organized as follows:

- Chapter 2 points out the necessary background knowledge to be understood before implementing this system. This chapter also includes literature reviews and related works.
- Chapter 3 describes the system and the proposed methodology in detail, by providing information, methods, and algorithms to develop the system.
- Chapter 4 illustrates the system architecture design, user interface and interaction with the mobile application.
- Chapter 5 shows and discusses the experiment results for drone flight control, device communication, object detection accuracies, and performance.
- Chapter 6 summarizes the proposed system and the thesis content, along with suggestions for further improvements and future work.

Chapter 2

Background knowledge and literature reviews

This chapter provides background knowledge essential for project understanding and review of the corresponding works. The background knowledge part covers the basics of drone control, localisation and mapping, object recognition, and mobile application in both hardware and software aspects. The literature review part provides the information obtained from reviewing related works.

2.1 Background knowledge

2.1.1 Drone dynamics

The drone will be able to fly autonomously is one of the main goals of the project. Basics of drone physics and drone control are essential for drone non-linear mechanism understanding and development.

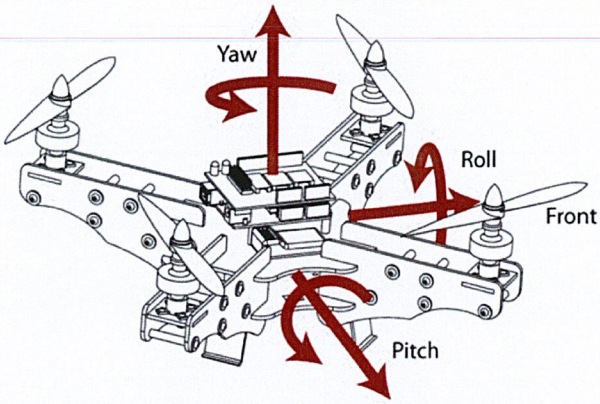


Figure 2-1. Quadcopter dynamics: roll, pitch, yaw.

Figure 2-1 shows the basic drone dynamics.[1] *Pitch* is the forward-backward motion, *Roll* is the tilt left-right motion, *Yaw* is the turn left-right motion, and lastly *Throttle* is the upward-downward motion.

2.1.2 Drone control

The drone or multi-rotor aircraft movement can be divided into vertical and horizontal direction. In the vertical direction, the drone can hover, ascend and descend depending on the thrust generated by the rotors. The thrust is determined by the voltage of each rotors which can be adjusted and controlled by the software.

[1] F. Zaman, *Nothing Beats a Clean Signal*, url=http://www.qualtre.com/2015/10/nothing-beats-a-clean-signal/ [Accessed: 18 Nov 2017]

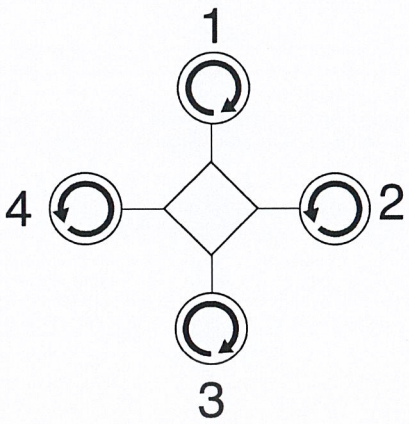


Figure 2-2. The drone's rotors direction.

Figure 2-2 describes the rotors direction. Rotors 1 and 3 spins in clockwise direction, and 2 and 4 spins anti-clockwise. In the horizontal direction, the drone will rotate at the non-equilibrium moment where rotors generate different thrust and all of the diagonal thrust of rotors should also be adjust to balance the drone, for example, left rotation can be achieved by increasing the thrust of rotor 1 and 3 and decreasing the thrust of rotor 2 and 4. In order to move forward in horizontal direction, the opposite rotor should generate more thrust to push the drone to ideal direction and all of the diagonal thrust should also be adjust to balance the drone. Figure 2-3 shows the thrust and direction of the rotor to turn left in yaw angle.

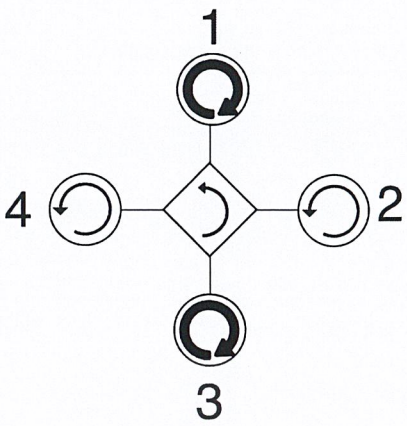


Figure 2-3. The left rotation.

2.1.3 ROS - Robot Operating System

ROS is a framework/middleware for collaborative robotics software development. It exists a collection of tools, libraries, and conventions to simplify the task of creating complex & robust robot behavior across a wide variety of robotic platforms

`Node` depicted in Figure 2-4 are processes that perform computation. Robot control system usually comprises many nodes. `Nodes` communicate with each other by passing `messages`. A `message` is simply a data structure, comprising typed fields.

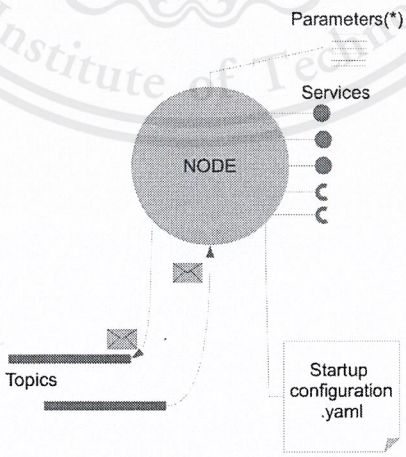


Figure 2-4. ROS node

`Messages` are routed via a transport system with publish / subscribe semantics shown in Figure 2-5. A node sends out a message by publishing it to a given `topic`. The `topic` is a name that is used to identify the content of the `message`. A `node` that is interested in a certain kind of data will subscribe to the appropriate `topic` asynchronously.

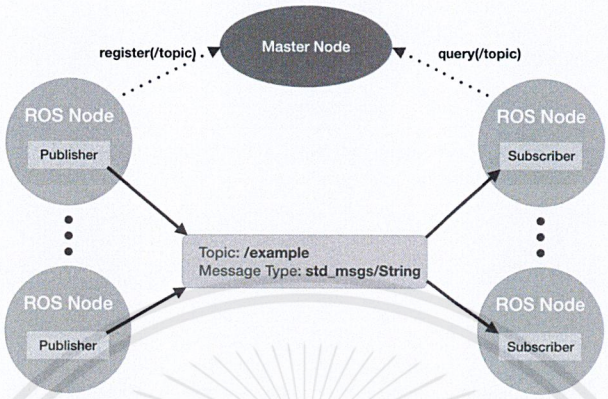


Figure 2-5. ROS Publisher/Subscriber model

2.1.3.1 ROS distributed architecture

In the system depicted in Figure 2-6, Raspberry Pi Model 3B is used as the onboard computer. Running all processes at the same time could result in high computational load, causing the microprocessor stop working and reboot. To be able to handle all the running tasks, the tasks could be separated and computed through distributed systems. This literature review explores the available robotic systems that utilizes the ROS distributed systems.

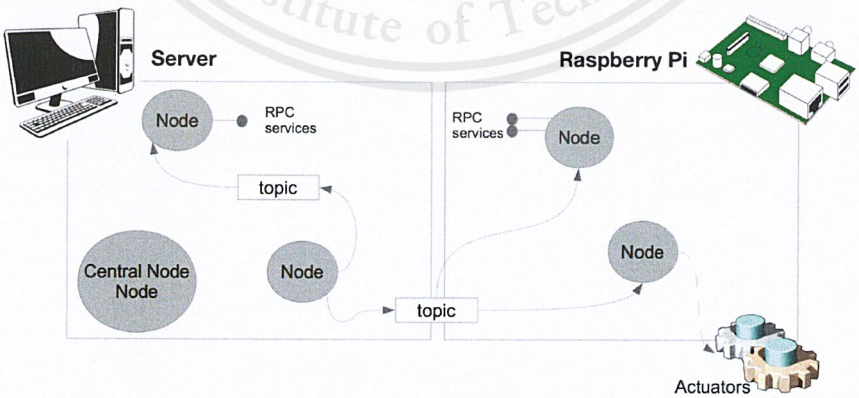


Figure 2-6. ROS distributed system architecture

2.1.4 Localization and mapping

Robot localization is the problem of estimating the robot's pose (state) associated with a map of the environment. Localization is a crucial component of successful robot navigation, along with environment recognition, learning and motion control. The process of building a map at the same time with computing the pose of the robot is called *SLAM*.

2.1.4.1 Simultaneous localization and mapping

SLAM can be applied with both 2D and 3D motions. It concerns with building a map of an unknown enclosed environment in real time by a mobile robot, while at the same time navigating the environment using the 2D/3D map. This is a solution to make the robot fully autonomous. SLAM is significantly useful in non-human reachable situations, for example search and rescue.

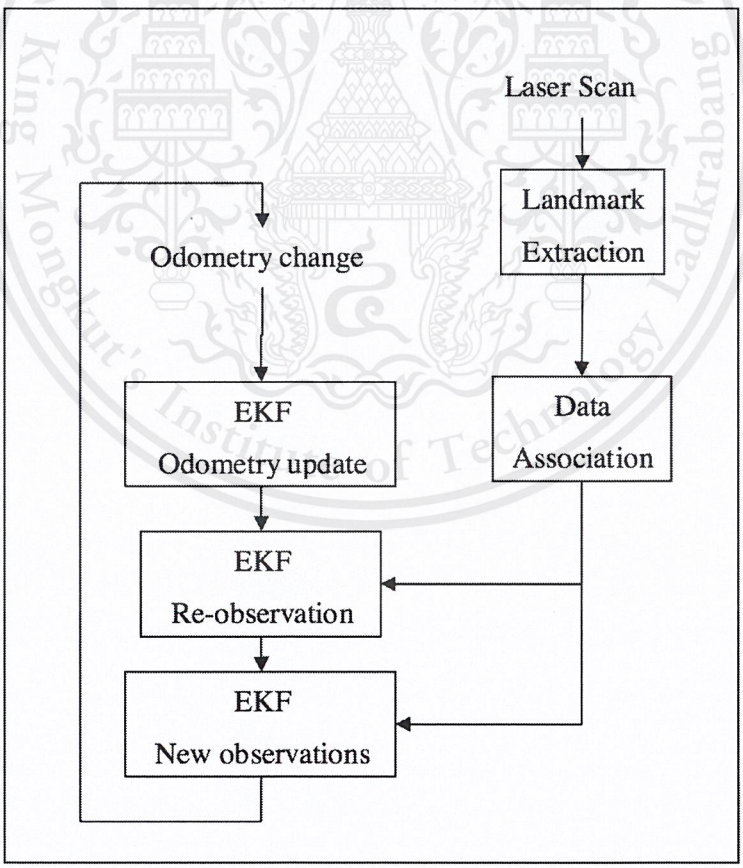


Figure 2-7. Overview of SLAM process. [29]

The goal of the SLAM algorithm is to use the environmental data to update the position of the robot, along with combining the robot's motions or odometry. Laser scanners are used to make the position of the robot more accurate according to the landmark features. The *Extended Kalman Filter* (EKF) is used to minimize the erroneous of the odometry, which helps to make SLAM algorithm more precise. Figure 2-7 shows the overview of the SLAM process. When the odometry or the motion of the drone changes, EKF will minimize the odometry's errors and update the odometry to be more precise. The algorithm then re-observes the environment using data from the laser scan, so by combining the odometry update and laser data, the localization of the drone and map is constructed.

2.1.4.2 Data

Data used in learning the map comes in two forms: odometry features and landmark features. The landmark features are extracted by the data from the laser scanner.

Laser data

Lidar uses laser light to densely sample the surface of objects or the earth, producing highly accurate x, y, z measurements (cloud point data). The data obtained from the laser are constructed into the 2D/3D map of cloud points. Laser scanner (light detection and range finder) device that is used in this project is the RPLidar A2 laser scanner (see more detail in Chapter 3). Most small robots are equipped with ultrasonic/infrared sensors, which have low accuracy and a limited number of readings. The range of these sensors is 2cm to 3m. If ultrasonic sensors are equipped in the front, left, and right of the robot, it is difficult to build a map. The combination of data and constructing the 3D model is difficult.

Lidar sensor is better than ultrasonic and infrared sensors, as one device covers 360 degrees spinning and supports SLAM algorithm. Figure 2-8 shows the representation of the map of point clouds from Lidar sensor.[2]

[2]R. Robyn, *What is a Point Cloud. What is LiDAR*,url=https://knowledge.safe.com/articles/257/ what-is-a-point-cloud-what-is-lidar.html [Accessed: 18 Nov 2017]

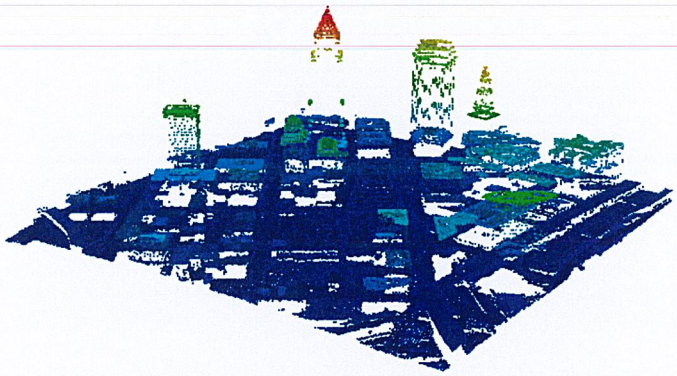


Figure 2-8. Point cloud data from Lidar sensor.

Odometry

Odometry is a method of approximating the position of the robot from the robot's motions, measured by acceleration and angle. This is used to approximate the initial position of the robot and to analyze the position from its movement. Odometry in robotics often refers to the estimation of the entire trajectory of a moving robot. So for every time instance t, there is a vector:

$$\left[x^t y^t z^t \alpha^t \beta^t \gamma^t\right], \tag{2.1}$$

which describes the complete pose of the robot at that instance. x,y,z are cartesian coordinates of the robot and α, β, γ are the euler angles (row, pitch, yaw). The odometry data will give the x, y, θ distance traveled by the drone at each timestep. Odometry data measurements are sensitive to noises, slippage, errors. This is because the data erroneousness comes from the only odometer for localization. SLAM solution uses x, y, θ observation data from odometer for an initial pose.

Landmarks

Landmarks are the point of the feature in the environment that is used by a robot to find its position, for example corners or line segments within the building. As shown in Figure 2-9, a simultaneous estimation of both robot and landmark locations is required. The true locations are never known or measured directly. Observations are made between actual robot position and landmark locations.

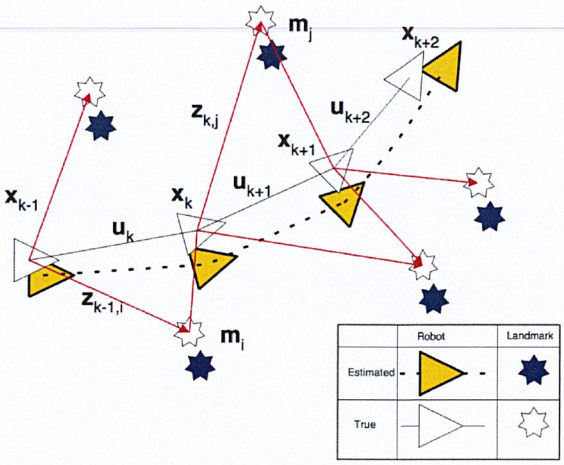


Figure 2-9. A simultaneous estimation of robot and landmark locations. [17]

2.1.4.3 Robot localization error handling

In common mobile robotics, precise localization is difficult as sensors and odometry data has accumulative errors. *Kalman filter* is usually used to minimize those erroneous.

Kalman Filter

Kalman filter is used with the robot's kinematic motion model in the propagation step to improve vehicle navigation by predicting the state and covariance of the robot in the next timestep [26]. The process of propagation is to use the kinematic model of the robot with odometric measurements, predicting the future state of the robot and the relative uncertainty in that prediction. The filtering creates an optimal estimation of the state of the drone and the error covariance data by combining its motion and the data observed from the environment. By the end, noises and accumulative errors are reduced. Equation (2.2) describes the brief idea of Kalman Filter:

Predict

$$\overline{\mu}_t = A_t \mu_{t-1} + B_t v_t$$
$$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Update (2.2)

$$K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$$
$$\mu_t = \overline{\mu}_t + K_t (z_t - C_t \overline{\mu}_t)$$
$$\Sigma_t = (I - K_t C_t) \overline{\Sigma}_t$$

where the belief $bel(x_t)$ at time t by the mean μ_t and covariance Σ_t are the representation of EKF. A new belief $bel(x_t)$ is predicted based on the previous belief $bel(x_{t-1})$, after the new measurement is received (odometry changed), along with the control data u_t. The predicted belief concerns with the state $\overline{\mu}_t$ at time t and covariance $\overline{\Sigma}_t$. The vector R_t denotes the noises from the odometry. The update step transforms the belief $(\overline{\mu}_t, \overline{\Sigma}_t)$ into expected belief (μ_t, Σ_t), by including the measurement z_t. The Kalman gain K_t specifies the degree of the odometry included for the new state estimate. The matrix A, B, C denotes the linear coefficients of the motions of the robot. Kalman Filter focuses on the difference between the actual measurement z_t and the expected measurement $C_t \overline{\mu}_t$ that is derived from the predicted state [30].

Kalman Filter requires a linear system, based on the assumption that the models of motion and measurement are effected by Gaussian noise. The linear combination of Gaussians is described resulting in another Gaussian. However, linear systems is insufficient to describe many real-life problems [30].

Extended Kalman Filter (EKF)

EKF is the core of SLAM algorithm, which keeps track of approximate of uncertainty in robots position and landmarks in the environment. The algorithm is used to update where the robot thinks it is according to the features or landmarks. In each iteration, it estimates the state(position) of the robot from odometry data and landmark observations from sensors.

The goals of EKF is to filter noises, process multiple measurements of a single variable, and predict the state of the system in the near future. The extended version of Kalman Filter is significant because of the non-linear nature of the drone's flight dynamics [24]. The algorithm attempts to approximate non-linear motion and sensor models to make them linear. This process is called *linearization*. The non-linear functions g and h concern the state transition and measurement probabilities:

$$\begin{aligned} x_t &= g(u_t, x_{t-1}), \\ z_t &= h(x_t) + \sigma_t, \end{aligned} \tag{2.3}$$

where x_t is the predicted state and z_t is the actual state. The two functions g and h replaces the matrix A, B, C of the regular Kalman Filter (2.2). A first-order linear approximation is calculated at the mean of the current belief (Gaussian), resulting in a linear function and results in linear Gaussian density shown in Equation (2.4) [30].

Predict

$$\begin{aligned} \overline{\mu}_t &= \boldsymbol{g(u_t, \mu_{t-1})} \\ \overline{\Sigma}_t &= \boldsymbol{G_t} \Sigma_{t-1} \boldsymbol{G_t^T} + R_t \end{aligned}$$

Update (2.4)

$$\begin{aligned} K_t &= \overline{\Sigma}_t \boldsymbol{H_t^T} (\boldsymbol{H_t} \overline{\Sigma}_t \boldsymbol{H_t^T} + Q_t)^{-1} \\ \mu_t &= \overline{\mu}_t + K_t (z_t - \boldsymbol{h(\overline{\mu}_t)}) \\ \Sigma_t &= (I - K_t \boldsymbol{H_t}) \overline{\Sigma}_t \end{aligned}$$

2.1.4.4 3D mapping

Real-time visualization is crucial for robot navigation and for human readability of current localization of the robot and the explored environment, including its trails. Laser data or point clouds are combined and plotted to represent the surrounding environment.

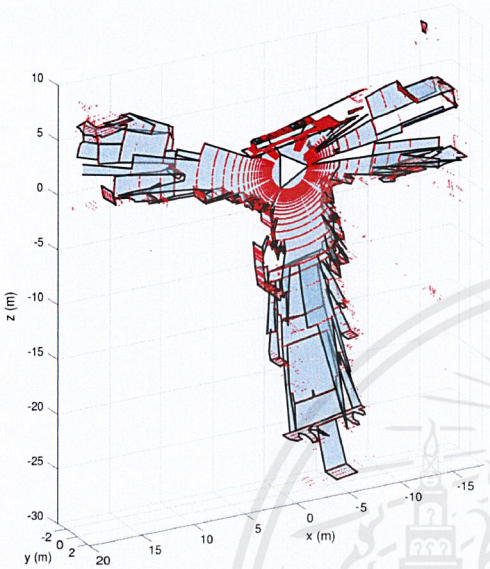


Figure 2-10. Local planar 3D map [1].

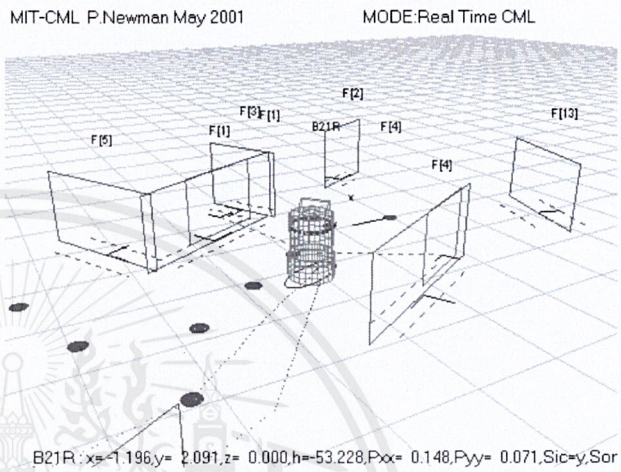


Figure 2-11. Realtime SLAM visualization by Newman, et al. [21].

As described in the Figure 2-10, the red dots represent points from the point cloud acquired by the LIDAR, while the white triangle denotes the robot's position and its facing direction. In order to localize the robot, SLAM algorithm simultaneously build the map and perform localization using odometry data and landmarks from the environment illustrated in Figure 2-11. The real-time process of SLAM enhances its performance and allows it to build a meaningful and precise map.

There are four major parts for converting laser data to 3D map models:

- Filtering off non-ground regions
- Segmentation and classification (Feature extraction)
- Building outline approximation
- 3D Modeling

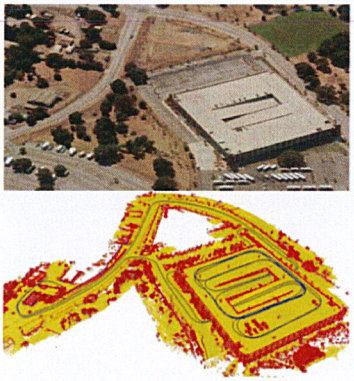


Figure 2-12. Raw point cloud data

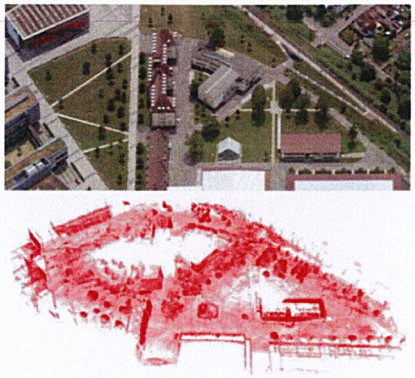


Figure 2-13. After filtering non-ground regions

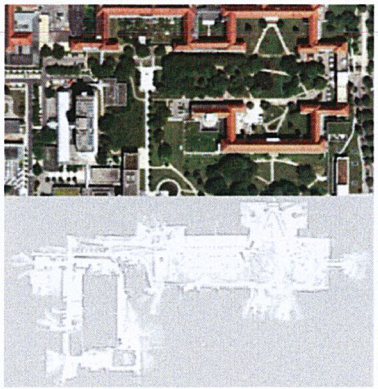


Figure 2-14. Outline of the building

Figure 2-12 shows the cloud point data from Lidar sensor including its depth in raw detail and Figure 2-13 shows all point cloud data after filtering off non-ground regions. Figure 2-14 is the outline of the buildings after the building line segments and corner features are extracted from the raw point cloud [15].

2.1.4.5 Feature extraction

The process of extracting the features of the corners and line segments of the room clarifies the overall environment structure, which improves the localization of robots. Typical features to be detected include concave corners, convex corners, and doors frames. Figure 2-15 illustrates the features that the robot detects:

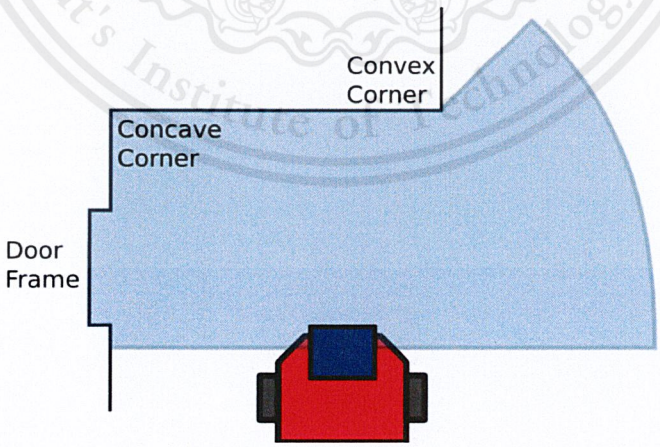


Figure 2-15. Landmark features of a room [26].

The blue area in Figure 2-15 denotes Lidar sensor range. Data from the sensor, also known as cloud point, is processed into an array of feature objects, which contains x,y,z position, covariance, and the boolean value to identify convex or concave corner. After receiving Lidar data, the next step is to detect all the lines illustrated in Figures 2-16 and 2-17:

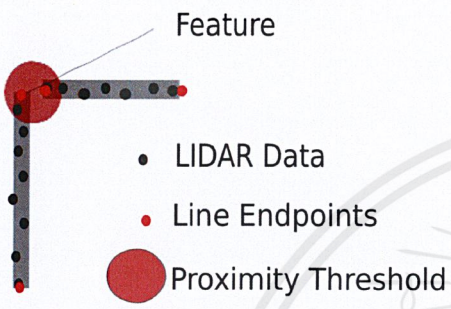


Figure 2-16. Landmark corner extraction [26].

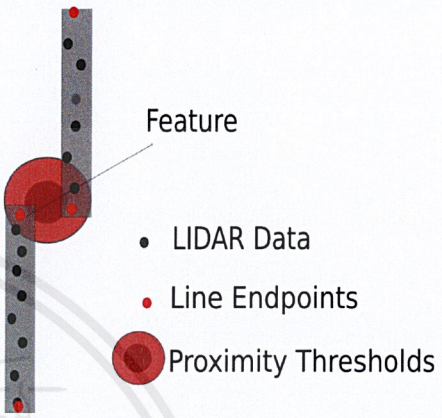


Figure 2-17. Landmark room line segment extraction [26].

The black dots in Figures 2-16 and 2-17 are the Lidar data that represents the line. The red dots denote the line endpoints, which is the sharp discontinuity of Lidar data or the end of the line. Figure 2-16 shows the detection of corners by looking for perpendicular lines. By observing the perpendicular lines, we can decide whether the corner is convex or concave. By using a boolean value, the type of corner feature can be identified and marked. Door or window features can be extracted by observing parallel lines that have endpoints close to one another, as shown in Figure 2-17. The double-sided proximity thresholds are used to reduce noisy feature extraction and to check the distance between both line endpoints in order to decide whether the door frame exists or not. A boolean flag is used to mark whether a feature is a door frame, convex corner, or concave corner.

If a parallel line is detected, the feature is considered a door frame. However, if perpendicular line is detected, further consideration is needed to define whether the feature is a convex or concave corner. The process is done by calculating the distances between the robot and the line endpoints. Figure 2-18 shows the robot detecting perpendicular line features. The left figure shows the identification of concave corner and on the right figure, the convex

corner.

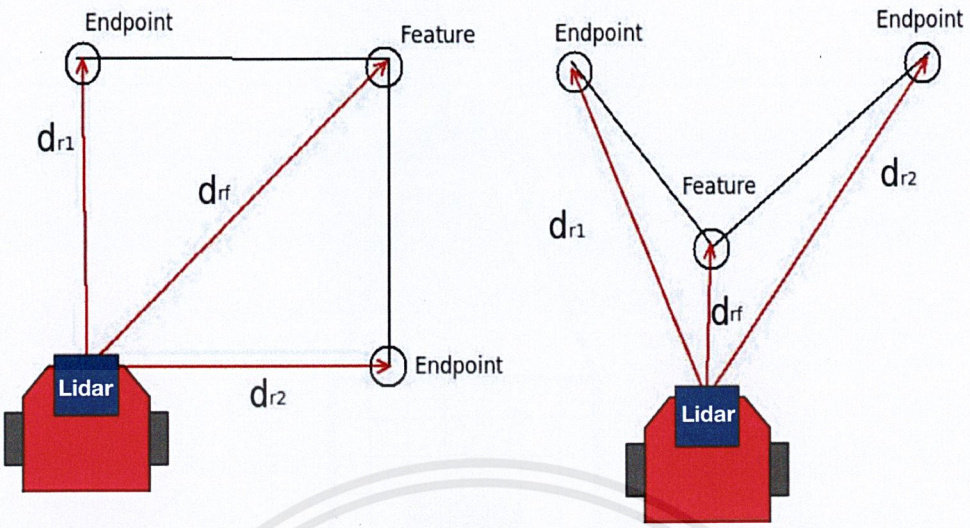


Figure 2-18. Landmark corner feature detection and identification [26].

2.1.4.6 Path planning

Navigation is a crucial component for autonomous vehicles. The UAV navigation in an unknown environment makes significant use of SLAM which implies the navigation with real-time sensor data which may include the discovery of newly established obstacles. The goal of system navigation focuses on finding collision-free path in real time while taking computational cost and energy consumption (travel length) under limited computational resource into account. There are several approaches for path planning where selection criteria varied by the environment handled and knowledge of the environment [9]. The algorithms can be classified as heuristic or complete based on coverage completeness of environment [16]. Path computation can also be perform either off-line where knowledge of the environment is partially known or on-line when known in advance.

Discrete occupancy grid map is the dominant representation in robotics. Every cell of the map is defined as a node of a graph [7]. Each node has three possible states which are free, occupied or unknown according to the sensor estimation. Computationally, a graph might be stored as adjacency list/matrix.

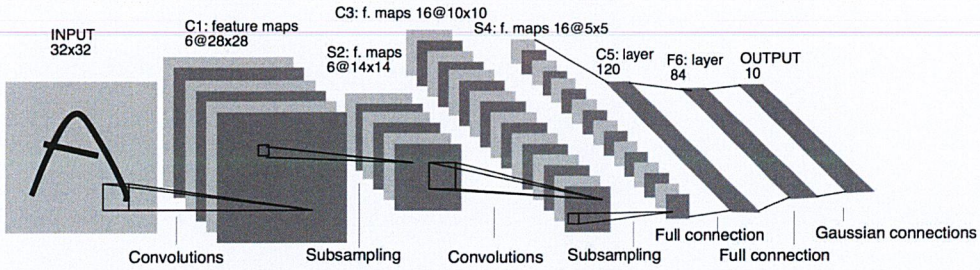


Figure 2-22. Architecture of LeNet-5 [?].

The LeNet5 architecture consists of several layers which interspersed depends on the type of data which can be divided into 3 main layers.

1. Convolution layer

   The convolution operator is used to extract spatial features from the image while preserving relative details by applying the filter (convolutional mask, window, or kernel) to the image matrix in spatial domain with matrix multiplication. Convolution operation[3] is illustrated in Figure 2-23.

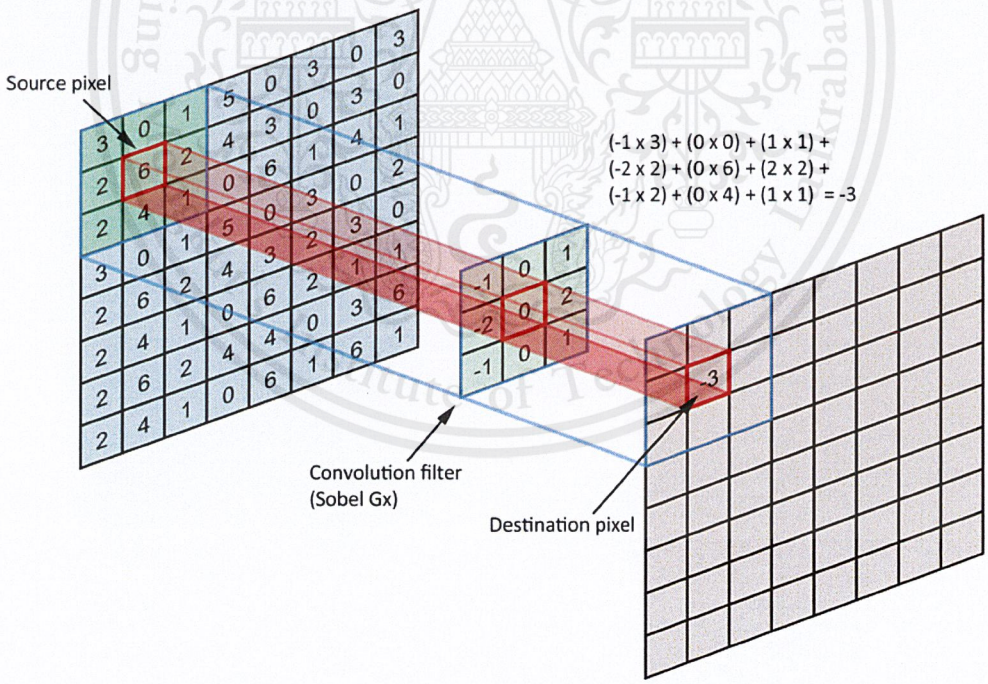


Figure 2-23. Single convolution layer.

[3] A. Durville, *Computer Vision with Convolution Networks*,url=https://github.com/OKStateACM/AI_Worksh Vision-with-Convolution-Networks [Accessed: 19 Nov 2017]

There are several types of filter which correspond to specific values or features for specific purpose. The output after convolution is called activation maps or feature maps with different weight vectors [?].

2. Pooling layer

   Feature maps can be used to map and identify the pattern without knowing exact location, therefore, downsampling operation can be applied [?]. There are several options for Pooling layer or sub-sampling layer and the *Maxpool* is the most popular. The operation reduces the resolution of feature map; for example, Maxpool replaces the local region with relative max value.

   Therefore, the computation cost is reduced as a result of spatial dimension reduction and overfitting is controlled as the position precision of features is reduced. The visualization of features in a fully trained model is illustrated in Figure 2-24.

3. Fully connected layer

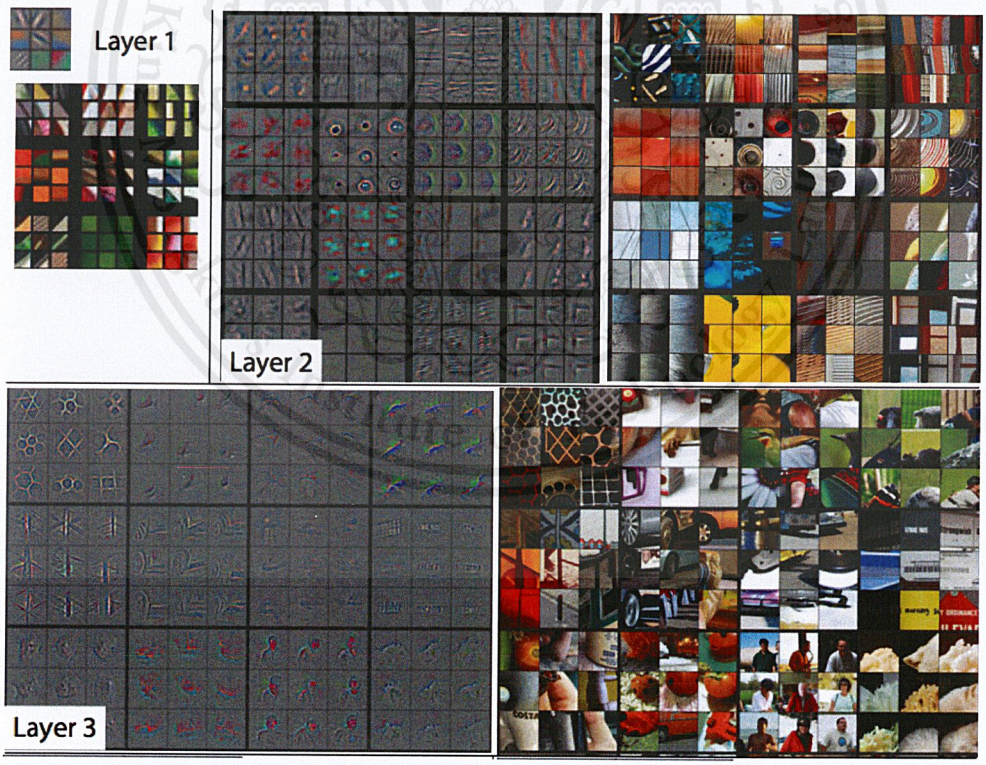


Figure 2-24. Visualization of features [27].

After detecting high level features, the fully connected layer acts as a classifier for image classes according to correlate features and weight of particular classes. Fully connected layer composes of input layer which has neurons connected to hidden layer that link to an output layer.[4] Hyperparameters control the output volume, *Stride* controls the step that filter convolves over the input, *Padding* pads the border input with specific value, and *Depth* is the number of filters/features. In the output layer, *Softmax* activation function is applied to normalize the class probabilities as an output by limiting the range of values of probabilities to be from 0 to 1

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, ..., K, \tag{2.5}$$

where z is an input vector of real number with K dimension.

2.1.5.3 Image datasets

Modern architecture comprised with massive database makes high performance classification. With large scale open source dataset designed for visual recognition use, the model can be trained and tested on powerful resource. The following datasets are prominent source of image data used by researchers.

- ImageNet

  An image database organized in large lexical hierarchy.[5] WordNet has approximately 14.2 million images in total with 21,841 images classified into synonym word sets. ImageNet is also used as evaluation dataset in Large Scale Visual Recognition Challenge (LSVRC).

- MS COCO

  A large image dataset developed by Microsoft for object detection, segmentation and caption with several features.[6] COCO has around 1.5 million object instances. The

[4]Convolutional Neural Networks (CNNs / ConvNets),url=http://cs231n.github.io/convolutional-networks [Accessed: 20 Nov 2017]

[5]ImageNet,url=http://image-net.org [Accessed: 8 Dec 2017]

[6]COCO dataset,url=http://cocodataset.org [Accessed: 8 Dec 2017]

dataset is used by researchers and image classification competition.

- Pascal VOC

  The dataset is well-know for the VOC challenge since 2005.[7] The dataset is derived from "flickr" website and available for training and testing. Pascal VOC has up to 11,530 images in total.

2.1.5.4 Pretrained models

Since several machine learning frameworks has already provided pretrained CNN models which allows transferring of functionalities and direct extension toward real-world application, object classification element will be developed on top of pretrained model while benifiting from reduced training effort.

- Alexnet

  A network composed of five convolutional layers and three fully-connected layers. Alexnet is trained using ImageNet dataset and won 2012 ILSVRC [6]. The network has capability to classify up to 1000 categories. The architecture of AlexNet is illustrated in Figure 2-25.

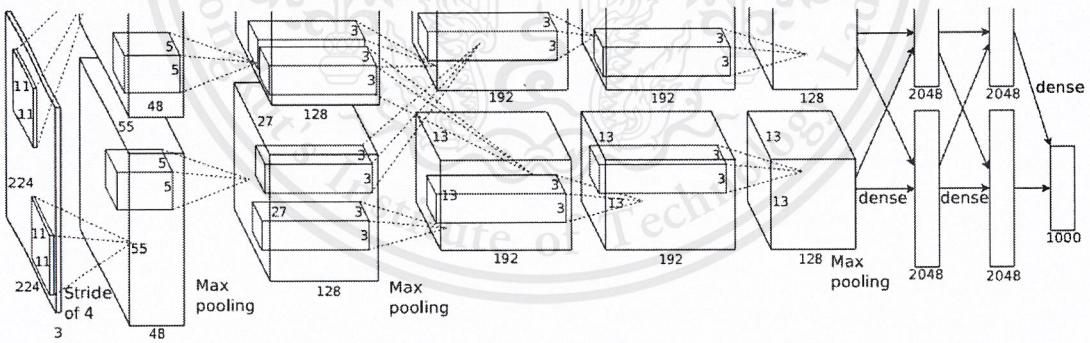


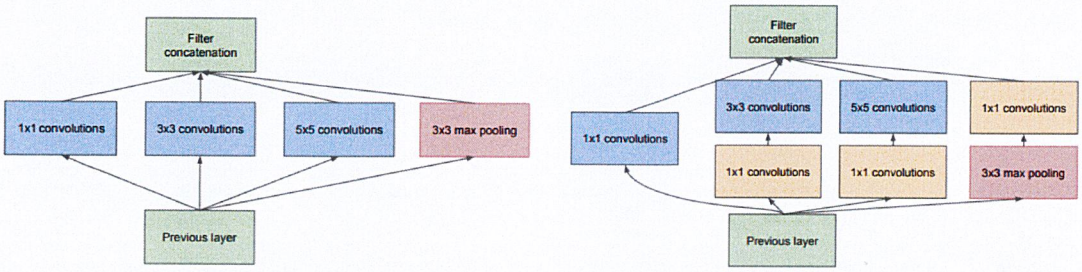
Figure 2-25. Architecture of AlexNet [?].

- GoogLeNet

  A deep convolutional neural network composes of 9 inception modules and 22 layers deep network. GoogLeNet was trained using ImageNet dataset and won 2014 Large

[7]Pascal Visual Object Classes,url=http://host.robots.ox.ac.uk/pascal/VOC [Accessed: 8 Dec 2017]

Visual Recognition Challenge (ILSVRC) [5]. The network uses 12 fewer parameters than AlexNet and gains a better accuracy. The architecture of Inception module and GoogLeNet[8] are illustrated in Figures 2-26 and 2-27, respectively.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2-26. Inception module [5].

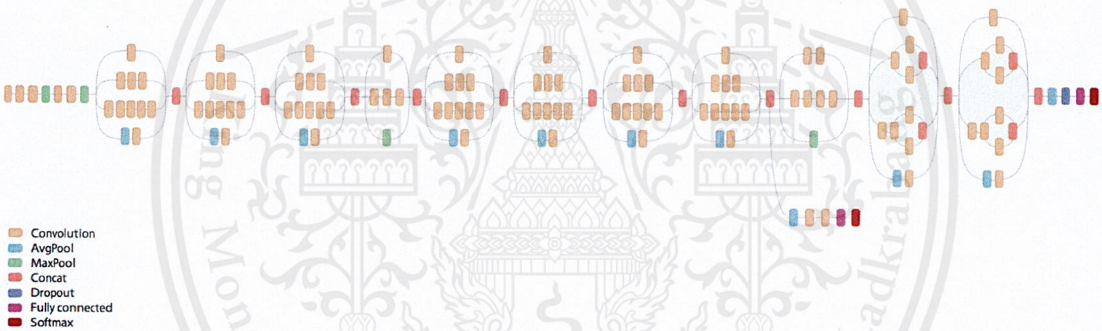


Figure 2-27. Schematic diagram of GoogLeNet.

- SqueezeNet

  A small CNN architecture with Alexnet-level accuracy on ImageNet with 50x fewer parameters [11]. The model reduces parameters by using model compression. This is done by resizing the filter, decreasing a number of input channels using squeeze layer, and downsample the architecture by increasing the stride in convolution or pooling layers. The organization of module and architecture of SqueezeNet is illustrated in Figures 2-28 and 2-29, respectively.

[8] *Train your own image classifier with Inception in TensorFlow*,url=https://research.googleblog.com/2016/03/tr your-own-image-classifier-with.html [Accessed: 22 Nov 2017]

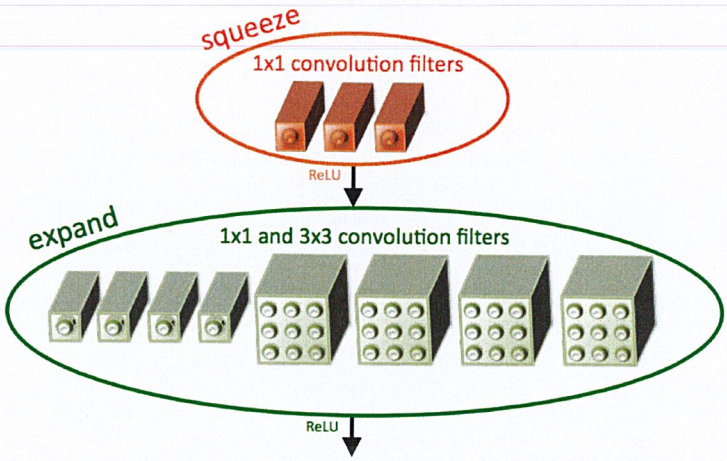


Figure 2-28. Organization of convolution filters in the Fire module [11].

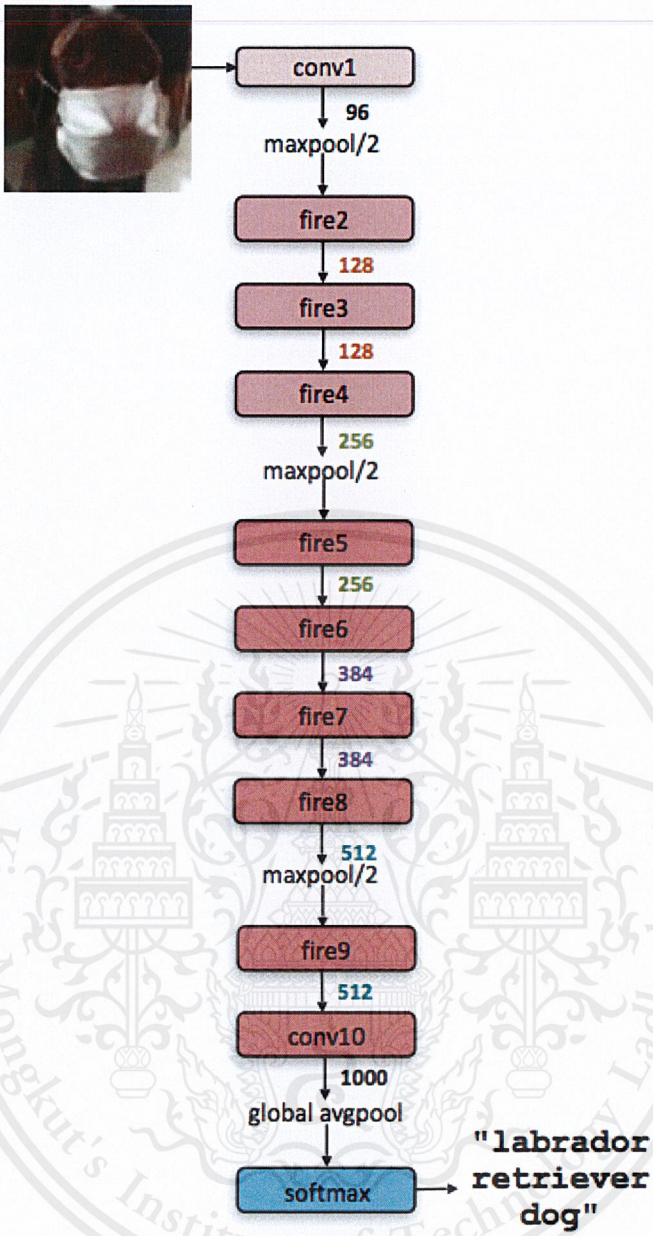


Figure 2-29. SqueezeNet architecture [11].

- ResNet

  ResNet is a 152-layer deep CNN architecture developed by Microsoft Research. ResNet or Residual Network won ILSVRC 2015 with accuracy surpassing human capabilities in some aspects. The network has significant increase in accuracy besides increasing in depth. The network addresses the training error problem caused by increasing in

layers by introducing deep residual learning network [25].

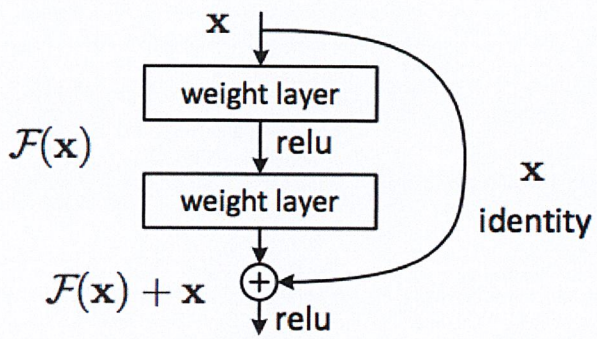


Figure 2-30. Residual learning block [25].

From Figure 2-30, the residual learning block is described as a block with input x and output is reformulated from traditional $F(x)$ to $H(x) = F(x) + x$ as the later form is easier in term of optimization due to carried information of x.

- R-CNN

  R-CNN or Region Based CNN provides the architecture combining capabilities of object localization with bounding box and object classification with label. The network performs greedy search in the initial process to extract around 2000 region proposals then use "warp" technique to resize the region into a specific size which can be feed into CNN models and classify region with support vector machines afterward [32].

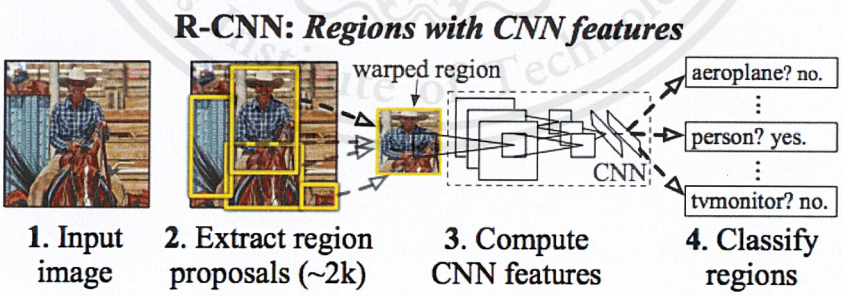


Figure 2-31. R-CNN object detection process [32].

- Faster R-CNN

Faster R-CNN is an improvement of early R-CNN models. The architecture introduces Region Proposal Network (RPN) to eliminate region proposal computation bottleneck by sharing convolutional features with detection network.

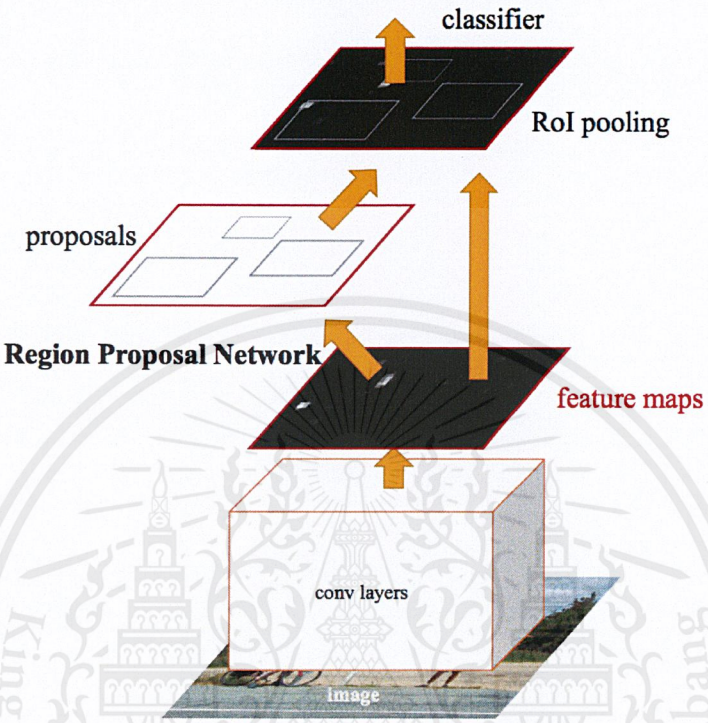


Figure 2-32. Faster R-CNN object detection process [35].

Figure 2-32 describes the detection process of Faster R-CNN. Regional proposal were produced from feature maps by applying RPN after the last CNN layer.

2.1.6 Mobile application

A mobile application is used for communication between the user and the system. It was implemented on an iOS platform. In this section the protocols that are used for the communication between the iOS application and the system along with the supported services will be described.

2.1.6.1 Communication protocol

The drone and mobile application has to communicate through the server in order to accomplish its functionalities. Protocol used will be explained in this section.

SOCKET

A socket is a logical endpoint for two-way communication that links two programs on the network.[9] On the server side, it runs on a specific computer with a socket bounded to a port number that waits for the request from a client for making connection. On the client side, it knows the hostname of the computer with the server and the port number to which the server listens. The client identifies itself to the server and tries to meet up with the server on the server's machine and port to make connection request, as shown in Figure 2-33.[10]

Figure 2-33. Client request for connection.

While the server accepts connection, new socket will be bounded to the server's local port. The server will have its remote endpoint set to the address and port of the client. If connection is accepted, a socket will be created. The client can use it to communicate with the server by either reading or writing from the socket, as shown in Figure 2-34.[11]

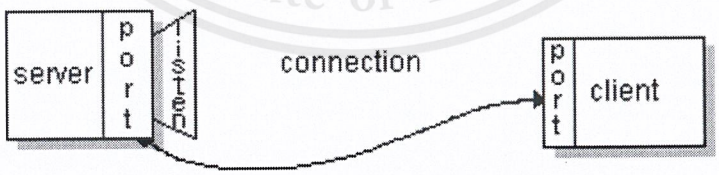


Figure 2-34. Connection being made.

[9] Oracle, *Lesson: All About Sockets @ONLINE*,url=https://docs.oracle.com/javase/tutorial/networking/sockets/index.html [Accessed 3 Oct 2017]

[10] Oracle, *Lesson: All About Sockets @ONLINE*, url=https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

[11] Oracle, *Lesson: All About Sockets @ONLINE*, url=https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. Also multiple connections between host and server is possible.

RTSP

The Real Time Streaming Protocol (RTSP) is a network control protocol focusing on handling the continuous media session such as audio or video across IP networks. It establishes and controls a single or several time-synchronized streams of continuous media. It is designed especially for use in entertainment and communications systems and is based on TCP for reliable delivery. It is used by the client application to communicate to the server for such information as type of application client used, the media file that is requested, the mechanism use to delivery the file such as unicast or multicast, UDP or TCP, and other important control information commands such as SETUP and PLAY. The actual multimedia content is not delivered through the RTSP connection, but it can be interleaved if required. RTSP can be viewed as a remote control for the streaming protocols.

2.1.6.2 APNs

Apple Push Notification service (APNs) is the key for the remote notification. It is a robust, highly secure, and efficient service use in propagating information to any iOS devices.

Once the application is launched on the device, the system automatically establishes an accredited, encrypted, and persistent IP connection between the application itself and the APNs which allows the application to setup and enable it to receive notifications.

Configuration in the developer account and the use of Apple-supplied cryptographic certificates are also required. A provider is a server that has been configured to work with APNs. Figure 2-35 shows remote notification's path for delivery.

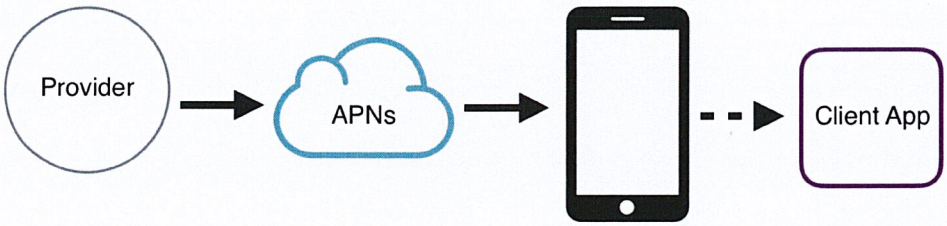


Figure 2-35. APNs path of delivery for a remote notification.

With push notification setup completely on both providers and application, providers can send notification requests to APNs where it conveys corresponding notification to each device on target. Once receive notification, the system delivers the payload to the appropriate application on the device along with handling interactions with the users.

If the notification arrives when the device is powered on while application is not running, the system can still display the notification. However, if device is powered off, APNs will take care of this by holding on to the notification and tries again later.

2.1.6.3 Video4Linux

Video4Linux, also known as V4L, is a collection of device drivers and an API that support real-time video capturing based on Linux systems. It can work with many USB webcams, TV, and several other related devices that standardize its output. This allows programmers to easily add video to the applications.

Some of the main features of V4L include:

- Video capture/output and tuning
- Video capture and output overlay
- Memory-to-Memory devices
- Raw and Sliced VBI capture and output

- Device topology discovery/control

V4L2 is the second version of the V4L that fixed some bugs and started appearing in the 2.5.x kernels. Its drivers include a compatibility mode for Video4Linux1 applications.

2.2 Literature review

This section describes the related works of this project and guidelines of the previous work to be improved in this project.

2.2.1 Indoor unmanned aerial vehicle (UAV) navigation

Wang and Cui has proposed the UAV indoor navigation system in [12] that is mainly based on the vision optical flow and Laser FastSLAM. The sensors embedded on the UAV include inertial measurement unit (IMU), camera, and laser scanner. The FastSLAM scan matching algorithm is used to estimate the planar position of the UAV. FastSLAM is a SLAM algorithm that integrates particle filters and extended Kalman filters [28]. In this algorithm, corners and line segments are used as map features to perform SLAM. Along with this, there is also a barometer to capture the UAV's height. Although the UAV path is well estimated with minimum hardware resources, the computational complexity is to be reduced.

Another robust and efficient indoor navigation system for UAV was proposed by Wang et al. [13]. Two scanning laser range finders along with IMU are used on-board. The value from the laser scanner mounted horizontally to give out the plane position of the UAV. The same technique is used to calculate the height of the UAV from the ground, but the value used is from laser scanner mounted vertically. From this paper, the authors used split-and-merge segmentation algorithm [14] with values from Lidar measurements to get the line segments. This navigation technique uses a planar localization algorithm with assumptions that translation and rotational variations are estimated efficiently. The algorithm includes feature extraction, rotation tracking, corner feature association, and position tracking. The limit to this navigation solution is that it only works with partially known environment.

Bachrach and He [2] had proposed another method for UAV localization in their paper entitled, *Autonomous flight in unknown indoor environments.* This method is only using

one Lidar sensor along with IMU for the UAV localization and map construction. The scan matching algorithm is used for planar localization. It is an algorithm that compare the current and previous scan observation and align them to make the position gained from odometry more accurate. However, this same method that works well with navigation when adopted for environmental mapping resulted in limited field of view. It would not work well with UAV applications like real time object classification and mapping that require comprehensive geometric information. Using the stated method, the 3D map could only be generated when UAV moves vertically, as it is difficult to horizontally get the geographic information of objects like cylinders and spheres. Also distinguishing objects on the floor from the actual floor can be difficult since only a small field of Lidar data is available to estimate the altitude.

2.2.2 Autonomous object tracking UAV

Engel has purposed an autonomous camera-based navigation of a quadcopter system operating in an unknown and non-GPS environment which complies with this project goals [18]. The system uses cameras attached to a Parrot AR.Drone for navigation, and computation is performed by the server connected to the drone via WLAN. The drone can localize itself using SLAM algorithm along with proportional-integral-differential (PID) drone control and data fusion and prediction method of EKF [18]. The proposed system uses software provided by Parrot AR.Drone and requires a custom change by using provided interface to access the drone's functionalities which has some limitation on hardware and software access or internal settings. The drone communication protocol is ad-hoc WLAN which is used to send sensors values, states of the drone, and video streaming to ground server which is an iOS application. The system uses monocular SLAM based on single camera for navigation with computer vision and maximum likelihood approach.

Another potential object recognition UAV system is real-time object detection for UAV based on cloud-based CNN proposed by Lee et al. [20]. The system uses a Parrot AR.Drone as platform and the position estimator is based on PID controller running on laptop connected over wireless connection. Furthermore, the R-CNN approach is used to detect multiple objects in almost at real-time which adopts the Amazon Web Services (AWS) computing

cloud to reduce computational and power loads. This introduces some communication latency among drone, laptop and the cloud. The object detection model is trained on the ImageNet dataset and navigation markers are provided. The system works successfully in controlled environment.

Han et al. proposed a system for object detection and tracking for smart drones on embedded system called Deep Drone for automatic detection and tracking [?]. The system targets on enabling embedded system on the drone the capabilities of running fast and robust deep learning under hardware constraints. The system uses CNN, histogram of oriented gradients (HOG) feature and kernel correlation filter (KCF) as non-expensive computation model. The system focuses on face detection of single person in an early stage. The R-CNN and Yolo detector are chosen to be evaluated as the project targets the fast and accurate recognition. The Yolo detector is faster but has limitation on the target distance so the Faster R-CNN algorithm is used.

Chapter 3

Methodology

This chapter describes the details of the method used to implement and develop the proposed system. The chapter is divided into three sections, including the drone, the server, and the mobile application. The drone system is separated into two parts: the hardware and the processing unit, which contains the localization and mapping, path planning, and object recognition. The full system architecture is illustrated in Figure 3-1. The server and drone support the ROS Distributed Architecture.

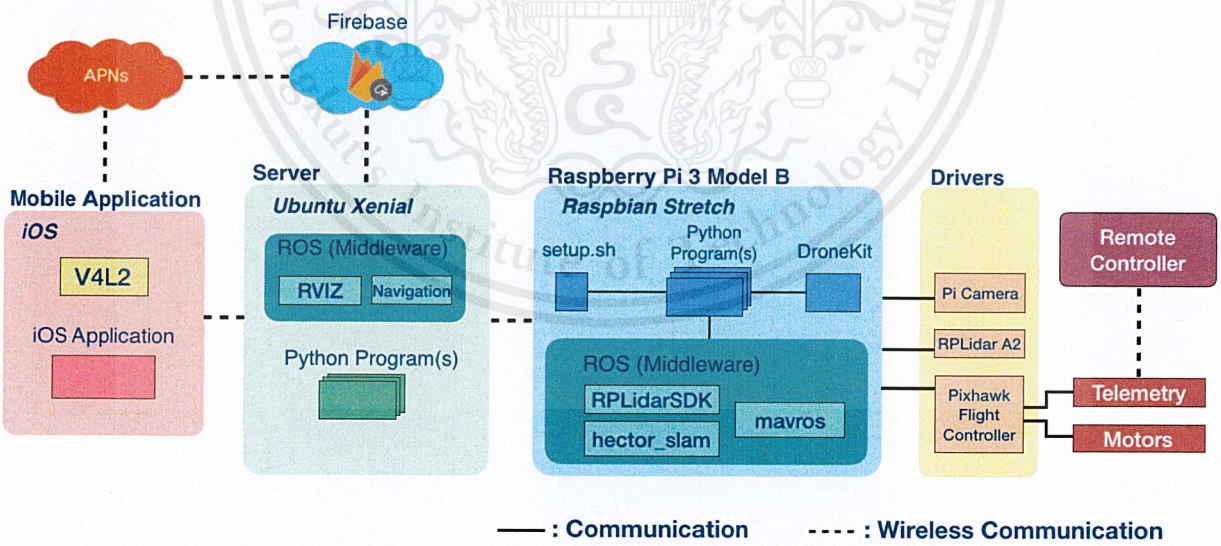


Figure 3-1. Full system architecture

In general, the work flow of the system starts by letting the user specify the product

key of the drone using the mobile application, the server then maps the mobile application's IP address with the specified drone's IP address. Next, the user enters the object name and sends it to the server through Socket. The server then signals the drone for system initialization. The camera captures the environment and publishes the video feeds into the RTSP server. The server retrieves the video stream to classify and recognize the object. When an object is found, the server will send a notification to the mobile application through Firebase, which will push it to the APNS. After the notification, the image of the object found and map snapshot is sent to the application. If the object is found but the environment has not been completely explored, the image and location will be saved in the database and the drone will proceed the search. Once the flight is completed, the drone hovers, indicating the end of flight. After the end of the flight, the mobile application will take the user to the summary page, which shows all the objects found and its associated location map. During the flight, the user can view the video live stream and the map exploration. The server also performs the path planning for the drone.

3.1 Drone

The drone is the subsystem that is developed to physically search for objects in indoor areas. The implementation consists of hardware setup, connectivity, and processing unit of the drone which include localization and mapping, publishing video streams from the camera module.

3.1.1 Hardware setup and connectivity

This section shows the hardware setup and connectivity of the drone.

3.1.1.1 Hardware components

In this section, devices and sensors used in this project will be described. This includes its usage in the project and specifications.

Raspberry Pi

Raspberry Pi 3 model B (Figure 3-2)[1] is used as the main drone controller in this project. It is a small single-board computer (microcontroller) equipped with a 1.2GHz 64-bit quad-core ARM Cortex-A53 CPU, with integrated 802.11n wireless LAN and Bluetooth 4.1. In this project, it will perform object recognition, localization and mapping, path planning of the drone, and communicate with the mobile device through socket and RTSP channels.

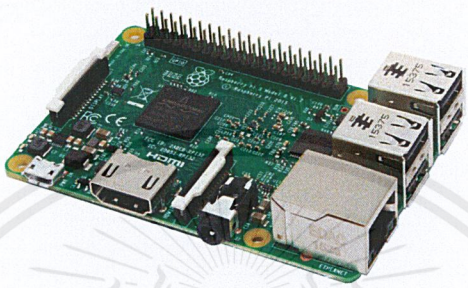


Figure 3-2. Raspberry Pi 3 model B.

Raspberry Pi camera module

Raspberry Pi camera board featuring V1 (Figure 3-3)[2] is used to stream videos and photos for object recognition and send video feeds to the mobile application. The camera is a 5-megapixel resolution camera, with focus lens on-board. The camera is capable of 2592 × 1944 pixel static images. It supports 1080p30, 720p60, and 640×480p60/90 video. Lastly, it utilizes the Camera Serial Interface (CSI) which is capable of handling extremely high data rates.

[1] 9Arduino, *Raspberry Pi 3 Model B*, url=https://www.9arduino.com/product/241/raspberry-pi-3-model-b [Accessed: 11 Dec 2017]

[2] ArduinoThai, *Raspberry Pi Camera Module [Accessed: 11 Dec 2017]*, url=https://www.arduinothai.com/product/245/raspberry-pi-camera-module

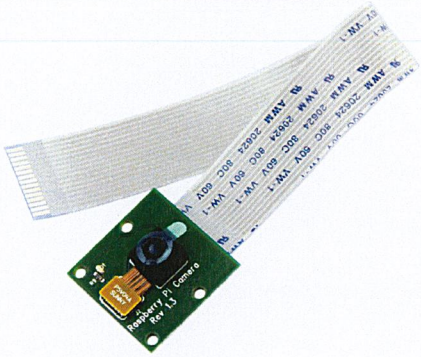


Figure 3-3. Raspberry Pi camera module V1.

RPLIDAR

Lidar sensor (Figure 3-4)[3] is an ideal sensor for robot localization and mapping. It emits modulated infrared laser signal and reflect onto the object. The reflected object returns a signal, which will be sampled by vision acquisition system of the Lidar sensor device. RPLIDAR is a low-cost LIDAR sensor suitable for indoor robotic SLAM application. It provides 360 degree scan field, 5.5Hz/10Hz rotating frequency with guaranteed 8 meter ranger distance. It performs high-speed distance measurement with more than 4000 samples per second. Users can customize the scanning frequency from 2Hz to 10Hz freely by controlling the speed of the scanning motor through PWM signal. The main usage of Lidar sensor is for SLAM, obstacle detection and avoidance, environmental scanning, 3D rebuilding, multi-point touching, and man-machine interaction.

Figure 3-4. RPLIDAR A2 laser scanner.

[3]RobotShop, *RPLIDAR A2 360 Laser Scanner*, url=https://www.robotshop.com /en/rplidar-a2-360-laser-scanner.html [Accessed: 11 Dec 2017]

Drone frame

The drone frame (Figure 3-5)[4] is a quadcopterX frame that supports four motors. The drone frame used is the F450 quadcopter kit. The frame is designed so that it is extremely light and versatile. This way implementers can add or remove certain technical elements.

Figure 3-5. F450 QuadcopterX frame

Drone ECS and motors

Drones are equipped with four 40A Electronic Speed Controller (ESC)[5], four motors[6], and four propellers[7] (Figure 3-6). ESC converts the PWM signal from the flight controller or radio receiver and drives the EMAX MT2213-920KV brushless motor by providing the appropriate level of electrical power. The motors are the devices that spin. By combining motors with propellers, it makes drone fly. The faster the motor spins, the faster it goes. However, the four motors must work together to be able to lift up, go forward, backward, downwards, turn left, and turn right according to the throttle, roll, pitch, and yaw.

---

[4]RadioC, *Q450 QUADCOPTER FRAME*, url=https://www.radioc.co.uk/Q450-quadcopter-frame-p/1000.htm [Accessed: 11 Dec 2017]

[5]Acme Hobbies, *HOBBYWING X-ROTOR 40A OPTO ESC*, url=http://www.acmehobbies.com.au/shop/moto speed-controllers/hobbywing-x-rotor-40a-opto-esc-for-multirotor-asia-pacific-area-version.html [Accessed: 11 Dec 2017]

[6]Hobby Wireless, *DJI 2212 920 KV MOTOR*, url=http://hobbywireless.com/motors-brushless-c-146_152/dji-2212-920-kv-motor-set-of-2-cwccw-p-4461.html [Accessed: 11 Dec 2017]

[7]AliExpress, *F450 Propeller Black*, url=https://www.aliexpress.com/popular/f450-propeller-black.html [Accessed: 11 Dec 2017]

Figure 3-6. This figure includes the propellers, four EMAX MT2213-920KV brushless motors, and four 40A ESCs without BECs.

Pixhawk

Pixhawk (Figure 3-7)[8] is a high-performance autopilot-on-module suitable for fixed wing, multi rotors, helicopters, cars, boats and any other robotic platforms that can move. It is targeted towards high-end research, amateur, and industrial needs. In this project, Pixhawk will be used as the flight controller and is connected to the Raspberry Pi (drone controller) through serial communication.

Figure 3-7. Pixhawk auto-pilot module as flight controller.

[8]PX4 AutoPilot, *Autopilot Hardware*, url=https://pixhawk.org [Accessed: 11 Dec 2017]

FS-i6S remote controller

The FS-i6S remote controller (RC) (Figure 3-8)[9] is used for controlling the drone through radio frequencies. The drone will be equipped with 915MHz telemetry and RC receiver to be able to receive radio signals from the RC and send the data to Pixhawk flight controller. The purpose of RC in this project is for safety, in case if problems occur during the flight. If a problem occurs, there is always a back up to control the drone manually to prevent the drone from crashing.

Figure 3-8. FS-i6S remote controller with RC receiver.

[9] AliExpress, *FS-i6S Remote Controller*, url=https://es.aliexpress.com/item/Flysky-FS-i6S-2-4G-10CH-AFHDS-Transmitter-With-FS-iA6B-Receiver-Remote-Control-For-Racer/32792488454.html [Accessed: 11 Dec 2017]

915MHz telemetry

The 915 MHz telemetry (Figure 3-9)[10] receives wireless data transfer, such as radio signals from the RC, and sends data to Pixhawk flight controller for more proceedings and calibration of data.

Figure 3-9. Telemetry used for wireless communication.

Lithium Polymer Battery

Lithium Polymer (LiPo) battery is used to power the motors of the drone. The battery is connected to the power distribution board that distributes the voltages to the four ESCs which will send power to the motors. The battery life depends on the its usage and load of the drone.

Universal Battery Eliminating Circuit 5V 3A

Universal Battery Eliminating Circuit is also known as UBEC. The UBEC 5V 3A is connected to the battery to convert voltage to supply 5V for the power of other devices and sensors, such as power Pixhawk flight controller.

[10] 3DZ, *915 MHz (American) Telemetry Radio Set*, url=https://store.3dr.com/products/915-mhz-telemetry-radio [Accessed: 11 Dec 2017]

3.1.1.2 Hardware setup

Components of the drone are assembled as shown in Figure 3-10. Raspberry Pi is used as the main controller of the drone which will perform the processing units and send commands to control the drone flight. The commands from Raspberry Pi to the flight controller will be described in section 3.1.1.5. Pixhawk device is the flight controller in this project, converts the commands to PWM signal to the ESCs in order to control the motors. LiPo battery is used to power the motors and Pixhawk controller. The battery is attached to the power distribution board to distribute voltage to 4 motors. BEC is also connected to the battery to power the Pixhawk. Furthermore, telemetry is attached to the drone to be able to communicate wirelessly with the ground station for setup and control. RC receiver is also attached to the drone to be able to receive radio signal commands from RC. In addition, attached on top of the drone is be RPLidar A2 which will scan the environment. Raw data obtained from Lidar will be operated and processed in the localization and mapping unit. In between Raspberry Pi and Lidar, there will be a mounting board lifted up from the drone to hold the Lidar sensor up high and to handle the vibration so the Lidar sensor could operate effectively.

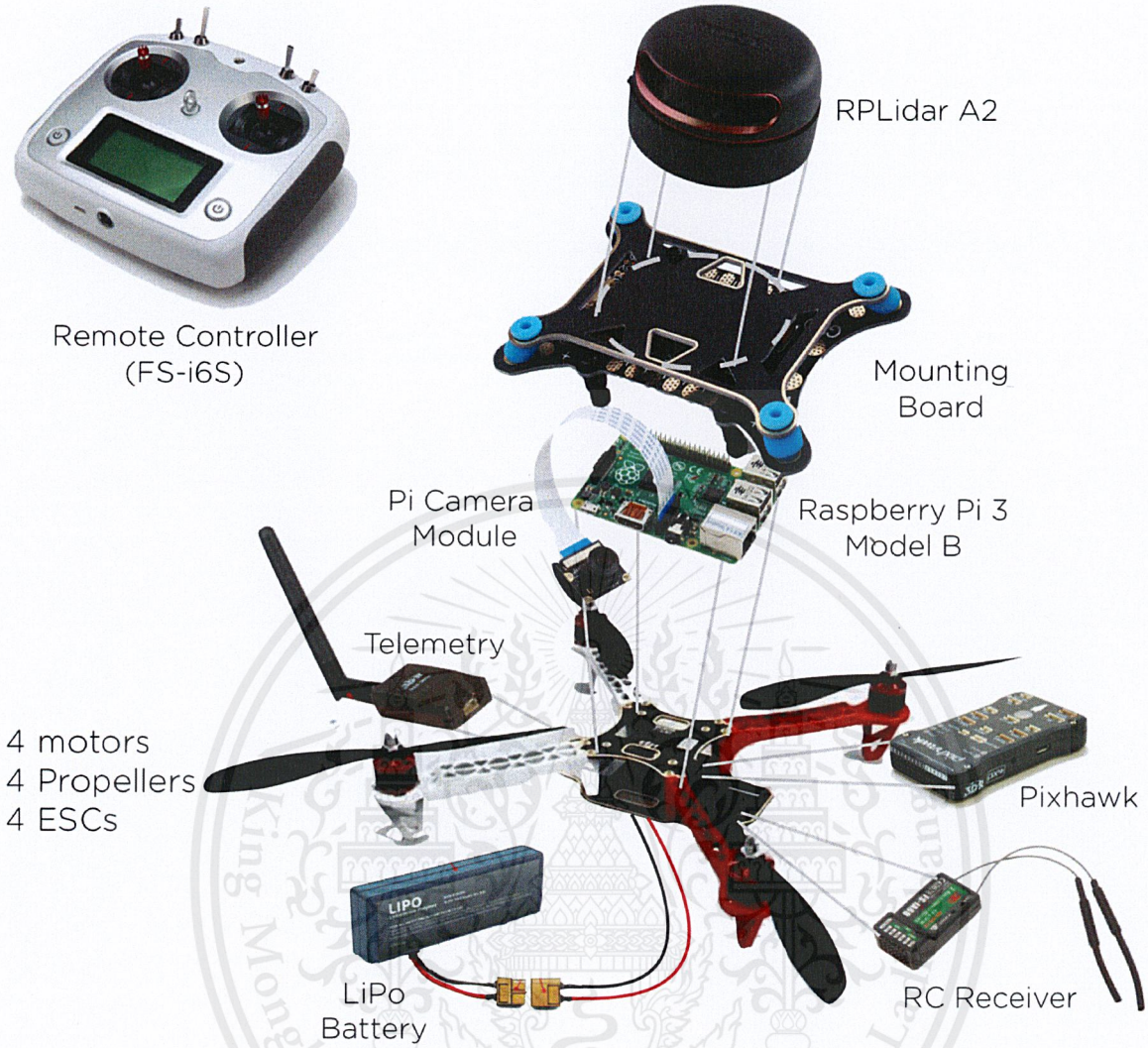


Figure 3-10. Drone Hardware Setup.

3.1.1.3 Hardware connection

The important hardware connection is shown in Figure 3-11.

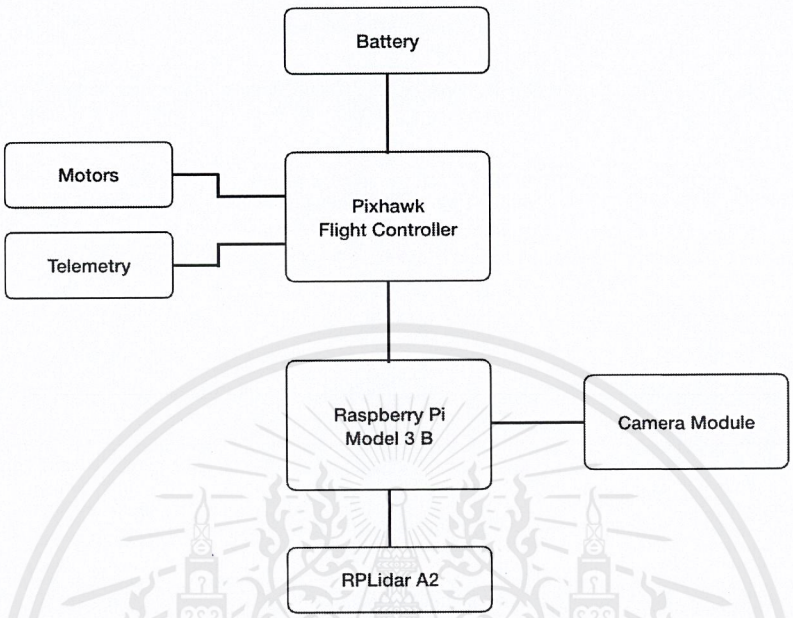


Figure 3-11. Drone connection.

Raspberry Pi is the main controller of the drone. It is equipped with Pi camera module, Movidius Neural Compute Stick, USB connection to RPLIDAR A2 laser scanner, and serial communication with Pixhawk flight controller. The Pixhawk or the flight controller has a buzzer and safety button for drone signaling. The connection between Pixhawk and the Raspberry Pi is through serial communication from Telemetry 2 of Pixhawk to GPIO pins of the Raspberry Pi. The Pixhawk is powered and connected to the battery with BEC 5V in the middle to convert battery voltage to 5V. Connecting with the battery is the power distribution board that powers the four motors attached to the board. Each individual motor and ESC is also connected to the Pixhawk output pins, which will send out drone movement control as PWM signals to the motors. The telemetry is connected to the Pixhawk through channel telemetry 1 for wireless communications. In addition, the RC Receiver is attached to the Pixhawk to receive radio signals from the RC, that controls the drone movements manually. More details of the hardware connection are shown in the schematic diagram in Figure 3-12.

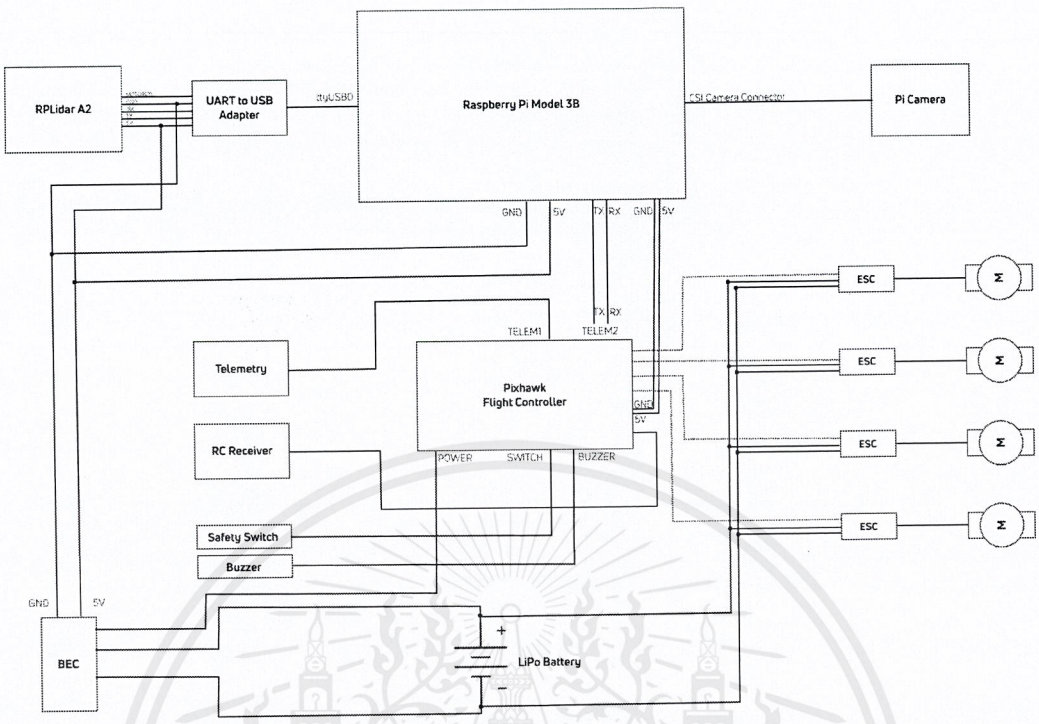


Figure 3-12. Schematic diagram of hardware connections

3.1.1.4 Drone setup

After all the components are assembled together, the setup process is needed.

Figure 3-13 shows the software used for setup, calibration, monitoring, and controlling of the drone.[11] This software is normally deployed onto the ground station device. In this project, it will be deployed on the Raspberry Pi. During setup, another telemetry is needed to connect to the ground station for wireless setup of the flight controller. Firstly, when the software is connected to the flight controller, COMPORT and BAUDRATE have to be selected. The baud rate used in this project is 57,600 bits per second. The setup includes updating latest firmware, selecting the type of vehicle, acceleration calibration, compass calibration, radio calibration, channel setup, PWM value setup, ESC calibration, and flight mode setup.

[11] ArduPilot, *Communicating with Raspberry Pi via MAVLink*, url=http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html

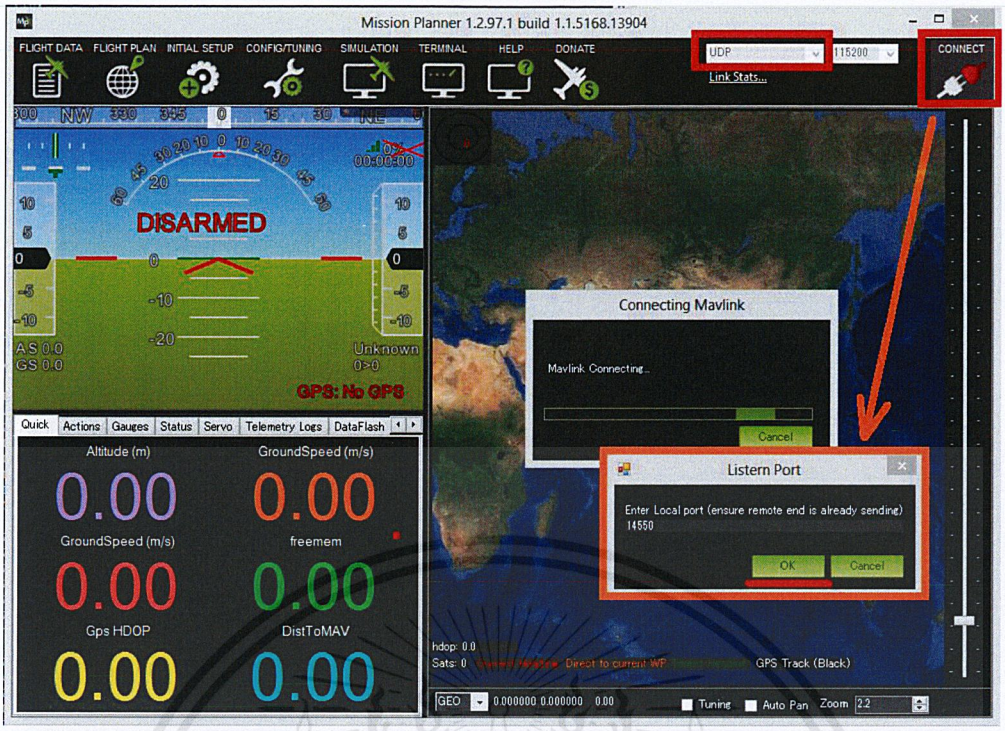


Figure 3-13. Mission Planner software for Pixhawk and drone setup, control, monitor, calibration.

3.1.1.5 Drone control

In the first phase of the project, the RC is used to control the drone manually. The flight mode suitable for this project is altitude hold, which always stabilizes the level of flight and the drone. This makes the control of drone convenient. The control commands depend on the throttle, roll, pitch, yaw value of motions from the RC.

In the second phase of development, the drone uses SLAM algorithm with Lidar sensor to aid on autonomous flight. So, the RC in the second phase is used for back up to handle and control the drone manually and to prevent emergency drone crash. The commands will be sent out from the Raspberry Pi to the Pixhawk through MavLink protocol described in Section 3.1.1.6. The control will be depending on path from the path planning process. In controlling the drone it is limited to only five degrees of freedom âĂŞ the roll is kept constant.

3.1.1.6 Communication between Raspberry Pi and Pixhawk

The protocol used to communicate between Raspberry Pi that run Raspbian and Pixhawk flight controller over serial communication is the MavLink protocol.[12] DroneKit API is being used to provide programmatic access to a connected vehicle's modules in order to retrieve parameter information and manipulate vehicle movement and operations through python program. In the ROS system, `mavros` package is used to bind the protocol with ROS. Pixhawk's TELEM2 port is connected to the Raspberry Pi's Ground, TX, and RX pins. Raspberry Pi can be powered by connecting the red V+ cable to the +5V pin or from USB.

More information of specific commands and setup used in the project can be found in Appendix A

3.1.1.7 Localization and mapping

Localization is crucial for the robot navigation process (hard real-time). The process used for building a map at the same time as computing the position of the robot is *SLAM*. The algorithm process is suitable for unknown environment. It allows the drone to localize itself and the algorithm also remembers the environment information in previously occupied areas. This algorithm is computed by the Raspberry Pi attached on the drone, as this prevent drone crash and must compute the map data from Lidar sensor at real-time.

The localization and mapping is implemented in ROS using `hector_slam` [23], an open-source SLAM framework that could be extended and used for various robot localization. The `hector_slam` uses the `hector_ mapping` node for learning a map of the environment and simultaneously estimates the platform's 2D pose at laser scanner frame rate. The frame names and options for `hector_mapping` have to be set correctly. The prerequisite for the implementation is to understand the basic understanding of Robot Operating System (ROS) parameters and launch files to be able to setup the localization and mapping process.

The `hector_mapping` is a SLAM approach that can be used without odometry as well as

[12] ArduPilot, *Communicating with Raspberry Pi via MAVLink*, url=http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html

on platforms that exhibit roll/pitch motion (of the sensor, the platform, or both). It utilizes Lidar laser scanner to be used instead of robot's odometry. `hector_geotiff` provides a node that can be used to save occupancy grid map, robot trajectory, and object of interest data to RoboCup Rescue compliant GeoTiff images. `hector_trajectory_server` keeps track of `tf` trajectories extracted from `tf` data and makes this data accessible via a service and topic.

The mapping during the drone navigation is constructed from data obtained from Lidar laser scanner.

3.1.2 Server

The server is the subsystem that handles most of the processing units of the system and handles the communication between the mobile application that is mapped with the specified drone. The implementation consists of visualizing drone's position and mapping, path planning, and object recognition from video stream from the camera module. The server is running on Ubuntu 16.04 (Xenial) which supports ROS system. Both the drone and server have their own local ROS system, and are connected through the ROS distributed system using TCP/IP protocol. The distributed system provides a convenient channel for passing data between two systems efficiently through `topics`. When an object is found, the server will fire a notification through Firebase, then sends the pictures of the object and the map snapshot.

3.1.2.1 Map visualization

After the `hector_mapping` module of `hector_slam` (computed on the Raspberry Pi) computes the drone's location from the laser data, it publishes the map data through `map` topic. The server will subscribe and listen to that topic. When there is map message passed through the `map` topic, the server can retrieve the map data and display it on `rviz` an ROS visualization tool shown in Figure 3-14.

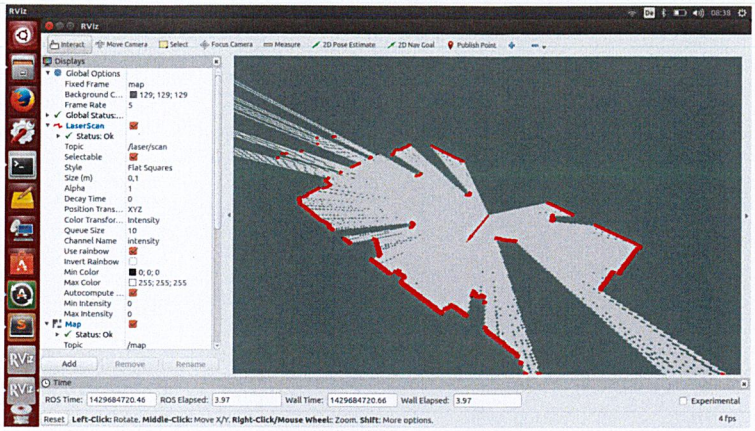


Figure 3-14. Rviz example.

During the flight, the map is continuously exported into `.bmp` file and sent from the server to the mobile application through socket for the application to display the map construction.

3.1.2.2 Path planning

For path planning development on ROS, there are interfaces available for custom planning algorithms implementation and optimization. The implementation of algorithm can be registered as plugins in order to use them in the `move_base` node within `navigation` package.[13] The planning module will be adopted from *navfn* library. As the path planning algorithms are calculated at real time, dealing with dynamic Environments is possible by using `navigation` package with layered architecture (local and global planner) in which obstacle detection is perform on-line by appropriate modules. The global planner uses the static map as input to plan a path. However, the local planner has a dynamic, local costmap build with current sensor inputs [7]. With this dynamic costmap, the local planner can avoid unexpected obstacles.

[13]Setup and Configuration of the Navigation Stack on a Robot,url=http://wiki.ros.org/navigation/Tutorials/RobotSetu
[Accessed: 8 Dec 2017]

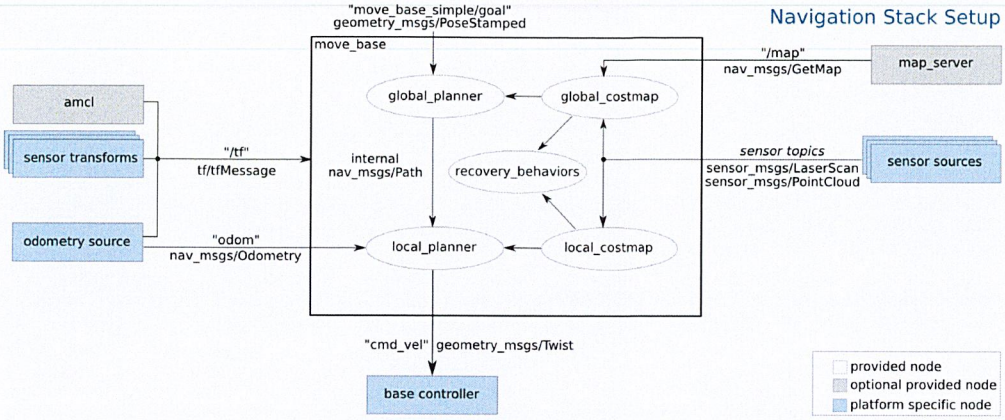


Figure 3-15. High-level view for navigation node [7].

3.1.2.3 Object classification

Object recognition is crucial for the object search task using camera. Apart from controlling and mapping, the server is also used as a processing unit to achieve task such as image recognition. Several approaches have been explored and compared relatively in performance and computation efficiency aspect.

Popular image recognition framework has been taken onto consideration and tested on Raspberry Pi with NCS. The result shows that the more optimized framework and model should be adopted to achieve a better frame rate under limited computational resource. Some models on Caffe framework such as SqueezeNet and BVLC reference model yield approximately 0.06 frame per second (fps) benchmark on 224 $\times$ 224 resolution. Inception V3 and GoogLeNet on TensorFlow yields around 0.06 and 0.3 fps, respectively. AlexNet from DeepBelief SDK yield approximately 0.3 fps. Therefore, the computation is transferred to the server to reduce computation loads and improve classification rate and accuracy.

The target platform for object recognition is a server running Ubuntu 16.04 LTS. For the machine learning framework, TensorFlow with pretrained object classification model is considered. With TensorFlow, high performance pretrained models such as Faster R-CNN ResNet and Faster R-CNN inception are offered. The Faster-RCNN ResNet 101 model is used to classify objects.

3.1.2.4 Workflow

The server is the center that links between the drone and the application allowing them to communicate, as shown in Figure 3-16 The data will pass through the server while the drone and mobile application connect to the server via Internet.

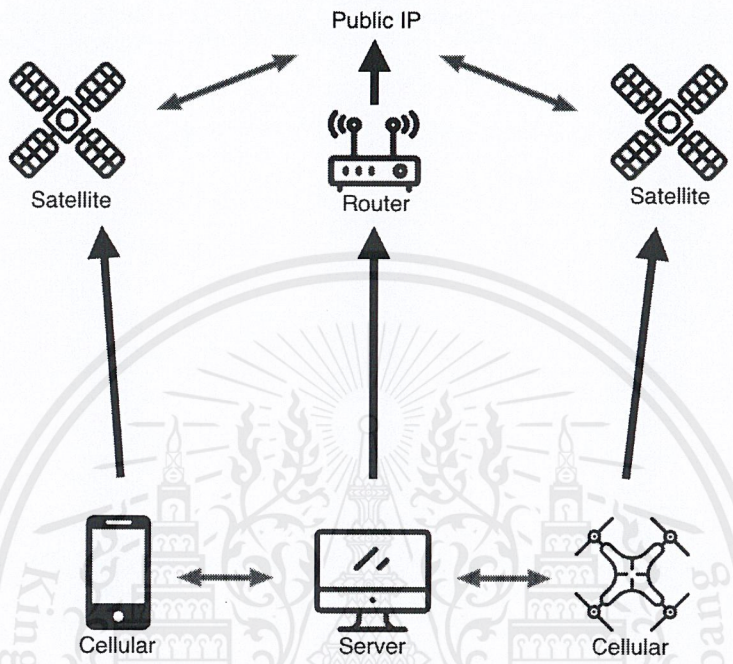


Figure 3-16. Connection between server, drone, and mobile application.

The work flow will be as follows:

1. The server connects to the Internet; assign fixed IP to the server.
2. The drone connects to the server by its fixed IP.
3. Once the application is turned on, application connects to the server.
4. Command to search for an object is sent from the application to the drone.
5. During flight, map construction data is sent from the drone to the server to construct the map and do path planning, and the server sends the map to the application. For real-time streaming data is being put on stream from which the application retrieves.

6. The server handles object recognition. Once the determined object is found, the server notifies the application and a pop-up notification will be shown to the user. The photo and location of the found object will be sent to the application.

7. In case of lost connection or close application during flight:

   - Data would be stored on Firebase.
   - Once the connection has been reestablished, the application would retrieve the stacked data of found objects from Firebase, and continue receiving map data and stream as usual.

8. At the end of the mission, the server will pass on the message to notify the application that the mission has ended.

3.1.2.5 Data format

Data sent between server and application is a string in JSON format. After receiving the data, it would be converted to JSON type for the convenience of data usage. For example "sender":"application", "object to search":"bottle"

3.1.2.6 Communication handling

Data are sent from both the drone and application to the server, so handling of data is a very important task. Socket protocol is used to send data between the application and server. In order to differentiate the data, we use different channel to receive the incoming data and tell the server what to do with each incoming data. For example, there are two different channels that handle incoming command to search for the object for the application and a channel to receiving incoming photo of object found from the drone.

The communication between the drone and the server uses the multi-master distributed network which is the nodes that listen to other masters and synchronizes with other masters. The `rostopics` are connected so the server can receive the drone's streams of data.

3.2 Mobile application

The mobile application is another subsystem that is developed for user's interaction with the drone. The mobile application will receive a command to search for target object from the user. This command will later on be processed accordingly with the server and drone to accomplished its object search task.

3.2.1 Communication with system

In order to accomplish its functionalities, the mobile application will have to communicate with the drone by sending and receiving message in the form of text, photo, and video through server. Methods used in achieving this are as follows:

3.2.1.1 Transferring text

Text is transferred between the application, server, Raspberry Pi through socket-io. The text will be sent through socket in the form of text string.

3.2.1.2 Transferring photo

Photo to be transferred are encoded into base64 where it can be sent through socket to its target. The message will be sent as a string of base64. After the target side got the message, it decodes base64 string back to the image type.

3.2.1.3 Live video

On the application, the UIImageView object will be used in representing the video image at real-time. The RTSP protocol will be used in transferring video from Raspberry Pi to the application, as shown in Figure 3-17. Once the next frame from video stream is received by the application, it will be decoded as UIImage type and the UIImageView object will update its image representation to the current image. The details are as follows:

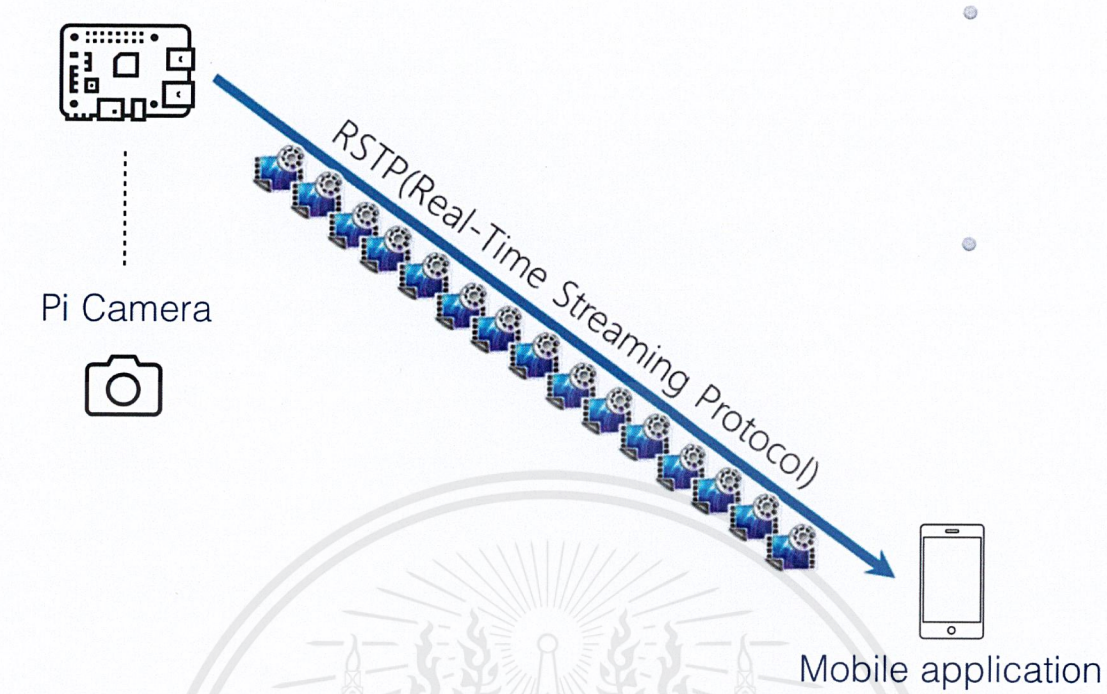


Figure 3-17. Video streaming using RTSP protocol

- Initialize with movie at moviePath (Output dimensions are set to source dimensions).
  1. Register all formats and codecs.
  2. Set the RTSP options.
  3. Retrieve stream information.
  4. Find first video stream.
  5. Get a pointer to the codec context for the video stream.
  6. Find the decoder for the video stream.
  7. Open codec.
  8. Allocate video frame.
- Read the next frame from the video stream. Returns false if no frame is read (video over).

3.2.2 Background notification

When the application is closed or is not connected to the Internet, once an object is found the notification will be made as background notification. Coming along with the notification are data of object found. For notification message will be sent to Firebase, then to Apple Server, and invoke notification on mobile application as shown in Figure 3-18.

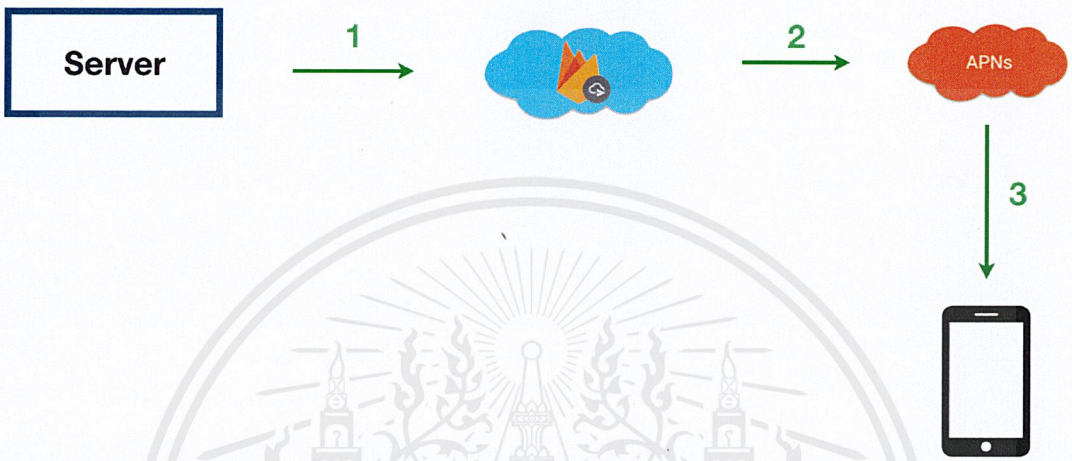


Figure 3-18. Background notification process

3.2.3 Real-time map construction

Figure 3-19. Data retrieval for map construction

As shown in Figure 3-19, raw data of the surrounded environment collected from RPLIDAR A2 and drone odometry data are sent to Raspberry Pi and to the server. The server will compute these data and give output in the form of a map. The map will be exported

and extracted to array of pixels. The array will be continuously sent to the application to construct the map at real-time. Once received the array of pixels, the application updates the UIViewController accordingly with the given array. It loops through the array pixel by pixel and updates each pixel on the UIViewController as how it is presented by the pixel scale in the array.

3.2.3.1 Updating pixels

The coloring of pixel will be displayed on the mobile application to show user the area that has been covered at real time. Following are steps of how to change color of specified pixel:

1. Extract the pixel buffer of the image
2. Loop through pixels
3. Change pixels color according to command sent from server
4. Create a new image from the buffer

3.2.4 Data management and distribution

For efficient data distribution over the application at real time, delegate pattern has put into used. We have created a class called, "DataManagement" which will take care of all data being passed from the server. DataManagement will be build as Singleton object to make sure all the view controller will share same attribute of DataManagement. Other view controller of the application during the flight (including: live-streaming page, real-time map construction page, show summary of object found during flight) will use data gathered by DataManagement. DataManagementDelegate protocol is implemented for each view controller.

Chapter 4

System analysis and design

This chapter presents an analysis and design of a system which provide a broader understanding of details and specifications of the system. This chapter contains system information ranging from the requirement analysis, user interface, and diagrams to class design.

4.1 System analysis

4.1.1 Requirements

The requirements of the system are defined and categorized according to FURPS+ model. The FURPS+ model describes one subcategory of functional requirement and five subcategories of non-functional requirements which are functionality, usability, reliability, performance, supportability and constraints respectively. Functionality defines features, capabilities and securities requirement of the system. Usability describes the human factor and documentation. Reliability is the system stability. Performance concerns with the speed and resource factors. Supportability determines the adaptability, maintainability and configurability of the system. The constraints or pseudo requirements defines implementation, interface and physical factors or limitations. The requirements of the system are illustrated in Table 4.1.

TABLE 4.1
SYSTEM REQUIREMENTS.

Requirement	FURPS+ Category
The system must be able to check and approve user input of object to search for.	Functionality
The system must be able to receive and display video streaming from the drone at real time.	Functionality
The system must be able to construct map at real time.	Functionality
The system must be able to give notification once object is found.	Functionality
The system must be able to keep record of location and photo of objects that had been found.	Functionality
The system must be able to display the location and photo of all objects found.	Functionality
The system must be able to give abort command to drone.	Functionality
The system must be able to classify types of object.	Functionality
The system must be able to navigate autonomously.	Functionality
The system must be able to perform path planning.	Functionality
The system must be able to localize itself.	Functionality
The system must be able to detect obstacles.	Functionality
The first time user enter the application, the introduction and instruction to the application will be provided.	Usability
Avoid icons that are too meaningless and abstract in user interface.	Usability
Auto connect to the drone once open an application.	Usability
The system must provide a camera stream within at least 10 frame rate (frames per second)	Performance

Continued on next page

Table 4.1 – continued from previous page	
Requirement	**FURPS+ Category**
The system is available on iOS platform.	Supportability
The must be maintainable and extensible.	Supportability
The system will be licensed by King Mongkut's Institute of Technology Ladkrabang.	Licensing(+)
The system requires RPLIDAR SDK.	Design constraints (+)
The system requires flight controller firmware.	Design constraints (+)
The system requires machine learning framework.	Design constraints (+)
The system requires database.	Design constraints (+)
The application requires Python version 2.7.X.	Design constraints (+)
The system requires OpenCV2.	Design constraints (+)
The system requires ROS Kinetic as robotic framework.	Design constraints (+)
The system requires ROS Navigation package.	Design constraints (+)
The system requires ROS SLAM package.	Design constraints (+)
The server requires Ubuntu 16.04 operating system	Design constraints (+)
The drone controller requires debian-based operating system	Design constraints (+)
The system requires server.	Physical requirement (+)
The system requires WiFi connection.	Physical requirement (+)
The system requires ip address configuration.	Physical requirement (+)
The system requires camera.	Physical requirement (+)
The system requires RPLIDAR A2 ranging sensor.	Physical requirement (+)
The system requires Pixhawk autopilot module.	Physical requirement (+)
The system requires Rasperry Pi 3B as drone controller.	Physical requirement (+)
The system requires 4 drone motors.	Physical requirement (+)
The system requires at least 2500 mAh battery to power.	Physical requirement (+)

The system will need the WiFi connection for the drone and application to be able to connect and communicate to the server through Internet. In order to achieve that, the IP address configuration is necessary. In order to connect to the server, the application and drone will have to know its IP address and make a connection to it. The server then matches the IP address of the application with the right drone to which it aims to communicate with. The camera will be attached to the drone to take the video as a source for object recognition to classify an object, as well as for live video streaming. The RPLIDAR A2 is also attached to the drone. It's used as a remote sensor to navigate the surrounded environment. It uses light in the form of a pulsed laser to measure ranges from the drone to the confined area. The data acquired from LIDAR will further be used with the SLAM algorithm to map the room.

For the flight controller, the module Pixhawk is introduced. Pixhawk autopilot module is responsible for the flight control. It will intake the flight command from the server and use it to control the flight. For the robot programming, ROS provides many open-source frameworks for robot development and interfacing, which reduces the robot programming effort. ROS is convenient for transferring implemented functionalities to personal robots. We use ROS for the framework needed to control the drone, SLAM computation, and path planning.

4.1.2 User interface

In this section, the UI implementation of the mobile application will be shown and described.

4.1.2.1 Launch screen page

As shown in Figure 4-1, the launch screen page will be shown once the user enters the application. It shows the logo of the application and disappears once the application is ready.

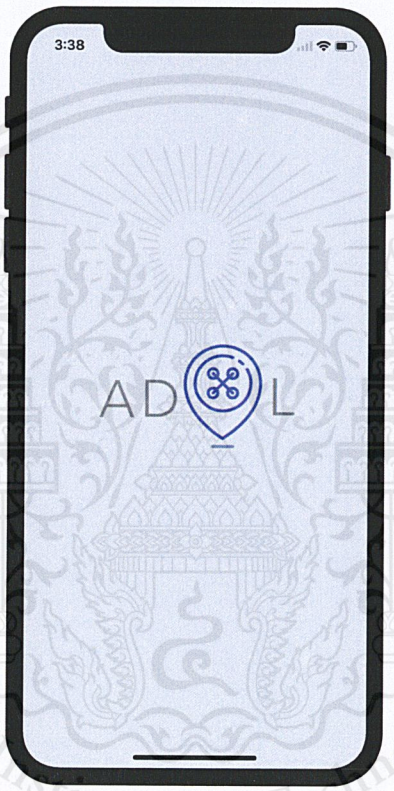


Figure 4-1. Launch screen page

4.1.2.2 Introduction page

There are three pages introduction, as shown in Figure 4-2, 4-3, 4-4. This is for the user to understand the purpose of this application and what it's going to be about in the application.

- Introduction 1: Search for an object using drone.
- Introduction 2: Can localize and return the location of object found.
- Introduction 3: Can view live video and map construction of the environment at real-time.

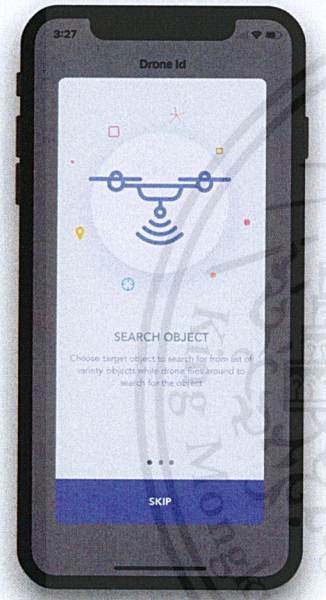


Figure 4-2. Introduction 1

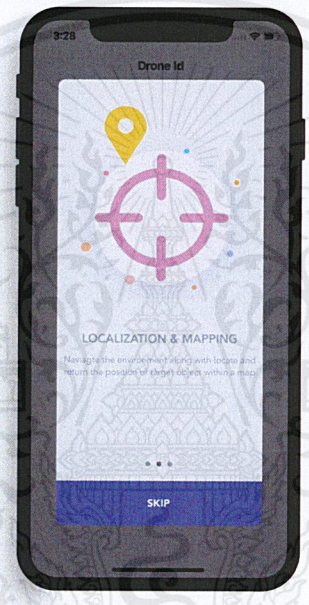


Figure 4-3. Introduction 2

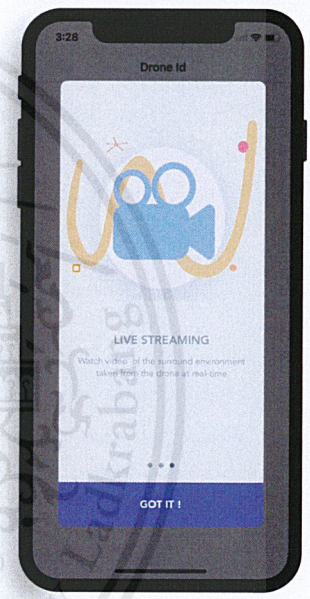


Figure 4-4. Introduction 3

4.1.2.3 Initialization and search

Figure 4-5 to 4-7 show the input pages for initialization of the system and search. This section of the application appears after the introduction where the user is asked to enter the IP address of the drone and object to search for. As entering the object to search for, a drop-down list of object related to one entered will appear for the user to pick from. Input will be validated before proceeding to the next step.

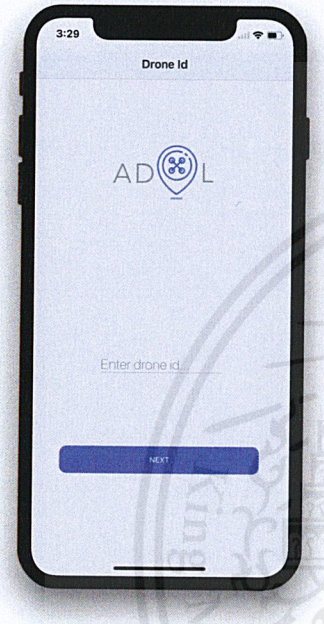


Figure 4-5. Input id

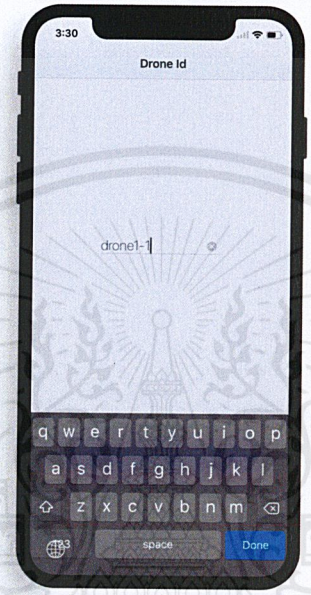


Figure 4-6. Input id

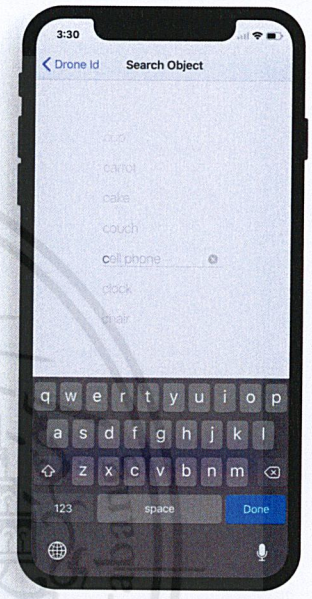


Figure 4-7. Input object

4.1.2.4 Live stream page

During the flight, the user can choose to view live streaming from the drone. This section of the application shows the live streaming from the drone, as shown in Figure 4-8.

Figure 4-8. Stream page

4.1.2.5 Real-time map construction page

As shown in Figure 4-9, this section shows the real-time map construction part of the application. The map is fed from the server where the graphics rendering is handled. The current location of the drone and object found will also be marked on the map.

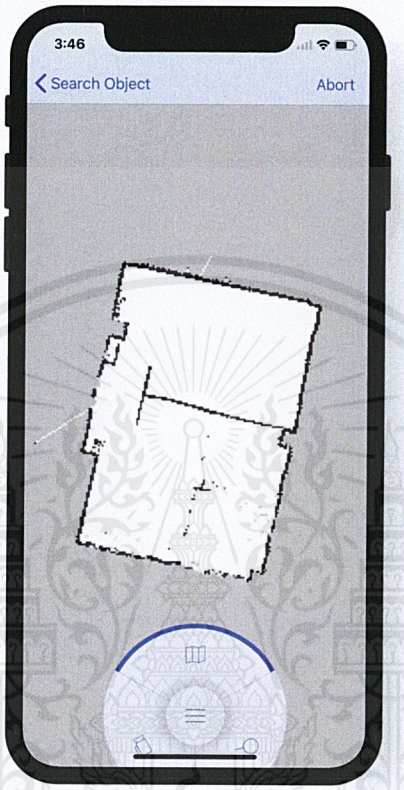


Figure 4-9. Map construction page

4.1.2.6 Record of object found

During the flight, once an object is found, its photo and location would be recorded. Photos of object found will be stored in the object found page as shown in Figure 4-10. Once user presses to view any photo, the photo will be selected to display as shown in Figure 4-11.

Figure 4-10. Record of object found

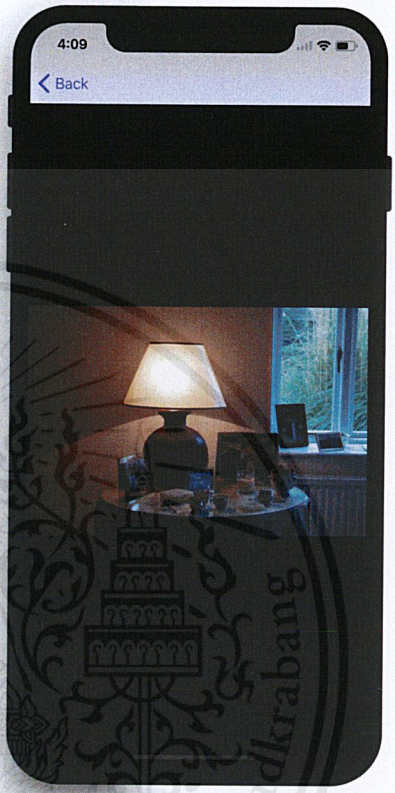


Figure 4-11. View object found

4.1.2.7 Abort mission

As shown in Figure 4-12, this appears when the user presses the abort button to end the mission. The application will ask the user if the user really wants to abort the mission before proceeding to the mission summary step.

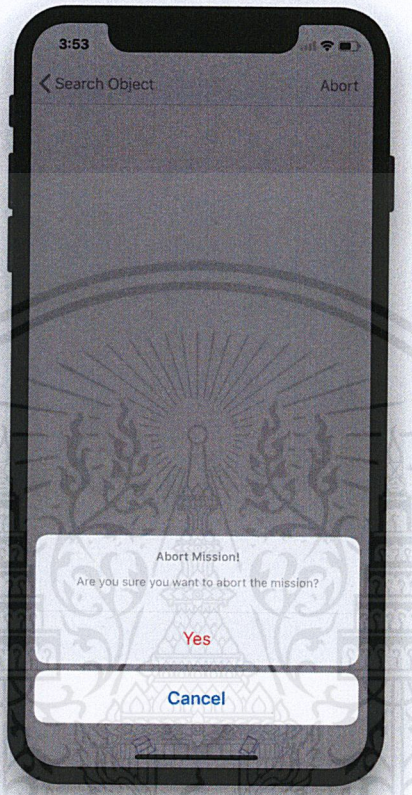


Figure 4-12. Abort mission page

4.1.2.8 Mission summary

At the end of the mission, all photos of the specified object will be displayed to the user, as shown in Figure 4-13. The user can look at these photos. If the user wants to view the map of a photo, the user can press on each item in the summary and the folding cells will open up and show the map with object's location, as shown in Figure 4-14.

Figure 4-13. Mission summary closed

Figure 4-14. Mission summary opened

4.1.3 Use case

This section will show the use case of the system as shown in 4-15. Reader will have a better understanding of how user can interact with the system.

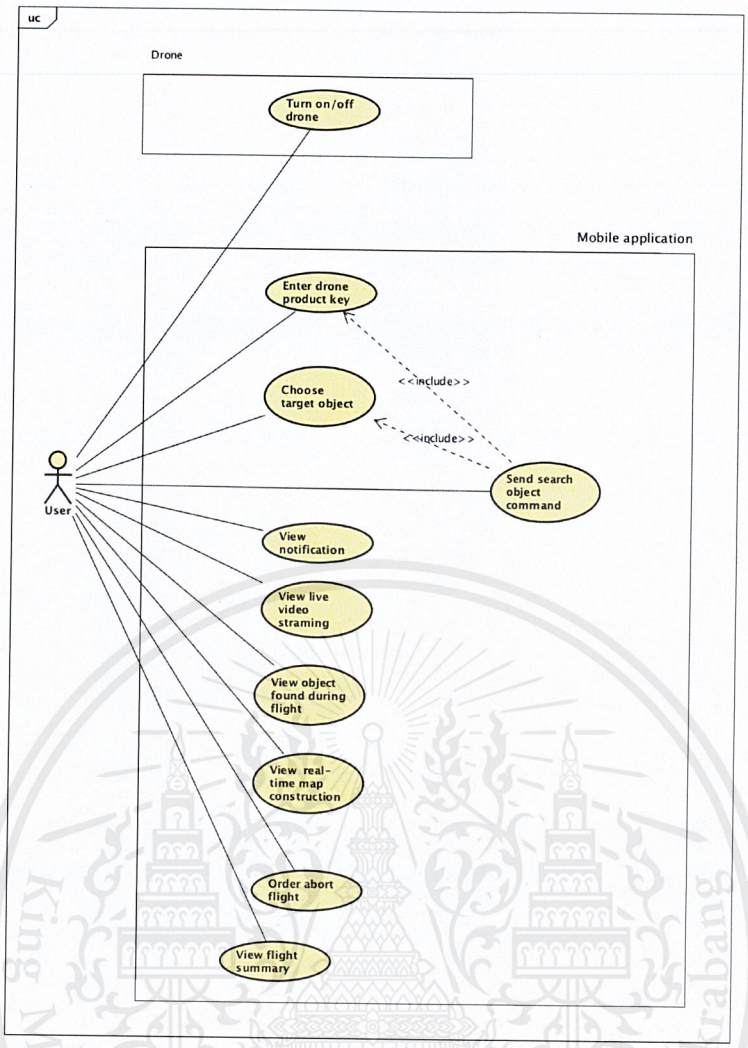


Figure 4-15. Use case diagram.

4.1.3.1 Use case description

1. **Choose target object and drone:**

   The user inputs number of the drone to which the user wants to connect and name of target object.

2. **Send flight command to drone (include use case 3):**

   The user presses confirm button to send all the input data to the drone.

3. **Send command to server:**

   The mobile application sends the data to the server for it to handle user's request.

4. **Order abort flight (include use case 3):**

   The user press abort button to abort the flight, or in another word to stop the drone's flight.

5. **View flight summary (include use case 3 and 6):**

   At the end of flight, the user views the summary of object found that shows the photo and location of found object.

6. **Summarize object found data:**

   The mobile application collect all the data of object found including it's photo and location, and summarize it into simple interface for user.

7. **View object found during flight (include use case 6):**

   During the flight, the user chooses to enter the mode view object found where the user can see a list of photo of object that has been found.

8. **View notification (include use case 9):**

   Once there's an object found, the user will see the pop-up notification to remind the user of the found object.

9. **Notify when object found:**

   The mobile application receives the message of found object and issues a pop-up notification showing that an object has been found.

10. **View live video streaming:**

    During the flight, the user chooses to enter the mode view video streaming where the user can see the video from the camera attached to the drone at real time.

11. **View real-time map construction:**

    During the flight, the user chooses to enter the mode view real-time map construction where the user can see the map of the room that has been constructed by data fed from the drone.

4.1.4 Activity diagram

The activity diagram in Figure 4-16 describes the overall flow of the system, which is separated into three parts: the mobile application, server, and drone.

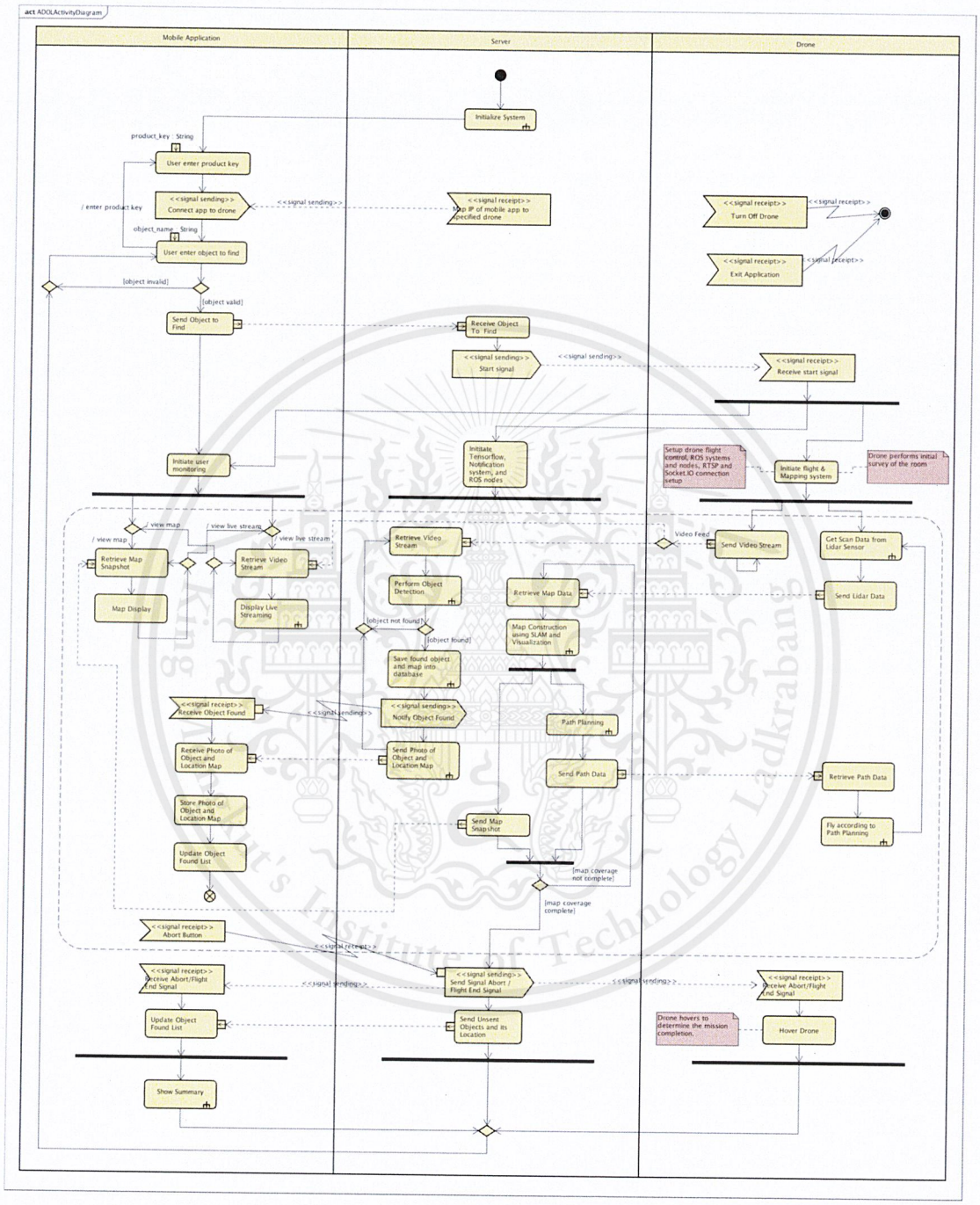


Figure 4-16. System activity diagram.

After the initialization of the server, the server will wait for the user to enter the drone's ID

and the object to search, in order for the server to pair the mobile application to the drone. Once the object entered by the user is valid, the server will send the signal to the drone to start the flight initialization. During the flight initialization, the drone will open the RTSP server for video streaming, enable the sensors, and fly up to the specified level. The server will initialize the object detection program, and prepare for the localization and mapping. Simultaneously, the drone will send the video stream and the laser scan data to the server, which the server will continuously get the video feed from the RTSP server to perform object detection using Tensorflow, and continuously get the laser scan data to perform mapping using SLAM algorithm. The map of the environment will be sent to the mobile application every one second, and once an object is found, the picture and the map including its position will be sent to the mobile application. If abort button is pressed during the flight or the flight has ended, the drone will terminate its flight by hovering and stops the running programs. The server will send all the collected data of the object found to the mobile application, in which the mobile application will use those data to display the flight summary to the user. The system will start again when the user enters the new object to search.

Figure 4-17 illustrates the activity diagram of the system initialization. The server will always be running, so when the user enters the application, the drone will be turned on. The drone will initialize its system by running the RTSP server program, set up the network to connect to the server by sending its IP and product key for the system to map with the mobile application.

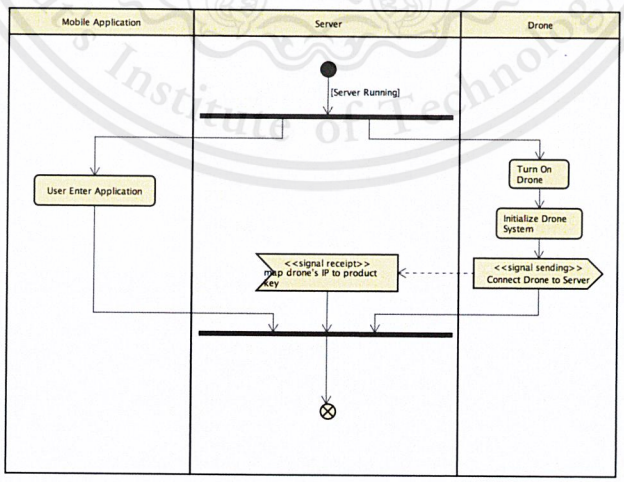


Figure 4-17. Initialize system activity diagram.

4.1.5 Class diagram

Figure 4-18 illustrates the packages and classes of the ADOL system. It contains the server package, drone package, mobile application package, socket-io package for the communication, RTSPPlayer package for the live video streaming, Firebase and APNS package for notification from the server to the application once an object is found.

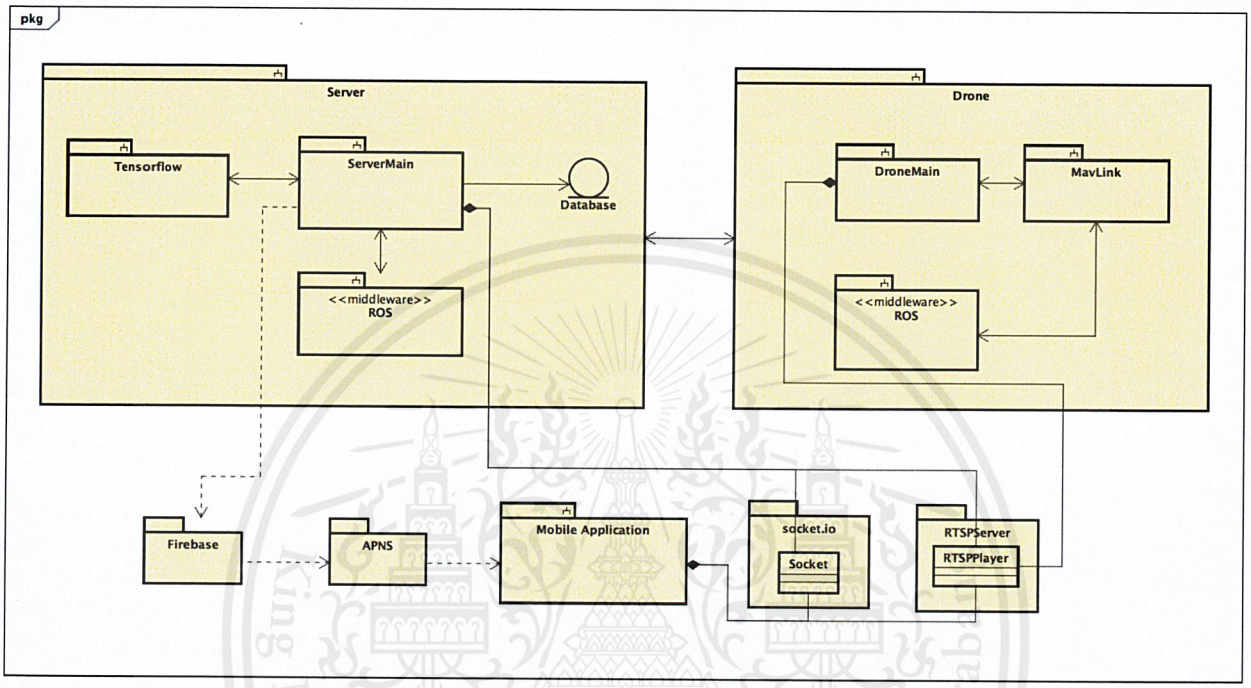


Figure 4-18. Packages in class diagram.

4.1.5.1 Server class diagram

The `ServerMain` package in Figure 4-19 includes the ServerMain class which is the main program that is always running and controls the communication between the mobile application and the corresponding drone. The `ServerMain` package communicates with the RTSPPlayer class, Tensorflow for object detection results from the video feeds, and ROS package for the computation for the navigation, localization, and mapping. Within the ROS package, the navigation package will be used for path planning and path coverage of the drone. `hector_slam` package is used for the localization and mapping by utilizing the laser scan data retrieved from the drone through the `rostopics`. The `roscore` package is the fundamental package for all ROS systems to start the ROS master, in order to register

rostopics.

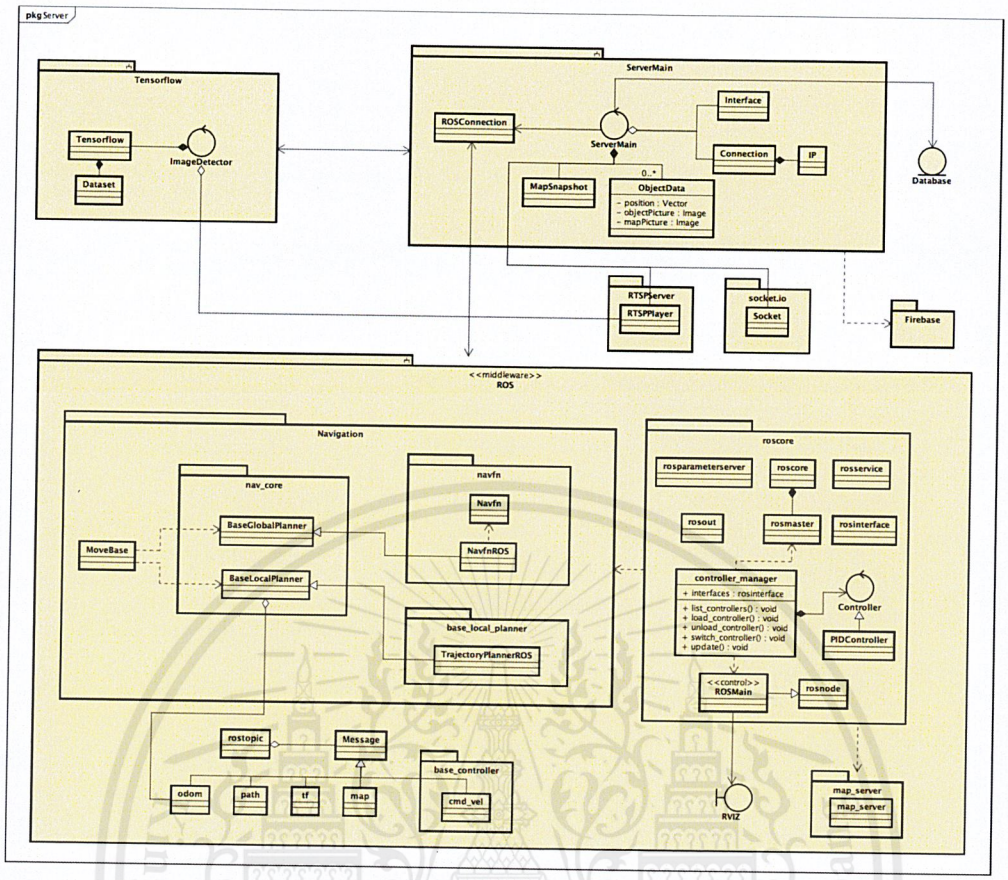


Figure 4-19. Server class diagram.

4.1.5.2 Drone class diagram

Figure 4-20 shows the drone packages. The `DroneMain` class is the main program of the drone, which it will enable the RTSPPlayer for publishing the video stream, use the ROS package for interfacing with the laser scanner and mavros interfacing with MavLink, which the MavLink package is used for drone control and interface with the flight controller. Due to Raspberry Pi's computation limit, the designed `hector_slam` package cannot be used on the drone. In this case, we solved the problem by transferring the package to the server to compute the localization and mapping, in which the drone will only send the laser scan data to reduce the computation load on the Raspberry Pi.

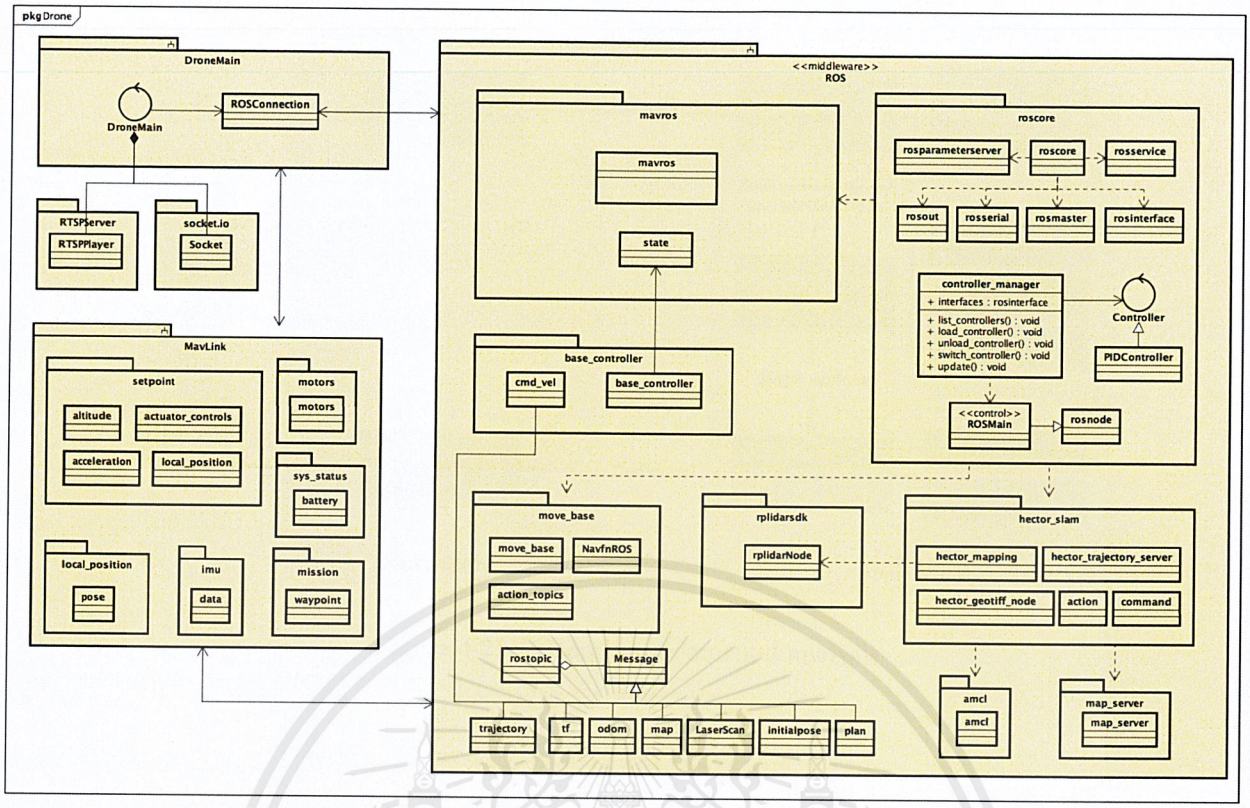


Figure 4-20. Drone class diagram.

4.1.6 Environment analysis

Figure 4-21 describes the environmental analysis of the drone. The environmental analysis diagram is separated into three parts: 1) the environment or the real world, 2) hardware sensors and actuators, 3) the software computation of the robot from the data of the environment from the sensors to perform the actuators. When the Lidar sensor scans the environment, the laser scan data will be used as the environmental perception to perform localization and mapping. Once the environmental model and the drone's pose are computed, the occupancy grid map is used for the robot's planning of path planning (mission planning) in order to perform the behavior and motion planning according to the path planned. Once the plan is computed, the targeted actions are then sent to the control part of the software, which the software will command the actuators or the motors of the drone through the flight controller. During this, the software will keep track of the path and trajectory of the drone. This process is done simultaneously.

Chapter 5

Experiments

This chapter explains the experimental results of the project along with providing description of elements essential for operation. Experiments are divided into four parts: the object classification, map construction, path planning and overall system. In each experiment, the objective, experiment setup together with the result are discussed. The measurement varies, depending on the functionality and accountability of each component regarding the major goal of the system.

5.1 Object classification

In order to provide appropriate result and evaluation, a relevant measurement and several object classification approaches are taken into account. Candidate models are selected based on the accuracy and speed in frame per second (fps) with respect to benchmarks and evaluation on real-world dataset.

5.1.1 Objective

The objective of object classification experiment is to investigate the performance of candidate models in order to select the most suitable algorithm for object searching task. The experiment aims at finding a CNN model which achieve the best combination of classification speed and accuracy. The criteria is defined as follows:

- **Speed** indicates the rate or number of frames that objects are classified in a period of time. In this context, the speed is measured in frame per second (fps).

- **Classification Accuracy** indicates the prediction accuracy of the model.

There are three variations of accuracy in model benchmarks as some models are trained on different dataset whereas the the models are further evaluated on real-world dataset. Details of the accuracy variation will be discussed in the following section. The target model is expected to deliver at least 9 fps in speed and 30 percent of classification accuracy.

5.1.2 Experiment setup

There are two main computation platforms for classification during the development: Raspberry Pi 3B with NCS and Ubuntu server. Details on the platforms and settings are described in Table 5.1.

TABLE 5.1
PLATFORM AND SETTINGS IN EXPERIMENTS.

CNN Model	Framework	Platform	Dataset
SqueezeNet v1.1	Caffe	NCS	ImageNet
SSD MobileNet v1	Caffe, TensorFlow	NCS, Ubuntu	MS-COCO, Pascal VOC071
TinyYolo v1	Caffe	NCS	Pascal VOC2012
GoogLeNet	Caffe	NCS	ImageNet
ResNet-18	Caffe	NCS	ImageNet
Faster R-CNN Inception v2	TensorFlow	Ubuntu	MS-COCO
Faster R-CNN ResNet 101	TensorFlow	Ubuntu	MS-COCO

The training data include ImageNet, MS-COCO, Pascal VOC0712 and Pascal VOC2012 while image data for this experiment are streamed from the camera attached to Raspberry Pi. The streaming took place in the computer laboratories and college's hall. Images from streaming were resized into 800×600 (width $\times$ height) resolution by using geometric image transformations function in OpenCV.

5.1.3 Result and discussions

At early stage, Raspberry Pi with NCS was used as target computation platform. After the experiments, literature reviews of CNN model and change in system design, the computation platform and model are altered. For the sake of clarity, benchmarks of object classification for the first and second approaches are respectively described in Tables 5.2 to 5.3.

TABLE 5.2
BENCHMARK OF OBJECT CLASSIFICATION ON RASPBERRY PI 3B WITH NCS.

CNN Model	Frame rate (fps)	Top-5 accuracy (%)	Top-1 accuracy (%)
SqueezeNet v1.1	16.63	80.3	57.5
SSD MobileNet v1	10.81	89.56	71.1
TinyYolo v1	7.83	83	61.1
GoogLeNet	9.21	88.9	68.7
ResNet-18	8.83	89	69

TABLE 5.3
BENCHMARK OF OBJECT CLASSIFICATION ON UBUNTU SERVER.

CNN Model	Frame rate (fps)	COCO mAP
SSD MobileNet v1	30	21
Faster R-CNN Inception v2	17.24	28
Faster R-CNN ResNet-101	9.43	32

There are three variations of accuracy measure, depending on the dataset. Top-n accuracy is the rate of correct prediction within top n labels output. COCO mAP is a mean average precision of MS-COCO dataset. From the evaluation of performance and trade-offs in speed and accuracy among models. The second approach tends to provide the most promising path toward the ideal speed and accuracy. Further result and evaluation of the second approach is illustrated in the following discussion. Figures 5-1 to 5-4 present some input and output images from different CNN models (see appendix B). The bounding box represents objects location while label represents object type and probability.

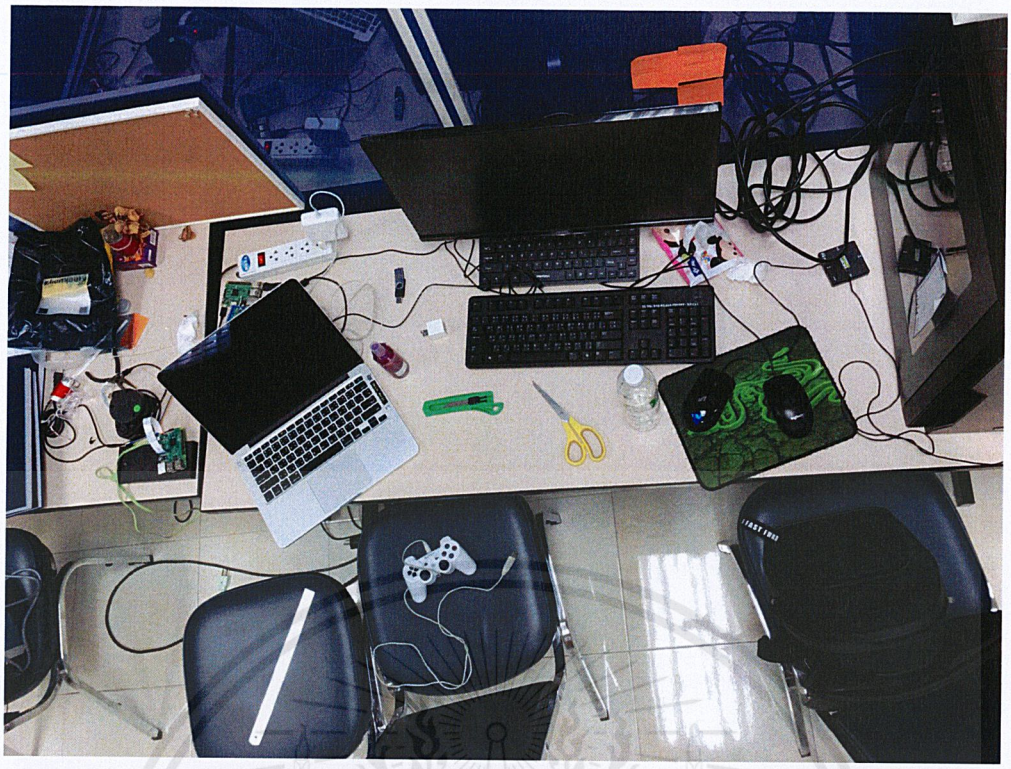


Figure 5-1. Input from camera stream.

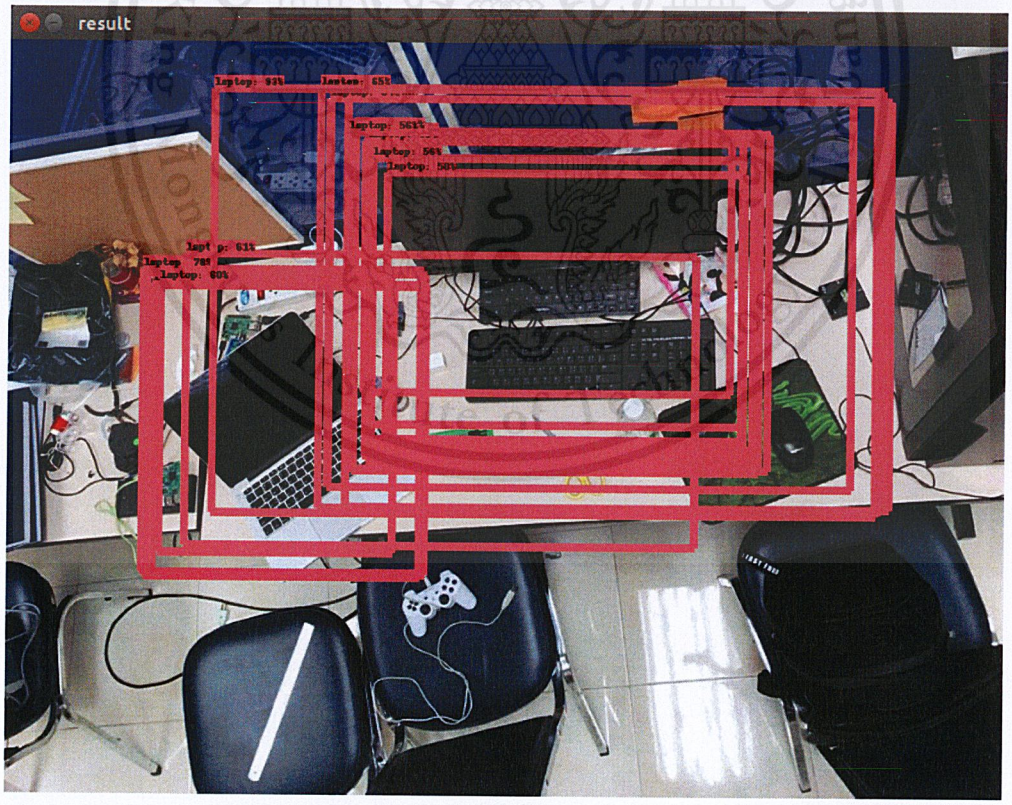


Figure 5-2. Classification result using SSD MobileNet V1 model.

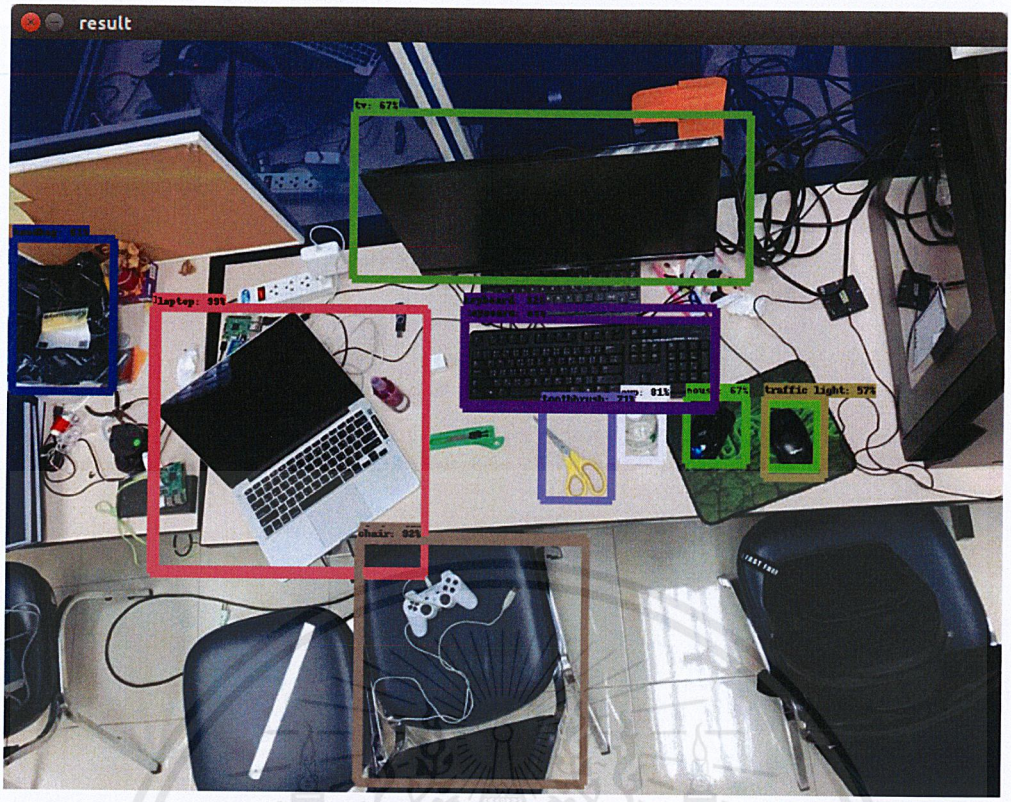


Figure 5-3. Classification result using Faster R-CNN ResNet-101 model.

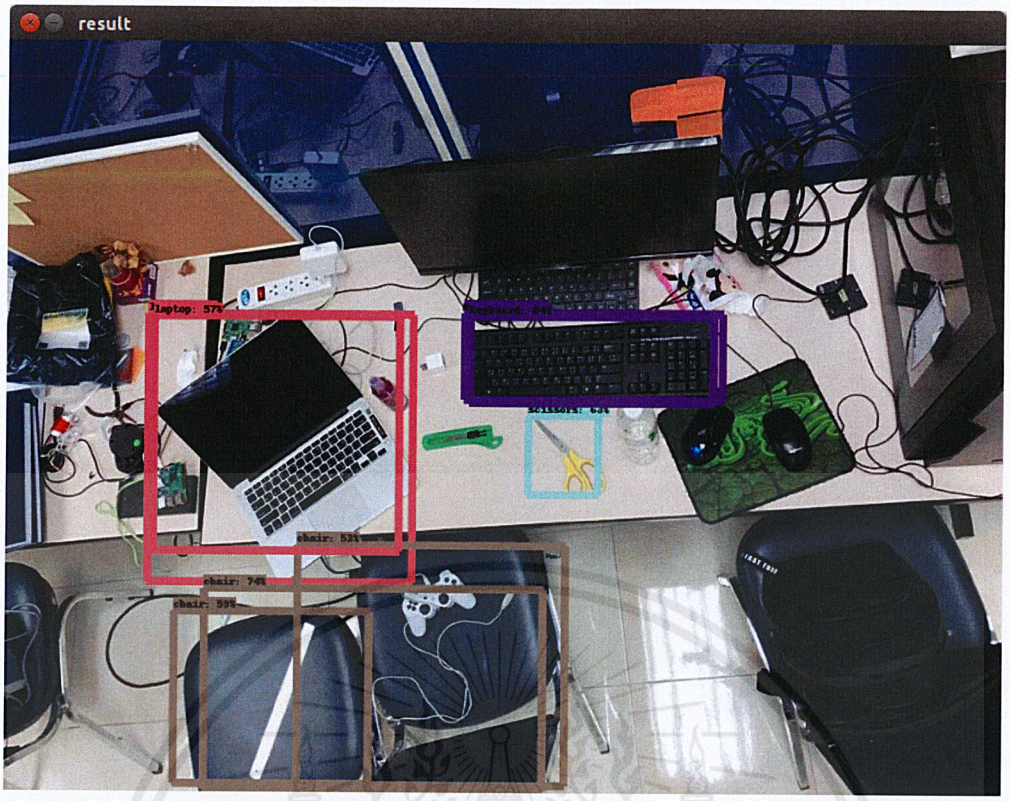


Figure 5-4. Classification result using Faster R-CNN Inception v2 model.

TABLE 5.4
MEASUREMENT RESULT OF OBJECT CLASSIFICATION.

CNN Model	$Rate_{detect}$ (%)	$Rate_{classify}$ (%)	$Accuracy_{classify}$ (%)
Faster R-CNN ResNet-101	52.63	67.14	35.34
Faster R-CNN Inception v2	40.60	68.52	27.82
SSD MobileNet v1	6.02	12.5	0.75

Table 5.4 refers to the result of object classification. The measures are calculated as follows:

$$Rate_{detect} = \frac{n_{detect} * 100}{N}, \tag{5.1}$$

$$Rate_{classify} = \frac{n_{classify} * 100}{n_{detected}}, \tag{5.2}$$

$$Accuracy_{classify} = \frac{n_{classify} * 100}{N}, \tag{5.3}$$

where $Rate_{detect}$ is the detection rate, $Rate_{classify}$ is the classification rate while N, n_{detect} and $n_{classify}$ are the total number of objects, count of object being detected and correctly classified respectively. The $Accuracy_{classify}$ is the classification accuracy. From the experiment, we can conclude that running Faster R-CNN ResNet-101 on Ubuntu server leads the most promising solution toward object searching problem.

5.2 Flight control

To provide an insight of flight experiment, the works contain attempts to perform the test in different characteristics affiliate with different factors including testing environment, payloads, sensors and platforms in order to testify and understand limitations of configuration and methodology.

5.2.1 Objective

The objective of flight control experiment is to investigate the efficiency and practicality of hardware and software integration along with gathering information for further improvement. Relevant components and factors were evaluated in each flight and the result may also be further used for generic adjustment according to the application. The following factors are concerned in the experiment.

- **Stability** indicates the ability to conform according to input commands and noise bearing which may have an impact on the system.
- **Localizability** indicates the ability to autonomously localize.

5.2.2 Experiment setup

The flight setup includes drone equipped Raspberry Pi 3B, Pixhawk flight controller, FLY SKY radio transmitter and radio receiver as main components. The configuration and tuning are performed with Mission Planner software. Some components are varied in different experiments which are RPLIDAR A2 and GPS due to the signal availability and noise factors.

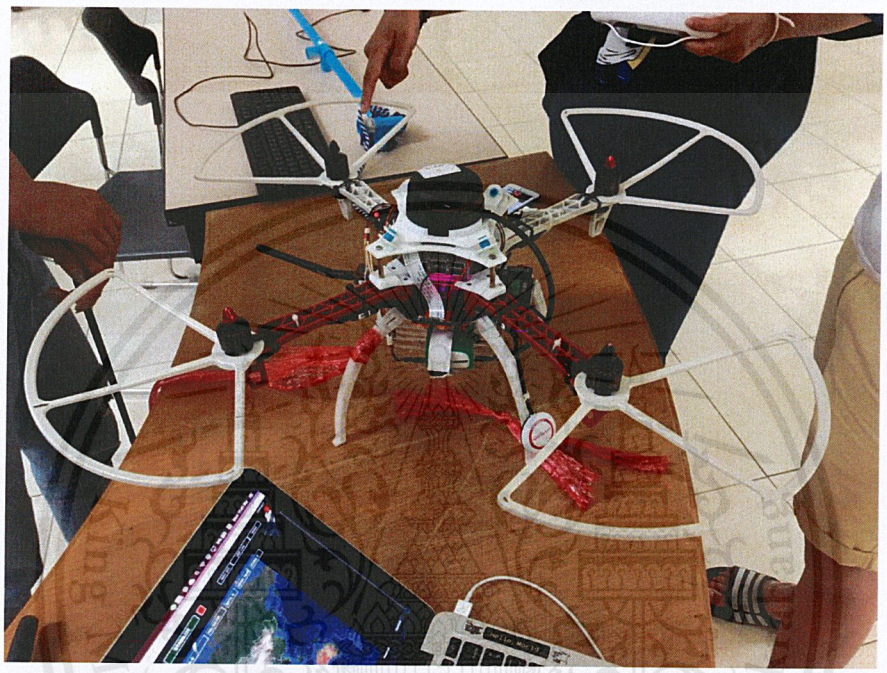


Figure 5-5. Fully-equipped drone configuration.

5.2.3 Result and discussions

Experiments in confined area and open-air area were conducted. In confined area, the weather factors were eliminated and the drone was cable of conforming with input command, however, the drone had some drifts due to self-generated wind and change in air pressure. In the first flight, there were stability issues. Figure 5-6 describes that the vehicle's roll, pitch and yaw were not conform with the designated orientation value. Afterward, the configuration and tuning were revised.

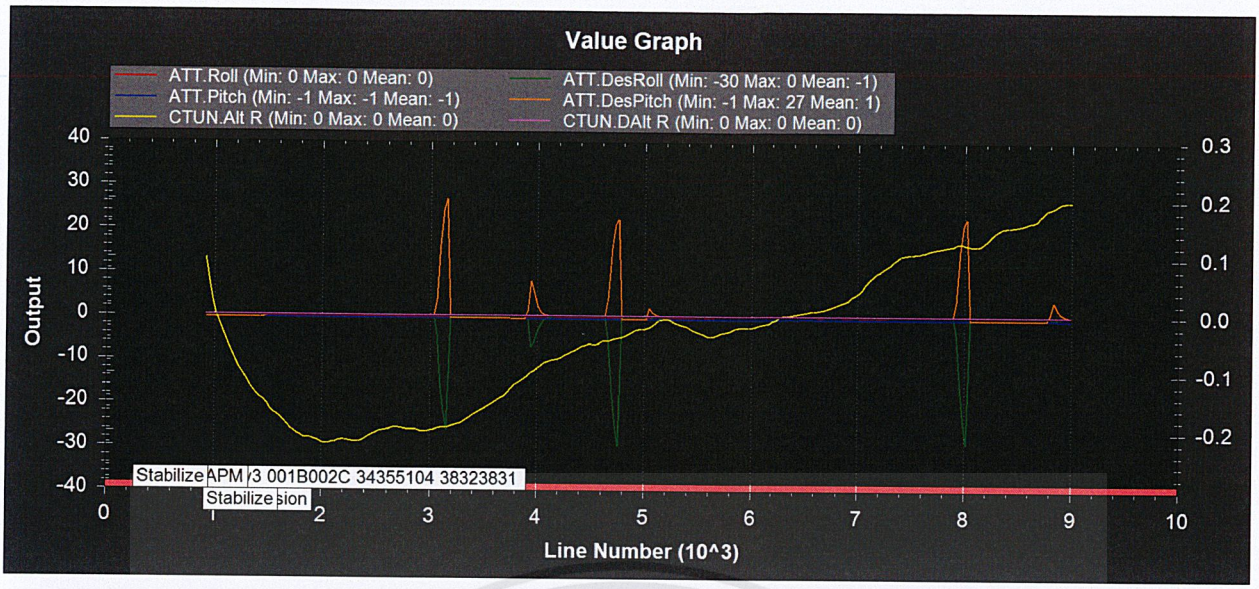


Figure 5-6. The graph showing the desired value and actual value of vehicle orientation.

For the localizability, the use of SLAM is critical as the GPS does not operate indoor; the Pixhawk firmware shall be modified which is beyond the scope of project. The experiment on SLAM will be elaborated in the following section.

For the stability in open-air, the drone is capable of maintaining the altitude and directions better than the former configuration despite some drift due to the wind. In the diagram below the vehicle's roll, pitch and yaw closely follows the designated value, showing an improvement of mechanical control after several flight and tuning.

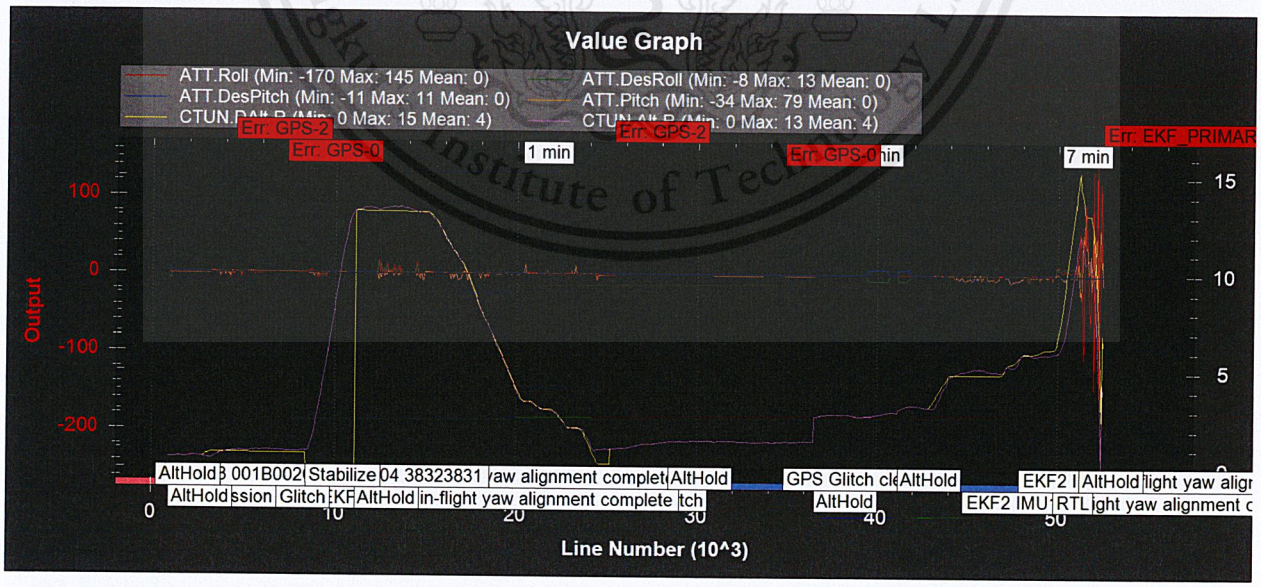


Figure 5-7. The graph showing the desired value and actual value of vehicle orientation.

On the other hand, in autonomous flight the GPS module has some measurement errors which leaded to the crash. Below is the graph describing the GPS glitch indicating the limitation of the hardware sensor. Figure 5-8 shows the decrease in the number of satellites visible (NSats) and increase in the dilution of precision (HDop).

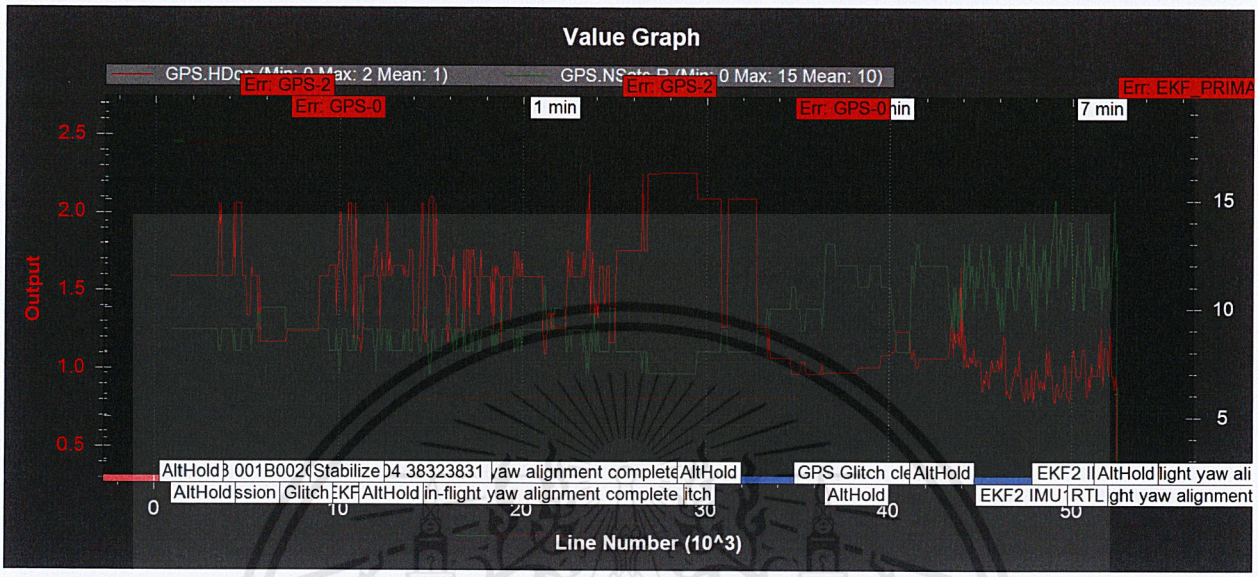


Figure 5-8. The graph showing GPS glitch values.

HDop below 1.5 indicates a positive result while value above 2.0 indicates a negative result, the diagram shows the frequent fluctuation of the HDop value which causes the localization error. The NSats values fell in a low range. The problem with GPS glitch can be solve by sanity checking the position with Extended Kalman Filter (EKF) implementation.

5.3 Receiving lidar data

5.3.1 Objective

The objective of this experiment is to see how effective the RPLidar A2 laser scanner can detect the obstacles, landmarks, or objects around.

5.3.2 Experiment setup

`rplidarNode` from the `rplidar_ros` package is used for this experiment. This node communicates with the laser scanner and publishes its scans to `/scan` topic with message type `sensor_msgs/LaserScan`. In other words, it reads RPLIDAR raw scan result using RPLIDAR's SDK and converts to ROS `LaserScan` message. The software used for display is called `rviz`. The experiment is conducted in IC15 room, or the computer vision lab of International College.

5.3.3 Results and discussion

Figure 5-9 shows the visualization of the Lidar sensor in `rviz` through the subscription of `scan` topic.

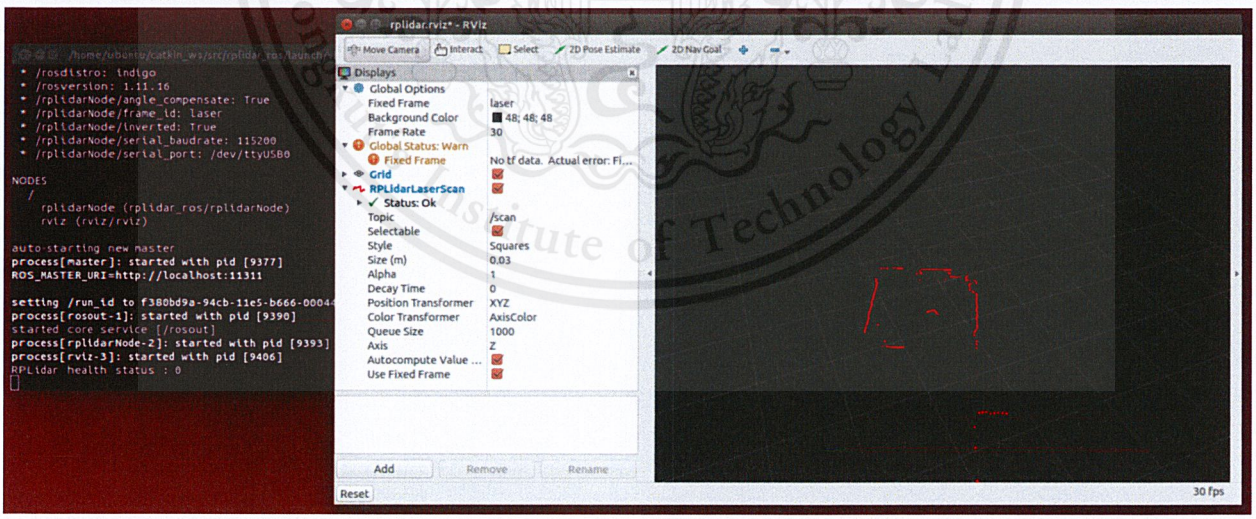


Figure 5-9. Display Lidar data as point clouds

As a result, the Lidar laser sensor can scan around the room quite accurately. It can

detect the walls, obstacles, moving objects, and people. The scan points are represented as red dots and are refreshed and updated according to the changing scan data, which publishes around 180-240 points per 20Hz. However, the dots return did not cover all 360 degrees, but it is sufficient for the use of localization and mapping.

5.4 Simultaneous localization and mapping

5.4.1 Objective

In order to experiment the performance of the map construction algorithm, compared with the dimensions of the real room without the use of odometry. The resolution of the map effects how large each pixel is used from the scan and mapping, so the most suitable resolution of the map to be displayed in the application has to be chosen in advance.

5.4.2 Experiment setup

The experiment is done by holding up the RPLidar A2 Lidar sensor around 2 meters, connected to a computer through USB running on Ubuntu and ROS, the Lidar sensor is also connected to the battery as the USB port of computers and microprocessors provides only 0.5A, while Lidar sensor needs 1.2A to 1.5A. These equipment are used to scan the IC15 room or the computer vision lab of International College. This experiment is conducted to see how much the mapping algorithm covers the room and to evaluate the effectiveness of the Lidar sensor used. From the obstacles/landmarks detected by the Lidar sensor to the constructed map using SLAM algorithm.

After starting the mapping program, the person holding up the Lidar up navigates in the center of the available walk path. The Lidar sensor then scans around with the maximum distance of 6 meters, rotating 360 degrees which return the array scan data of the distances between the obstacles/landmarks for 20Hz. After navigating the whole room, the map data is saved as `.pgm` format which we can use to evaluate the coverage areas.

The coverage area done by the mapping was measured as the percentages of the number of pixels that covered the constructed map depending on each resolution set comparing with

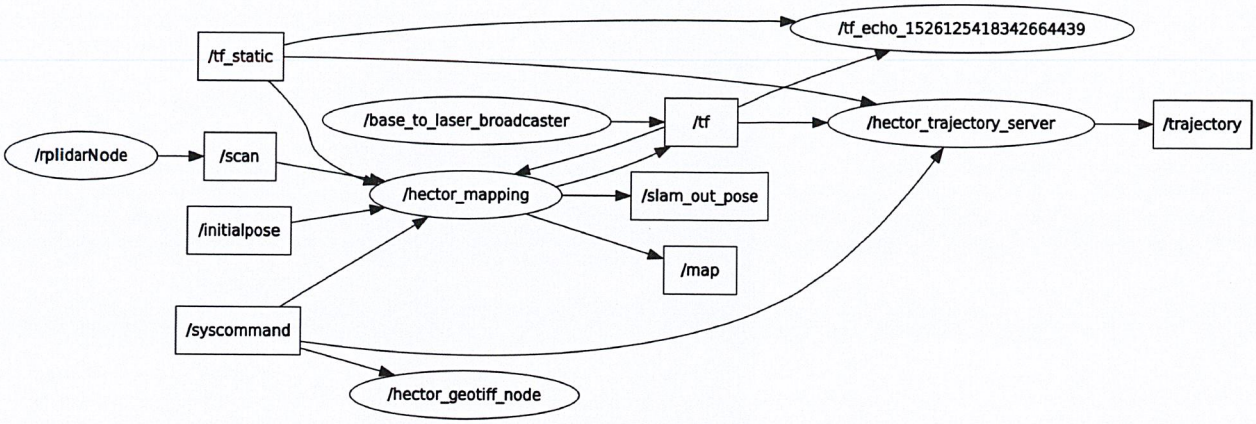


Figure 5-11. Graph of active nodes and topics

Next, with some parameters adjusted and some rotation error fixing, the map of the room can be constructed without the rotation issues. We can then experiment in the map resolutions. The map resolution units are in meters per pixel (m/px).

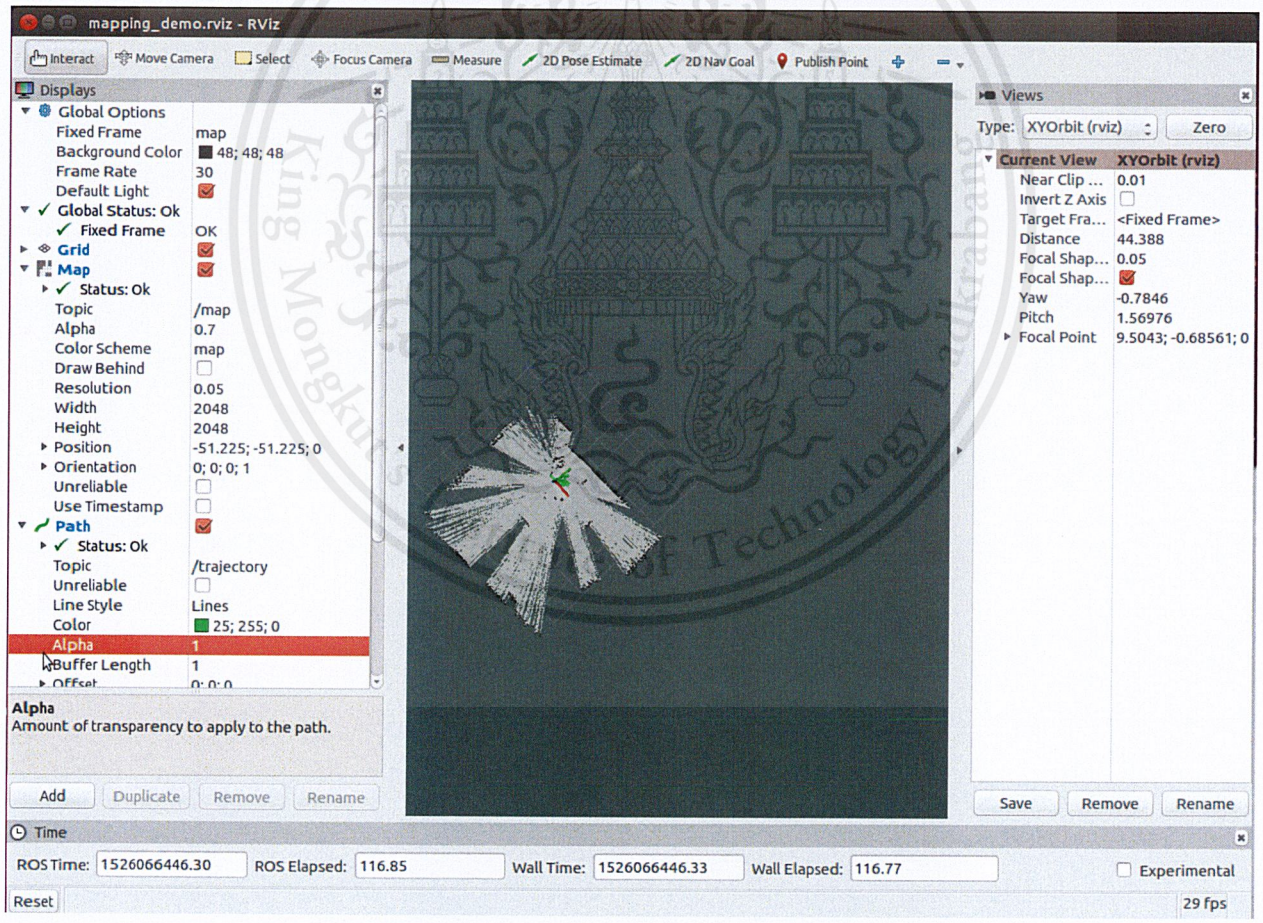


Figure 5-12. Map Construction using 0.05m/px

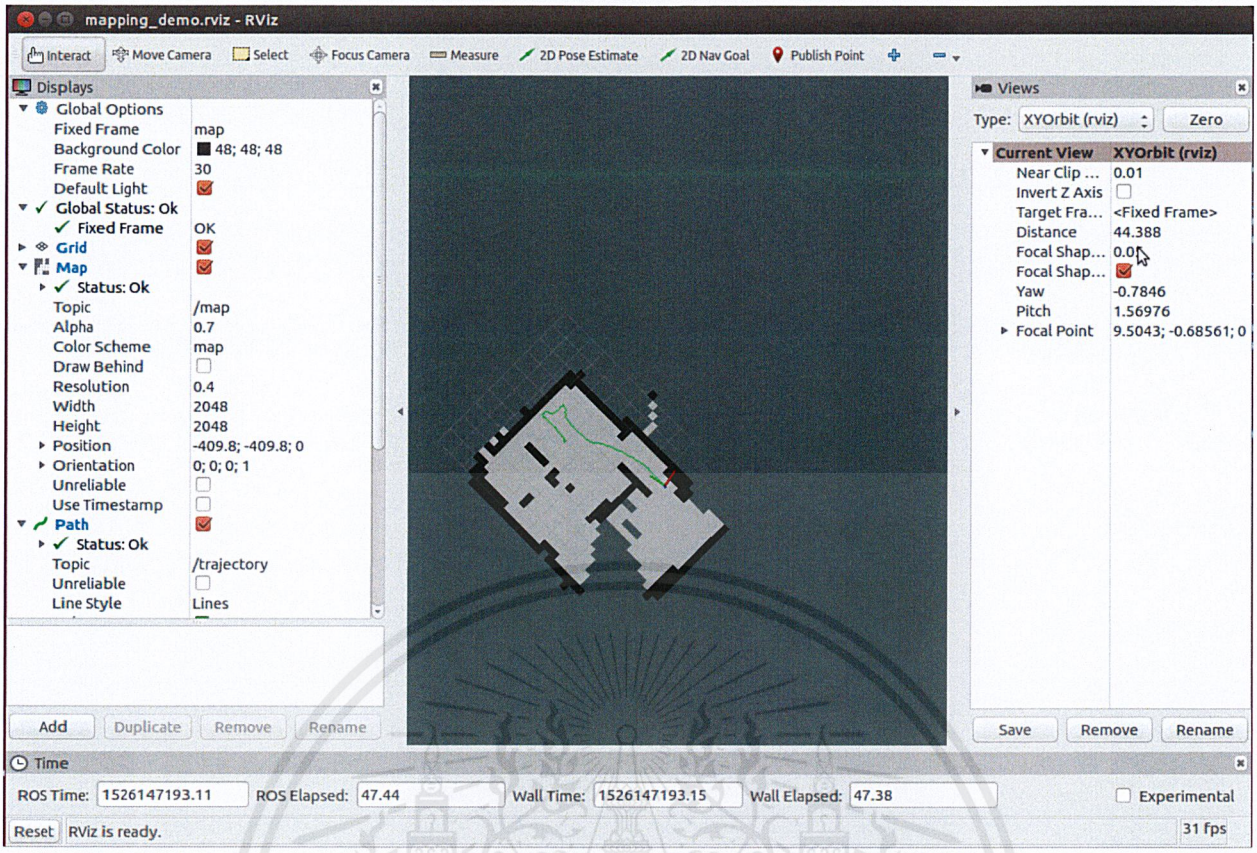


Figure 5-13. Map Construction using 0.4m/px

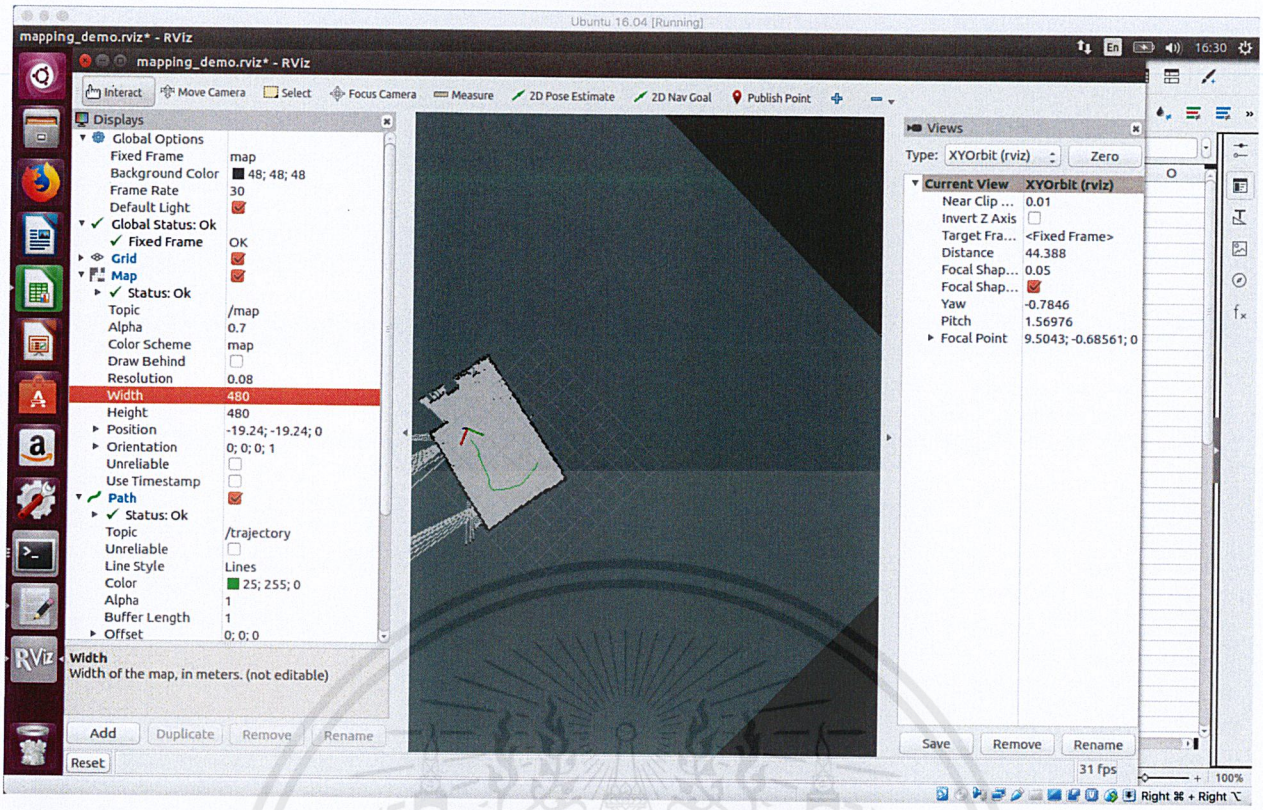


Figure 5-14. Map Construction using 0.08m/px

Figures 5-12 to 5-13 are the maps constructed and displayed in `rviz` from the laser scan data and the mapping algorithm. The maps exported are displayed in Figures 5-15 and 5-16.

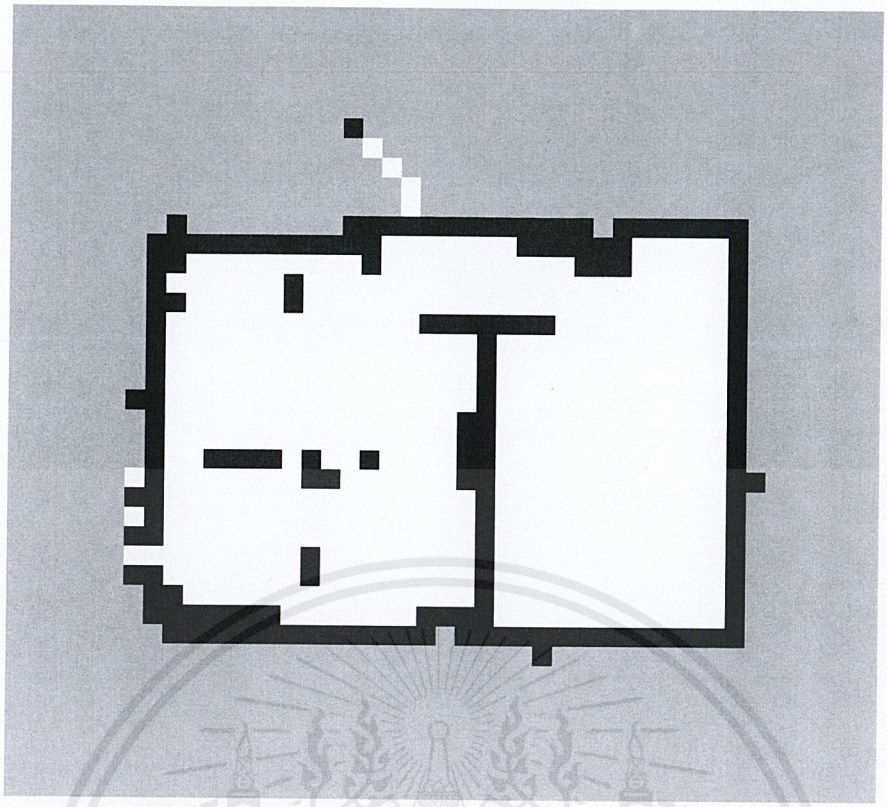


Figure 5-15. Exported map with 0.4m/px resolution

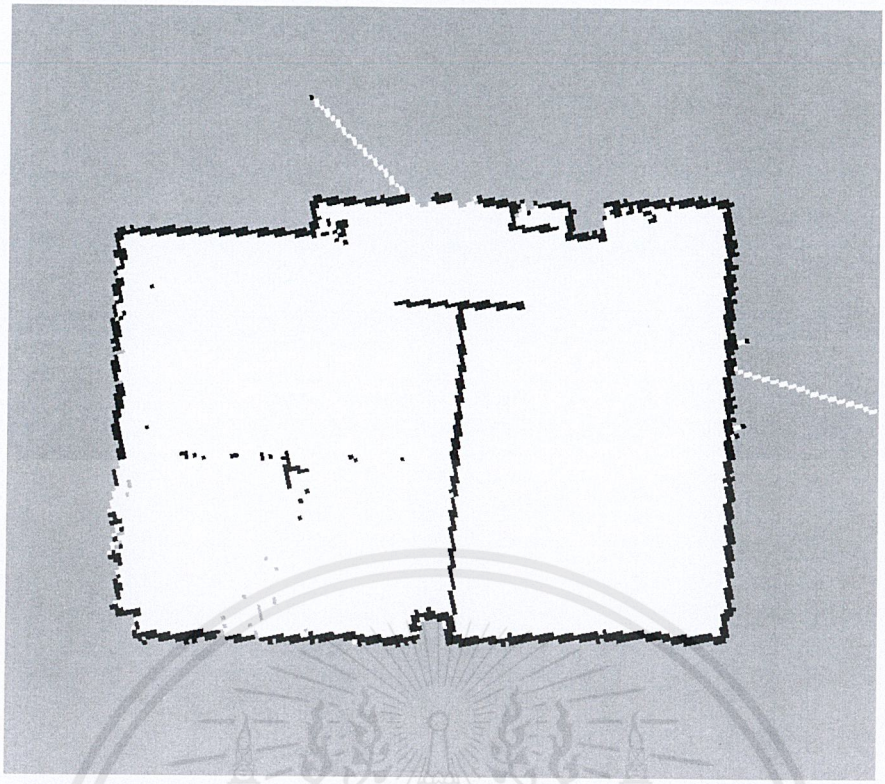


Figure 5-16. Exported map with 0.08m/px resolution

These two resolutions are selected for this experiment as to see the clear difference of low and high resolution of mapping. The 0.4m/px resolution is selected, as this size corresponds to the size of the tiles in the room. The 0.05m/px resolution are too small, so we pick a larger resolution which is 0.08m/px for the experiment. As from Figure 5-15 the walls are almost covered, however, some obstacles are missed out as the resolution is too low. The map from Figure 5-16 is more detailed. As the resolution is high, there will be many unnecessary black dots on the map. Both of the maps constructed contains some errors, as the Lidar scanner could not effectively detect the windows, as there exist long lines that pass through the window.

TABLE 5.5
MAP CONSTRUCTION COVERAGE AND ERROR

Resolution	Coverage (%)	Error (%)
0.4 m/px	97.51	2.82
0.08 m/px	95.69	5.11

As a result shown in Table 5.5, 0.4m/px map misses some of the details and 0.08m/px map is too detailed to be displayed to the users, so 0.2m/px resolution would be the suitable resolution for the application, as it collects more necessary landmarks and obstacles. However, there exists some errors from the windows, and miss some obstacles. On the other hand, the map constructed is sufficient for the users to determine the area and location.

Chapter 6

Conclusion

Seeing there is a room for improvement in the drone system, we focus on developing the drone system by integrating it with other technological aspects. The objective of this project is to develop a prototype system focusing on developing, applying, and integrating different technological aspects including object recognition, path planning, indoor navigation, and control via mobile application to the drone system. The aim of this project is to develop the base for further drone system extension needed for future application. Within our scope, system developed should be able to fly autonomously and localize object.

The system composed of three major components which are the drone controlled by the Raspberry Pi, server, and mobile application. Attached to the drone are sensors that retrieve data from the surrounding environment for further processing of the application. Raspberry Pi handles these data along with drone control. These data are passed down to the server to be further processed and accomplished the process of object recognition, path planning, indoor navigation, and communicating with mobile application. The mobile application handles communication between the user and server.

To achieve the goal, experiments are to be made on following aspects: object classification, path planning, flight control, receiving lidar data, simultaneous localization and mapping, and the integrated system. The object classification experiment shows that running Faster R-CNN ResNet-101 on Ubuntu server leads the most promising solution toward object searching problem for the system in speed and accuracy aspect. For navigation, the path planning achieved in the simulation has demonstrated the capability to perform au-

tonomous navigation based on the given map. The flight control experiment has investigated the efficiency and practicality of hardware and software integration which supply relevant information for system configuration and improvement. Getting across the limitation of hardware sensors and firmware are also one of the challenging factors in indoor navigation.

From experimenting with RPLidar A2, which is one of the light-weighted and effective Lidar sensor, the result shows that it is able to scan landmarks, obstacles, objects, and humans with precision accurate enough to achieve the goal of the project. However, sufficient power should be supplied.

For localization and mapping, an experiment was made with the low and high-resolution maps. The 0.4m/px map misses some of the details and 0.08m/px map is too detailed to be displayed to the users, so 0.2m/px resolution would be the suitable resolution for the application, as it collects more necessary landmarks and obstacles. However, there exists some errors from the windows, and miss some obstacles. On the other hand, the map constructed is sufficient for the users to determine the area and location.

After experimenting on the subsystems, these subsystems were all integrated together to test as the whole system. However, the result show that integration could not be made possible due the many challenges. First was due to the flight controller's firmware limitation for indoor navigation which affects the the drone control capability. Next is the limitation of platform processing power, in this case is Raspberry Pi. Also, there are problems having to do with the lack of real-time operating system availability in embedded platform.

Most of the subsystem was being achieved. The implementation of autonomous navigation and planning was made possible in the ROS simulation which based on native library and real-world data. Also, object localization is achieved in the real world using pre-trained TensorFlow Faster R-CNN ResNet-101 model which. Flight control was successfully made from the Raspberry Pi with the aid of GPS. Also, the mobile application could successfully communicate to the drone system. However, there are some challenges faced as mentioned earlier that do not allow the integration of the whole system.

Along with challenges faced from the project, it was learned that the integration of the system could be made possible if the firmware is made to support indoor navigation. A configuration on the firmware could be made, however, it is beyond the scope of this project.

Also by having the platform processing power that is high enough along with the real-time operating system availability in embedded platform, integration of the subsystems could be made possible. However, hardware being used in this project is limited to the cost limitation.

In conclusion, the project has achieved its goal of flying autonomously and localizing the object. Different technological aspects are achieved in being used to improve the drone system. Although there are some challenges that prevent it from the perfect system integration, with application on the suggested improvement methods, the system should be able to be used in practice and extended further for future development.

6.1 Future work

For the future extension, applying different technological aspects to the drone system would allow it to bring about many useful applications. For example in the future extension, the recognition capability can be extended with face recognition for security or other applications. Extended from the base prototype system, an adaptive exploration system for both indoor and outdoor survey could also be developed. Moreover, with the improvement in the processing platform, it is possible to achieve the real-time on-board vehicle controller. With cutting-edge technology, in the future drone technology could be extended much further than one can ever imagine.

Appendices

Appendix A

Communication between Raspberry Pi and Pixhawk

The protocol used to communicate between Raspberry Pi running Raspbian and Pixhawk flight controller over serial communication is the MavLink protocol. Pixhawk's TELEM2 port is connected to the Raspberry Pi's Ground, TX and RX pins and Raspberry Pi can be powered by connecting the red V+ cable to the +5V pin or from USB.

A.1 Raspberry Pi and Pixhawk communication setup

The first step is to setup the Raspberry Pi by flash one of the existing APSync images and setting up the Pixhawk by connecting to the ground station.

A.2 Raspberry Pi setup

The Raspberry Pi should be set up by able to connect with an SSH/Telnet client, install the required packages on the Raspberry Pi for Internet connection confirmation, and to disable OS use of serial ports using

A.3 Mission planner

The last step is to connect the Pixhawk to the Mission Planner on the Raspberry Pi. The Pixhawk will respond to MAVLink commands received through Telemetry 1 and Telemetry 2 ports and adding an –out <ipaddress>:14550 to the MAVProxy startup command with the being the address of the Raspberry Pi running the mission planner.

The control and drone flight can be monitored through Mission Planner software or the MavLink protocol in the Raspberry Pi.

Appendix B

Classification results from TensorFlow models

The classification results performed by different CNN models are separated in Section B.1 to B.14. In each section, there are tables describing the observation from the classification.

B.1 Image 1

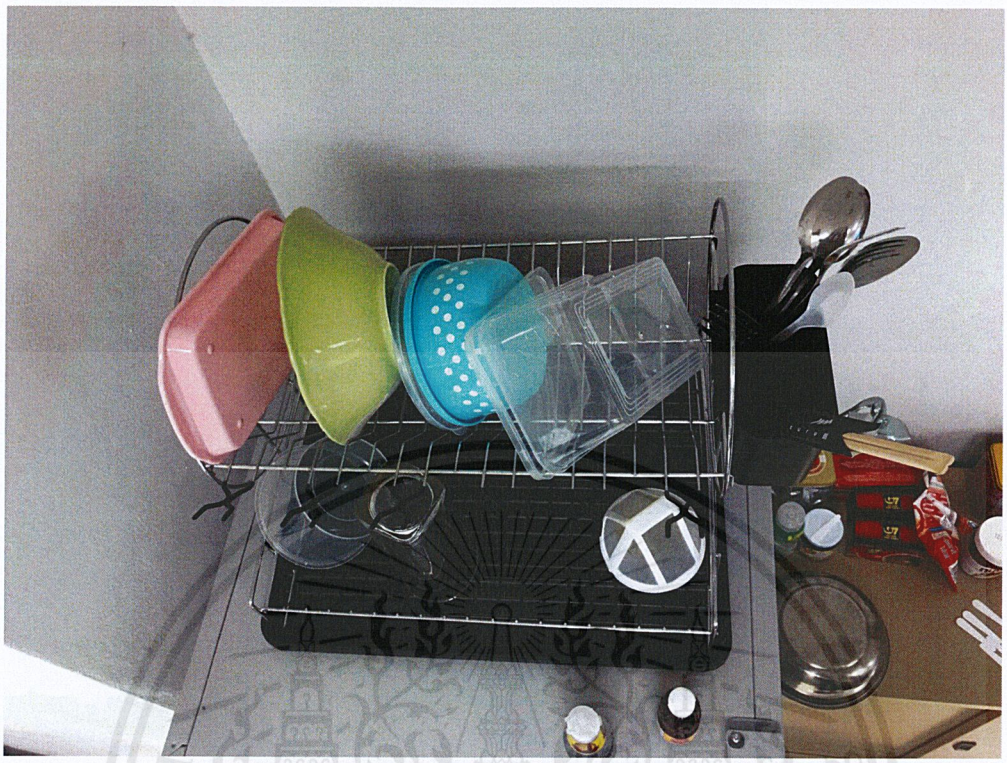


Figure B-1. Input from camera stream.

Figure B-2. Classification result using SSD MobileNet V1 model.

Figure B-3. Classification result using Faster R-CNN ResNet-101 model.

Figure B-4. Classification result using Faster R-CNN Inception v2 model.

From Figures B-1 to B-4, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected three objects. Tables B.1 to B.2 list the result sorting from left to right.

TABLE B.1
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	purple	keyboard	59
2	white	bottle	58
3	beige	spoon	50

Faster R-CNN Inception v2 model detected one object.

TABLE B.2
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL

Bounding box	Color	Label	Probability (%)
1	beige	spoon	63

B.2 Image 2

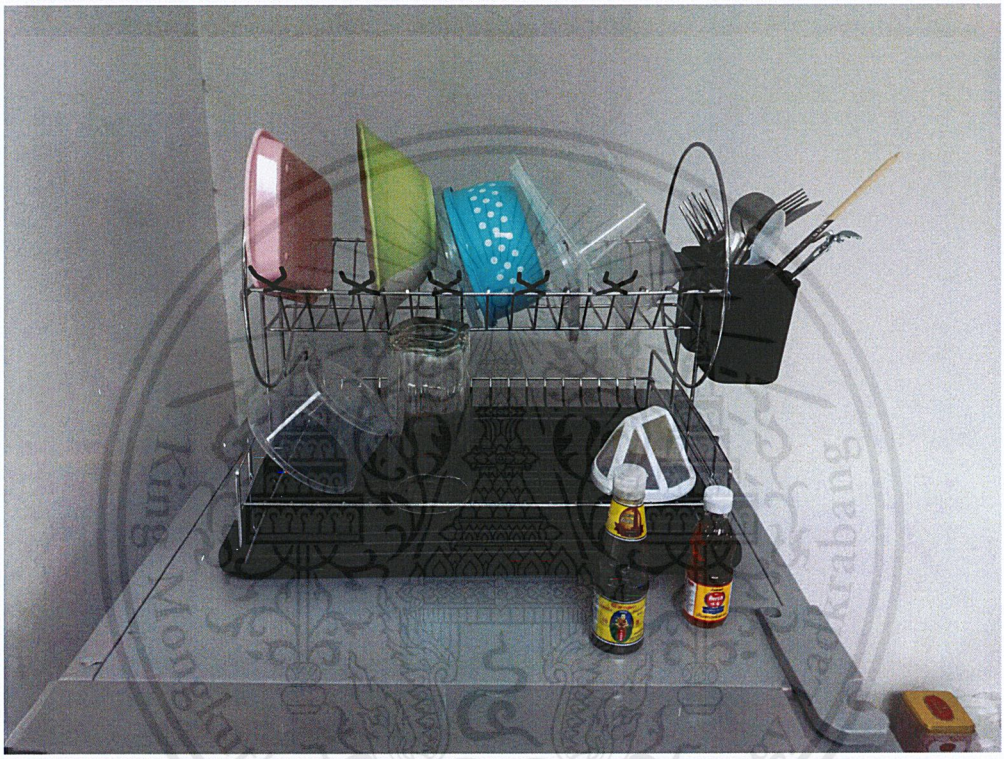


Figure B-5. Input from camera stream.

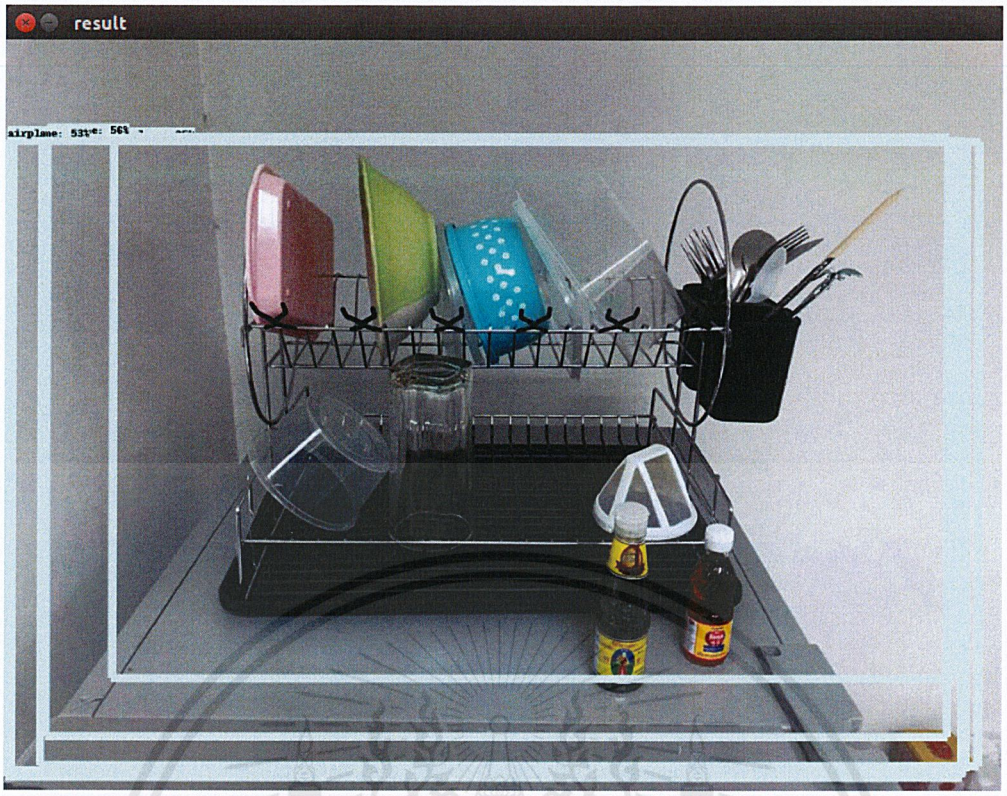


Figure B-6. Classification result using SSD MobileNet V1 model.

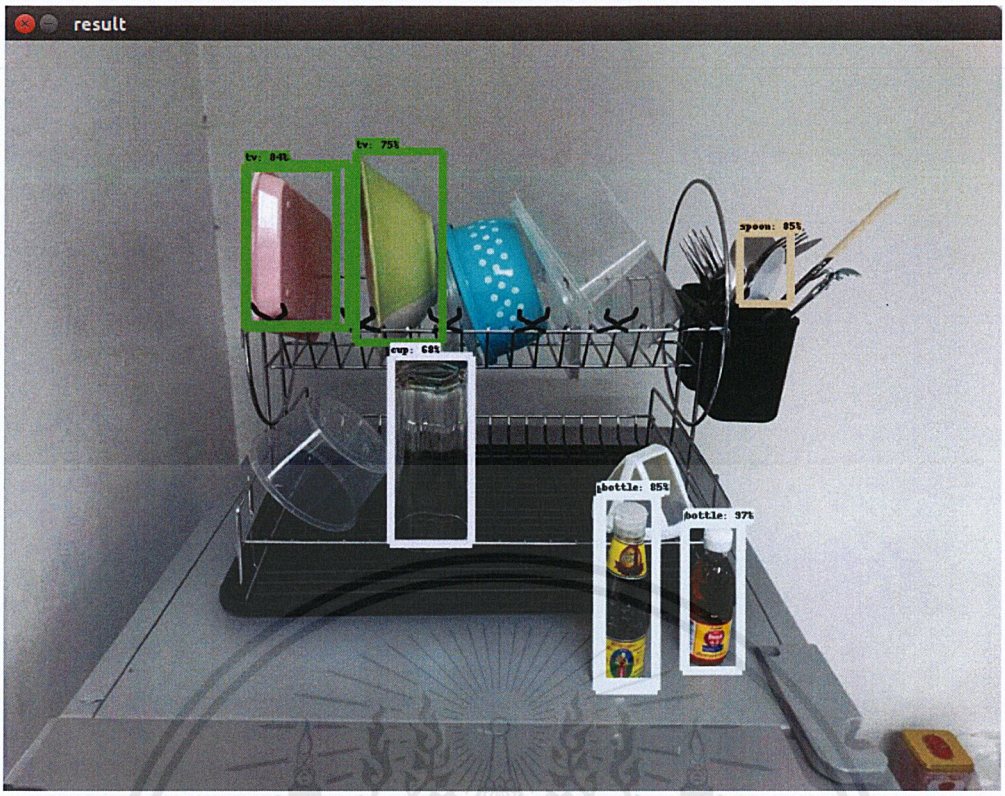


Figure B-7. Classification result using Faster R-CNN ResNet-101 model.

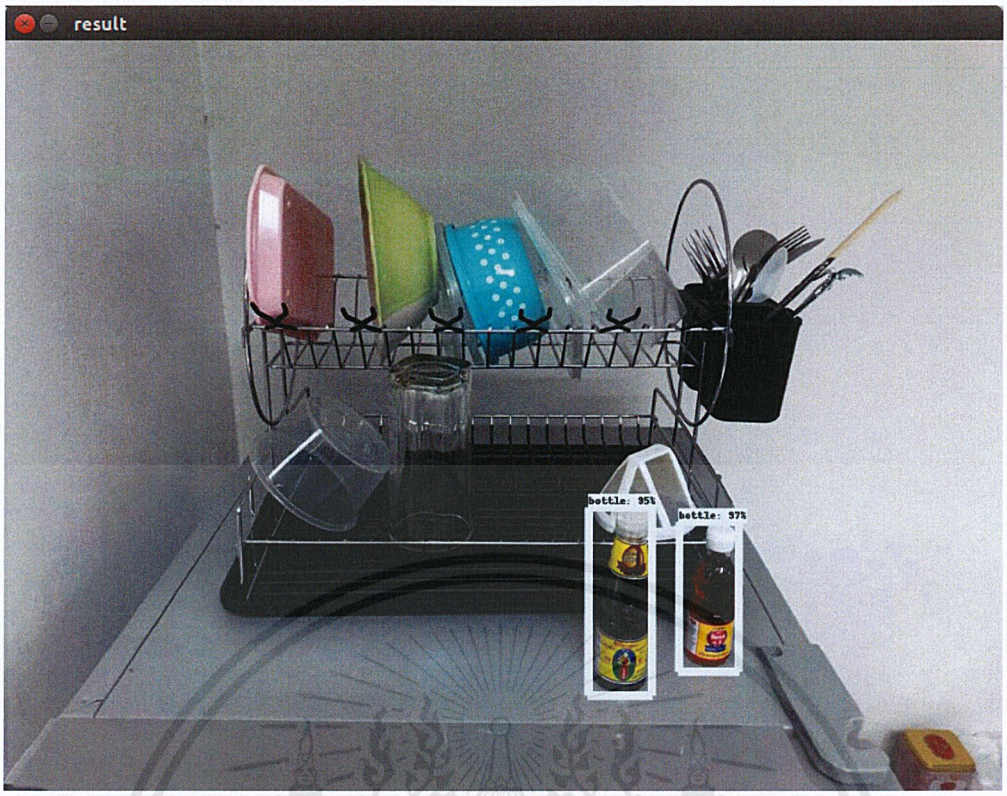


Figure B-8. Classification result using Faster R-CNN Inception v2 model.

From Figures B-5 to B-8, SSD MobileNet V1 model detected 1 object.

TABLE B.3
CLASSIFICATION RESULT FROM SSD MOBILENET V1 MODEL.

Bounding box	Color	Label	Probability (%)
1	blue	airplane	56

Faster R-CNN ResNet-101 model detected six objects.

Table B.4
Classification result from Faster R-CNN ResNet-101 model.

Bounding box	Color	Label	Probability (%)
1	green	tv	84
2	green	tv	75
3	pink	cup	68
4	white	bottle	85
5	white	bottle	97
6	beige	spoon	85

Faster R-CNN Inception v2 model detected two objects.

Table B.5
Classification result from Faster R-CNN Inception v2 model.

Bounding box	Color	Label	Probability (%)
1	white	bottle	95
2	white	bottle	97

B.3 Image 3

Figure B-9. Input from camera stream.

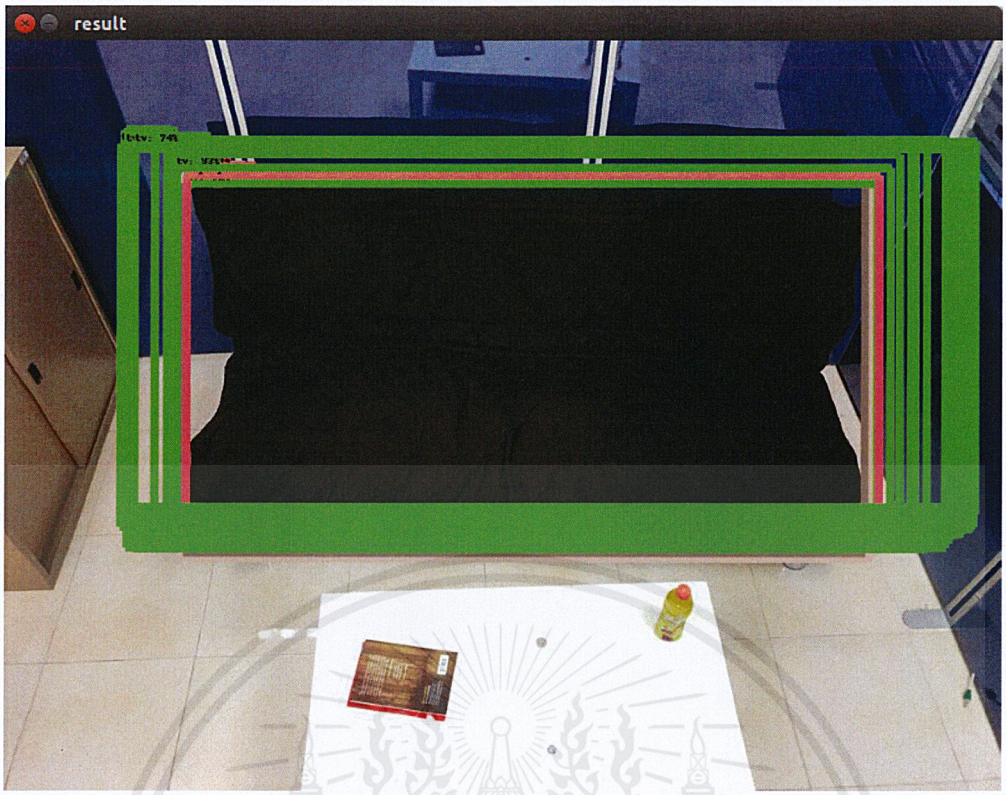


Figure B-10. Classification result using SSD MobileNet V1 model.

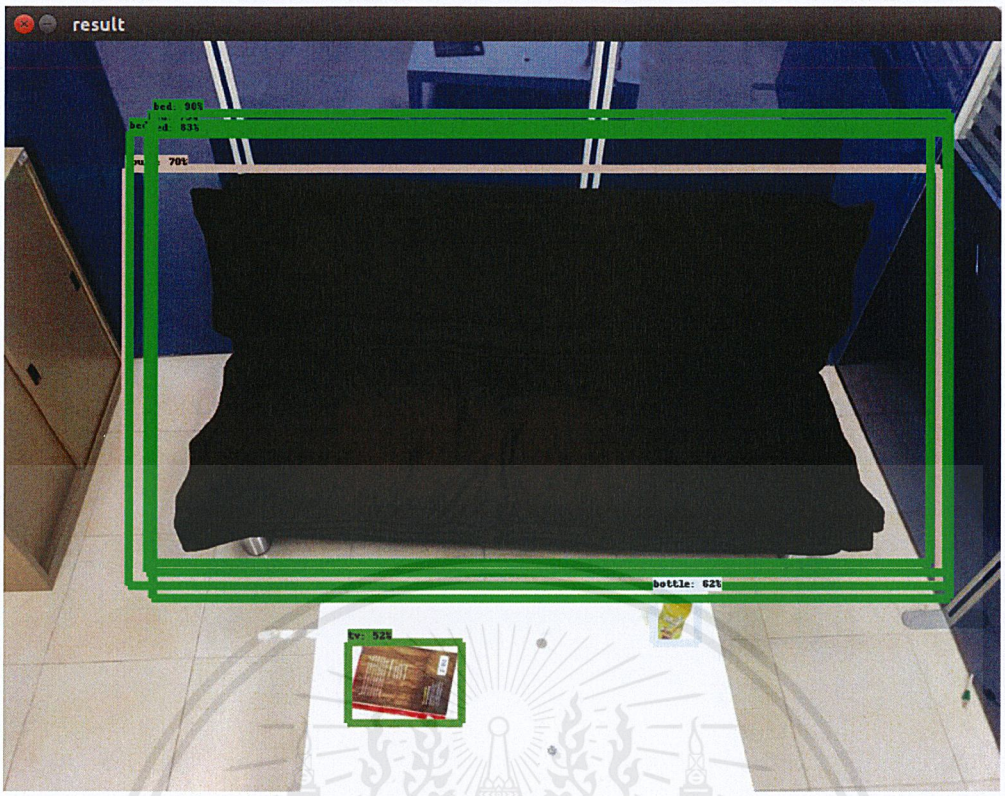


Figure B-11. Classification result using Faster R-CNN ResNet-101 model.

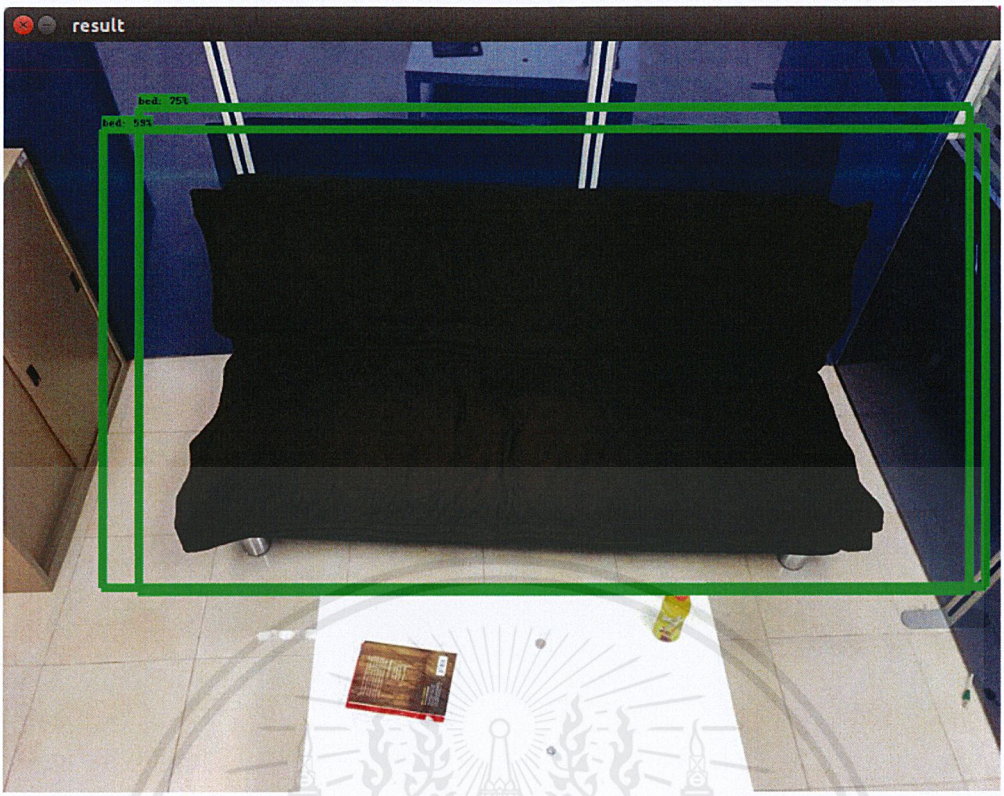


Figure B-12. Classification result using Faster R-CNN Inception v2 model.

From Figures B-9 to B-12, SSD MobileNet V1 model detected one object.

Table B.6
Classification result from SSD MobileNet V1 model.

Bounding box	Color	Label	Probability (%)
1	green	tv	74

Faster R-CNN ResNet-101 model detected three objects.

Table B.7
Classification result from Faster R-CNN ResNet-101 model.

Bounding box	Color	Label	Probability (%)
1	light green	bed	90
2	green	tv	52
3	white	bottle	62

Faster R-CNN Inception v2 model detected one object.

TABLE B.8
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	light green	bed	75

B.4 Image 4

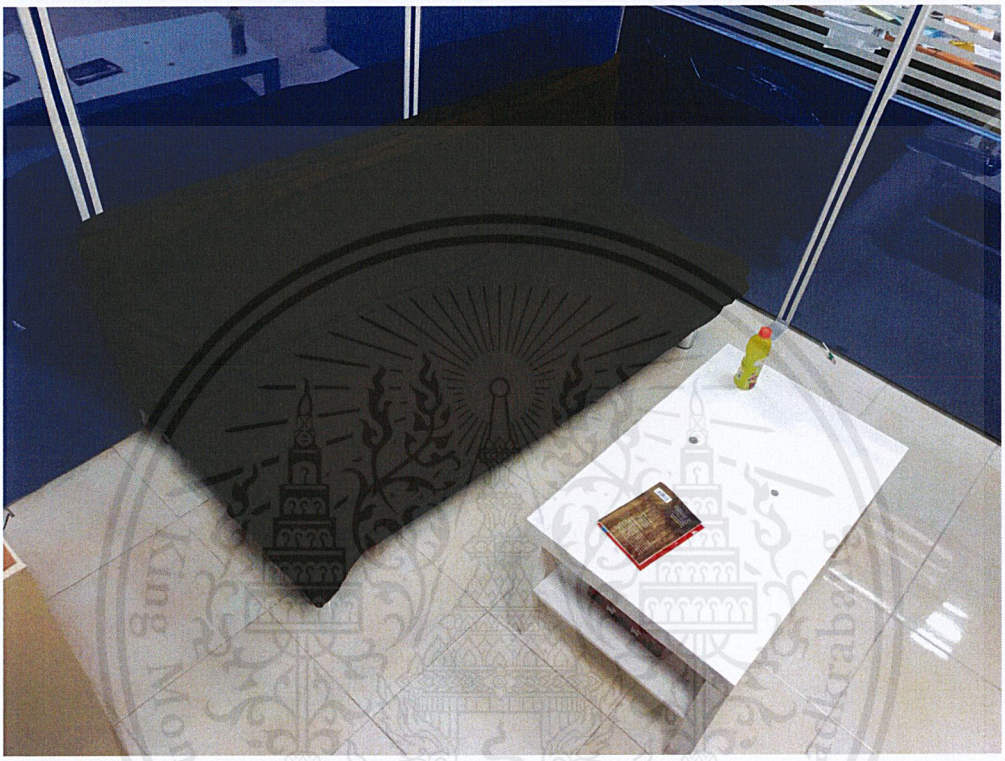


Figure B-13. Input from camera stream.

Figure B-14. Classification result using SSD MobileNet V1 model.

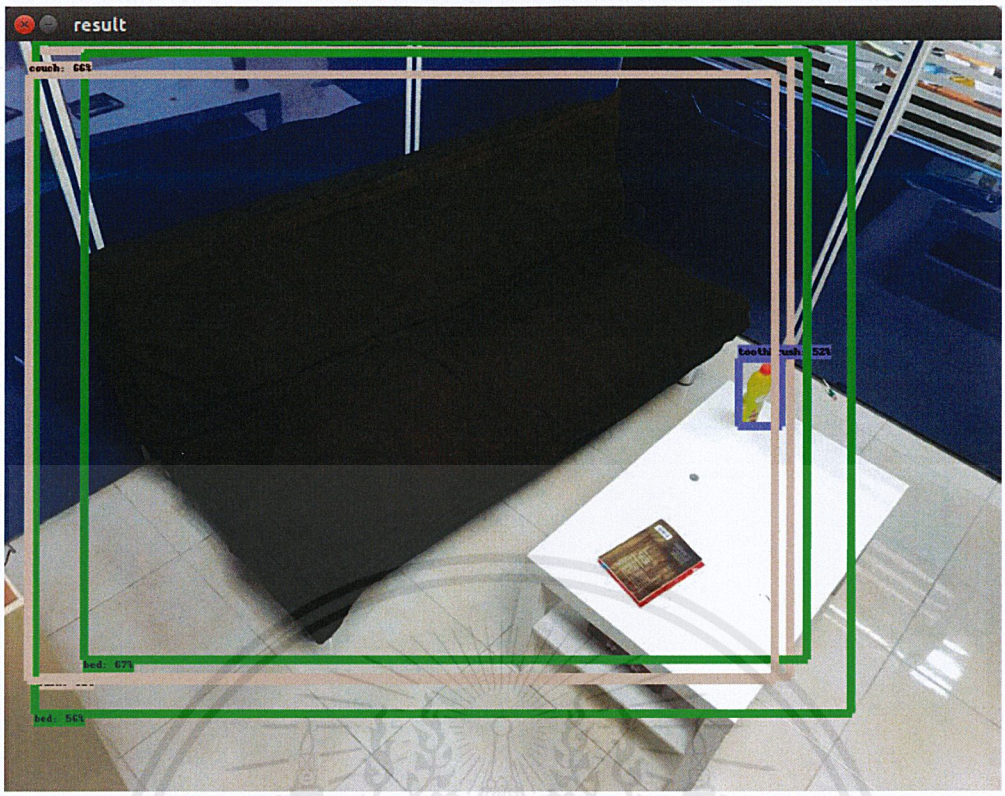


Figure B-15. Classification result using Faster R-CNN ResNet-101 model.

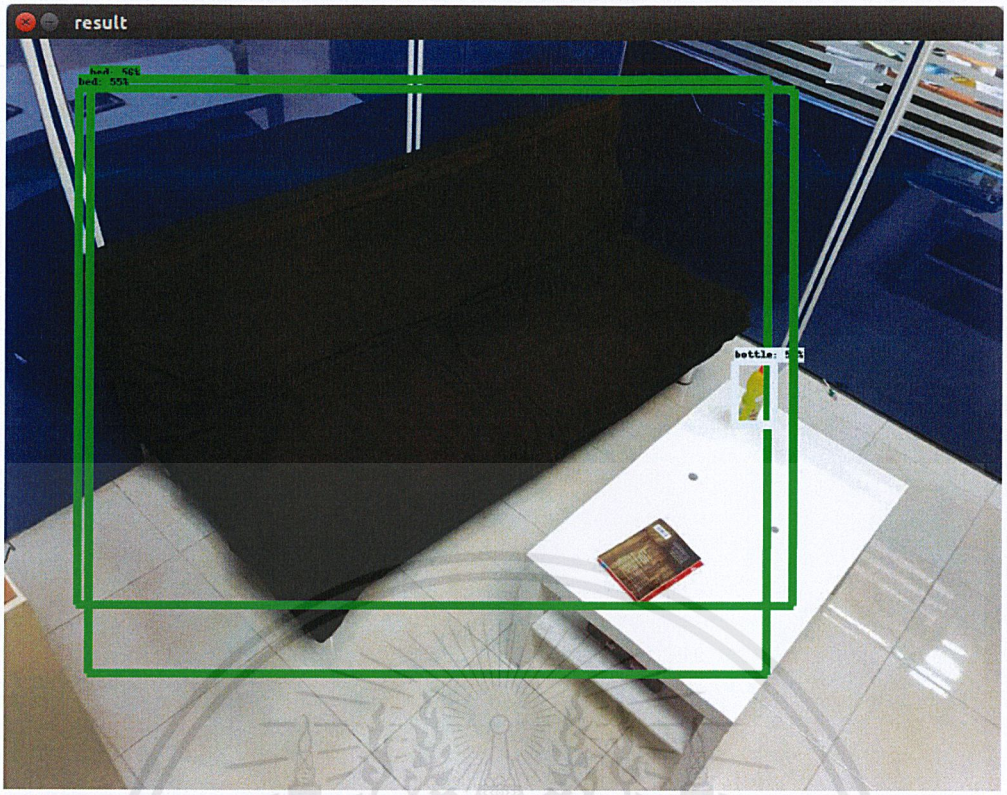


Figure B-16. Classification result using Faster R-CNN Inception v2 model.

From Figures B-13 to B-16, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected three objects.

TABLE B.9
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	light green	bed	67
2	beige	couch	66
3	purple	toothbrush	52

Faster R-CNN Inception v2 model detected two objects.

TABLE B.10
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	light green	bed	55
2	white	bottle	56

B.5 Image 5

Figure B-17. Input from camera stream.

Figure B-18. Classification result using SSD MobileNet V1 model.

Figure B-19. Classification result using Faster R-CNN ResNet-101 model.

Figure B-20. Classification result using Faster R-CNN Inception v2 model.

From Figures B-17 to B-20, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected four objects.

TABLE B.11
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	green	refridgerator	88
2	white	bottle	54
3	pink	cup	55
4	blue	clock	87

Faster R-CNN Inception v2 model detected four objects.

TABLE B.12
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	green	bottle	88
2	white	bottle	91
3	white	bottle	57
4	pink	cup	57

B.6 Image 6

Figure B-21. Input from camera stream.

Figure B-22. Classification result using SSD MobileNet V1 model.

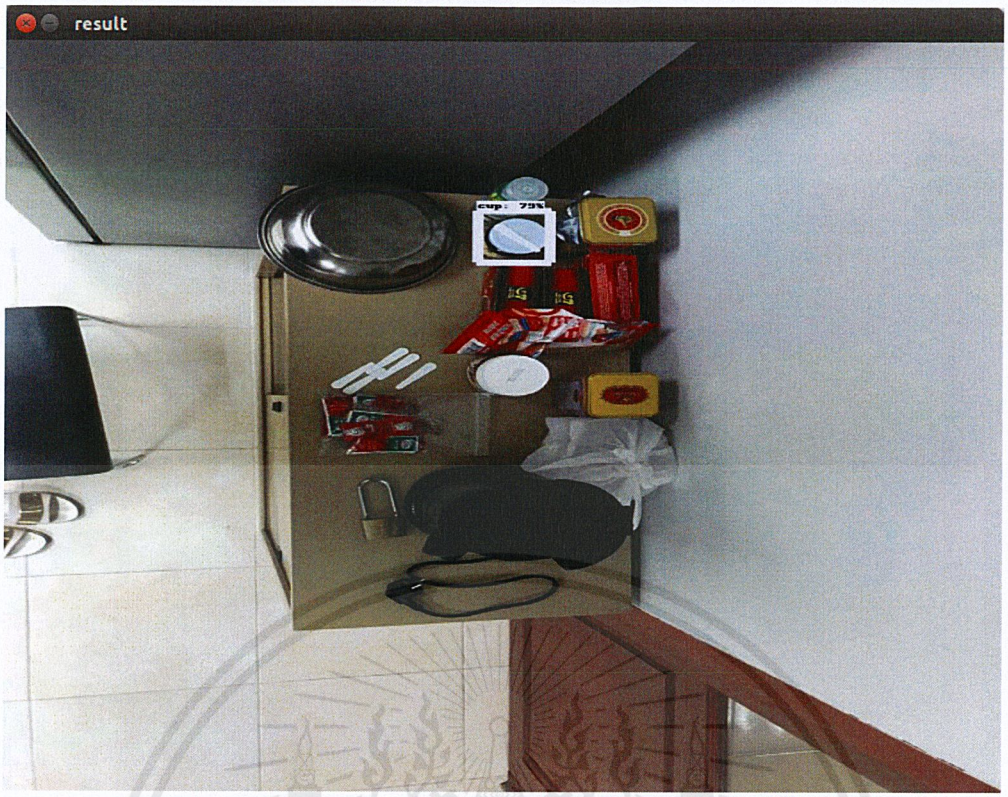


Figure B-23. Classification result using Faster R-CNN ResNet-101 model.

Figure B-24. Classification result using Faster R-CNN Inception v2 model.

From Figures B-21 to B-24, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected one object.

Table B.13
Classification result from Faster R-CNN ResNet-101 model.

Bounding box	Color	Label	Probability (%)
1	pink	cup	79

Faster R-CNN Inception v2 model detected 1 object.

Table B.14
Classification result from Faster R-CNN Inception v2 model.

Bounding box	Color	Label	Probability (%)
1	green	refridgerator	50

B.7 Image 7

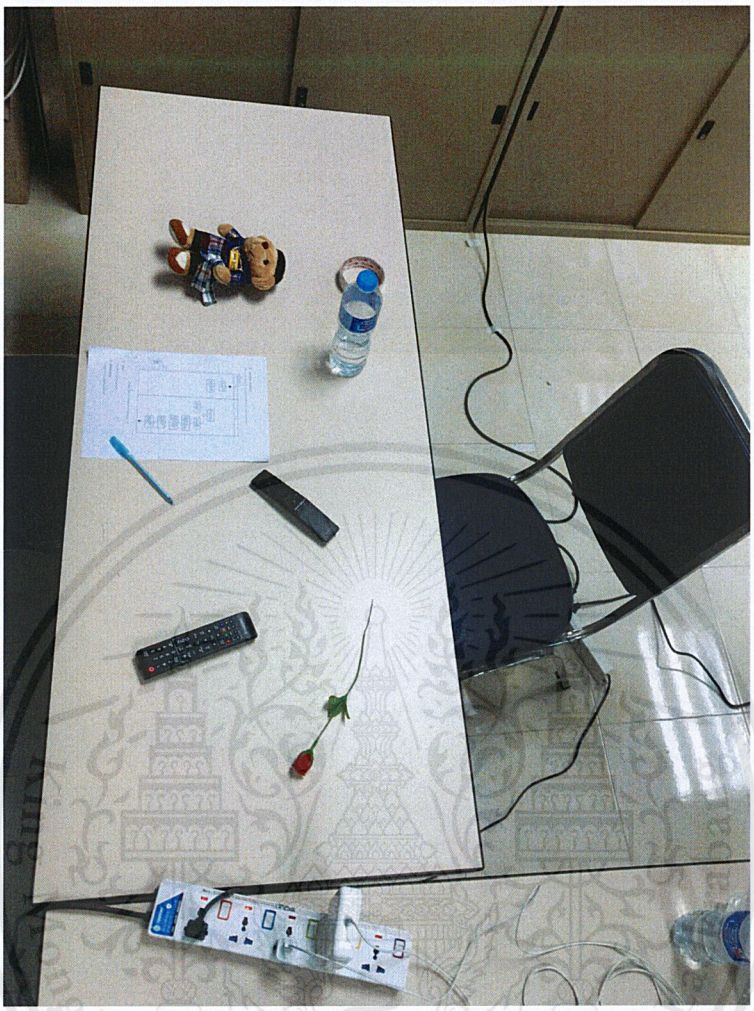


Figure B-25. Input from camera stream.

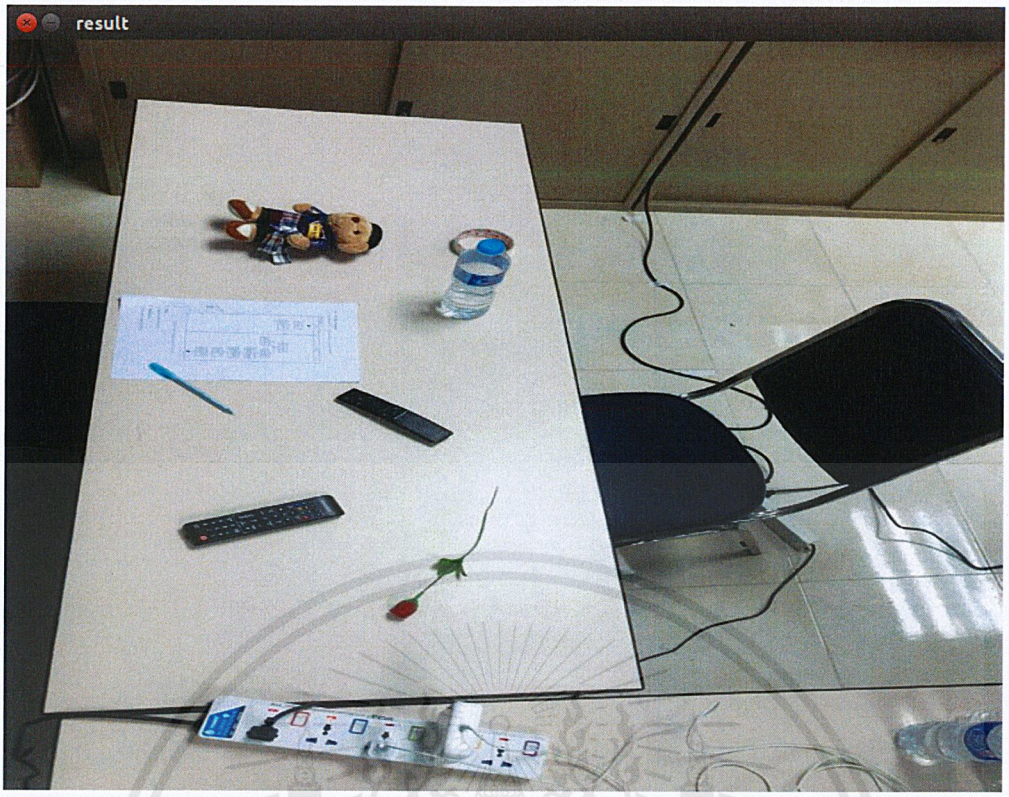


Figure B-26. Classification result using SSD MobileNet V1 model.

Figure B-27. Classification result using Faster R-CNN ResNet-101 model.

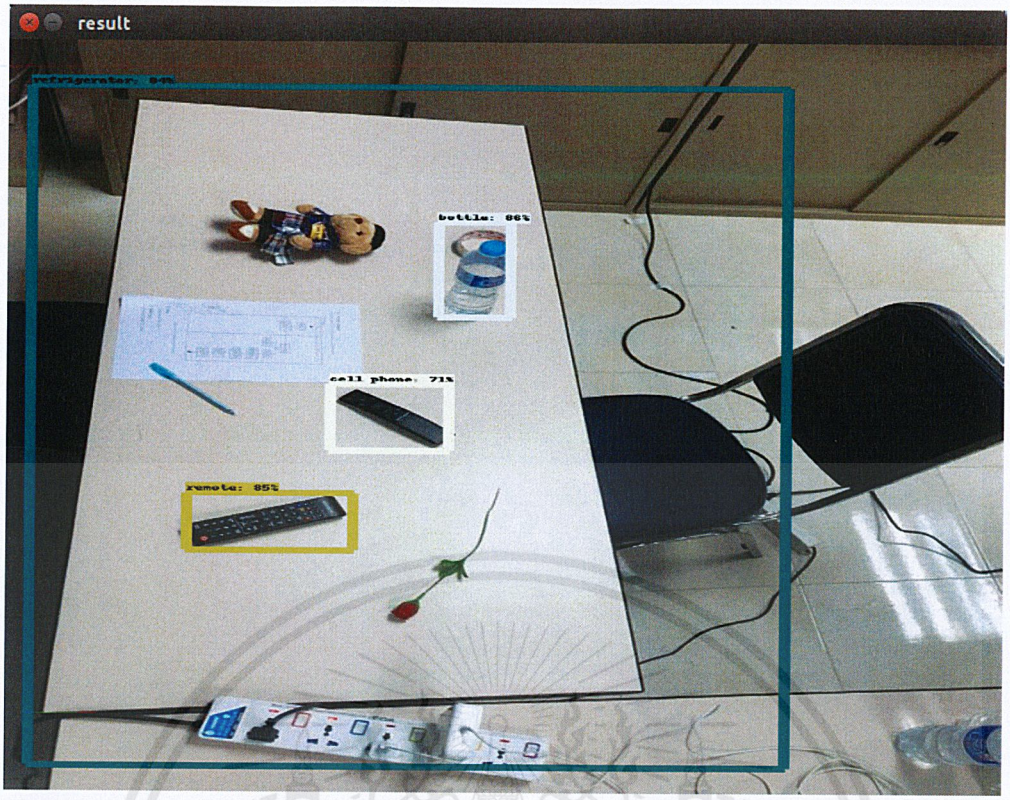


Figure B-28. Classification result using Faster R-CNN Inception v2 model.

From Figures B-25 to B-28, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected five objects.

TABLE B.15
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	green	refridgerator	91
2	yellow	remote	80
3	yellow	remote	78
4	white	bottle	74
5	white	bottle	54

Faster R-CNN Inception v2 model detected four objects.

TABLE B.16
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	green	refridgerator	84
2	yellow	remote	85
3	light yellow	cell phone	71
4	white	bottle	86

B.8 Image 8

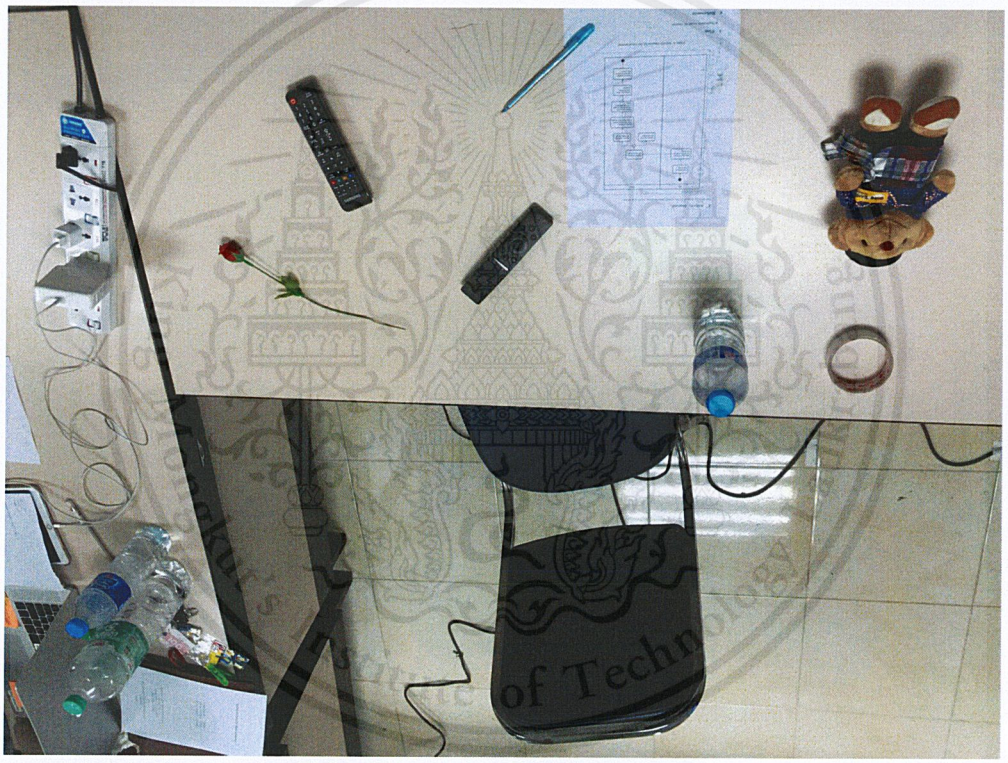


Figure B-29. Input from camera stream.

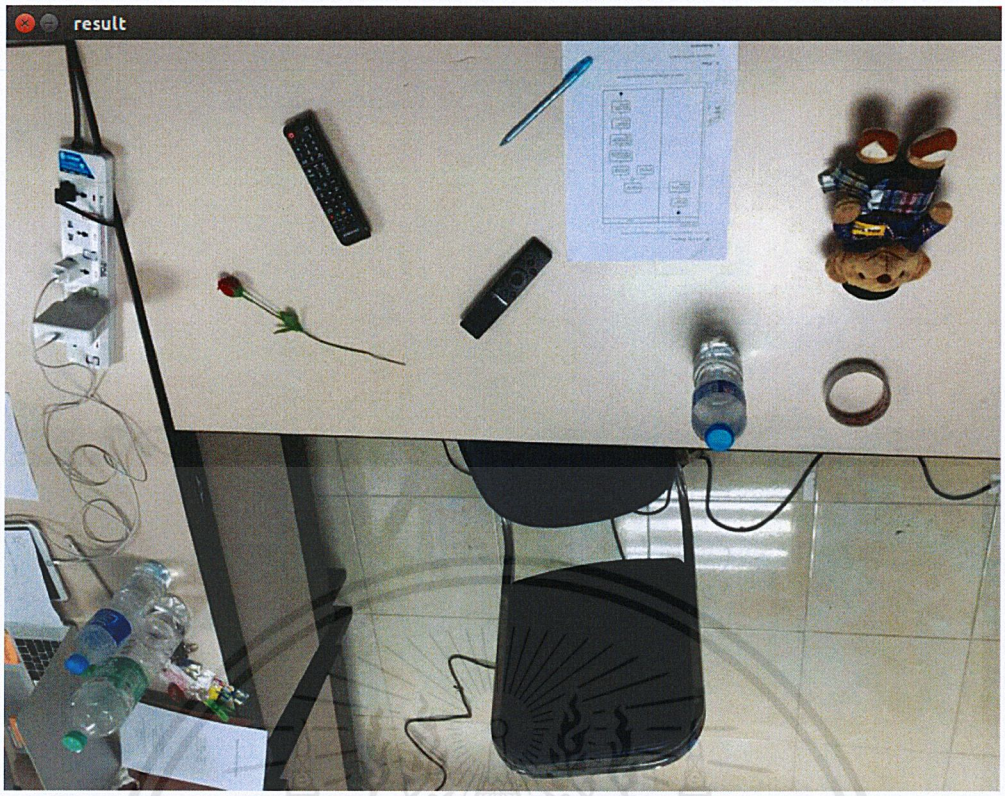


Figure B-30. Classification result using SSD MobileNet V1 model.

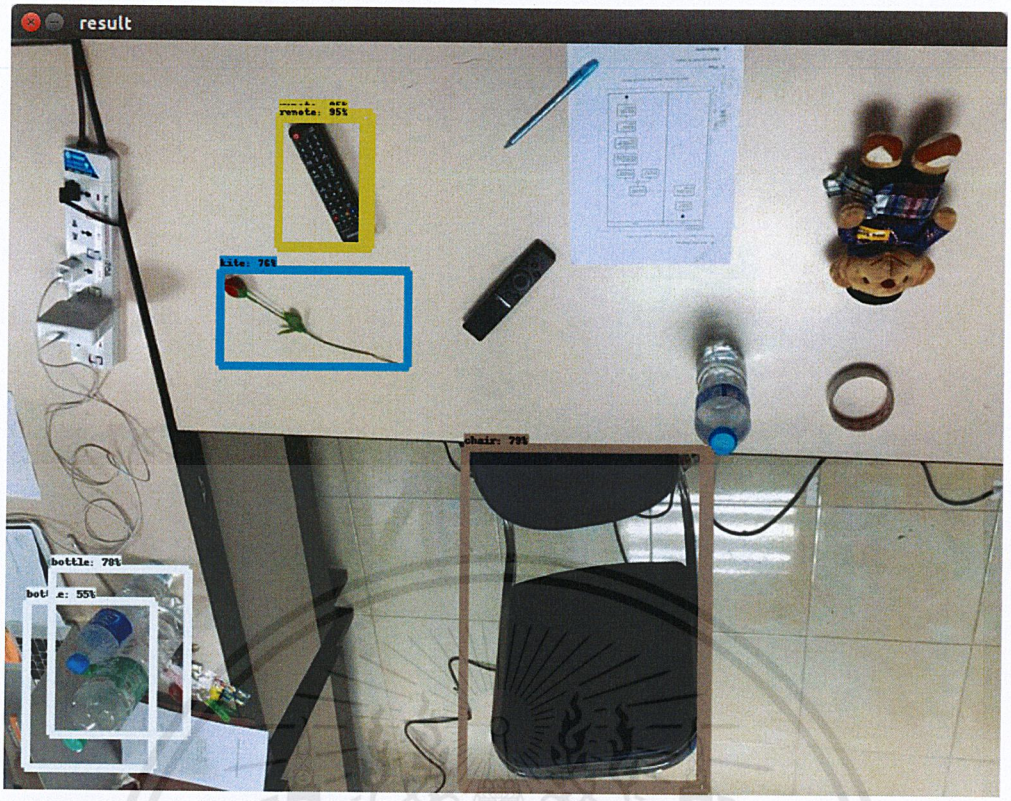


Figure B-32. Classification result using Faster R-CNN Inception v2 model.

From Figures B-29 to B-32, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected two objects.

TABLE B.17
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	yellow	remote	50
2	brown	chair	97

Faster R-CNN Inception v2 model detected five objects.

Table B.18
Classification result from Faster R-CNN Inception v2 model.

Bounding box	Color	Label	Probability (%)
1	white	bottle	55
2	white	bottle	78
3	blue	bottle	76
4	yellow	cup	95
5	brown	chair	79

B.9 Image 9

Figure B-33. Input from camera stream.

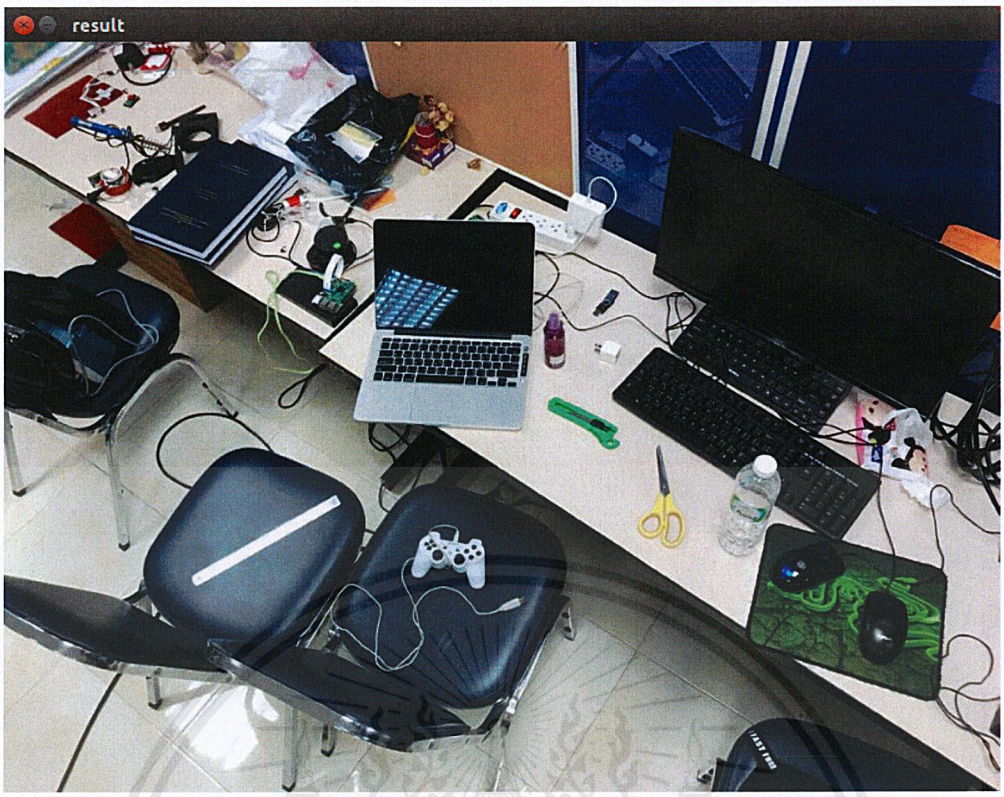


Figure B-34. Classification result using SSD MobileNet V1 model.

Figure B-35. Classification result using Faster R-CNN ResNet-101 model.

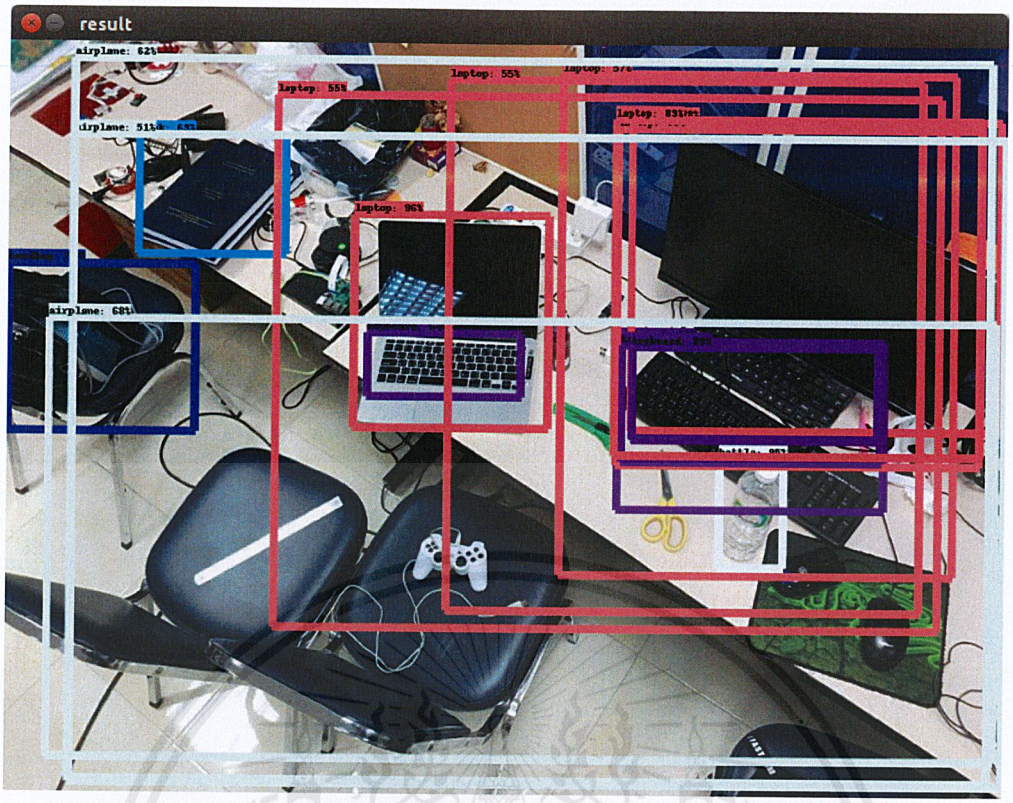


Figure B-36. Classification result using Faster R-CNN Inception v2 model.

From Figures B-33 to B-36, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected eight objects.

TABLE B.19
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	light cream	motorcycle	51
2	brown	chair	98
3	blue	book	71
4	coral	laptop	57
5	purple	keyboard	99
6	green	tv	92
7	white	bottle	95
8	light green	mouse	76

Faster R-CNN Inception v2 model detected eleven objects.

TABLE B.20
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	navy	handbag	57
2	light blue	airplane	68
3	light blue	airplane	62
4	blue	book	69
5	coral	laptop	55
6	coral	laptop	96
7	purple	keyboard	64
8	coral	laptop	55
9	coral	laptop	89
10	purple	keyboard	89
11	white	bottle	95

B.10 Image 10

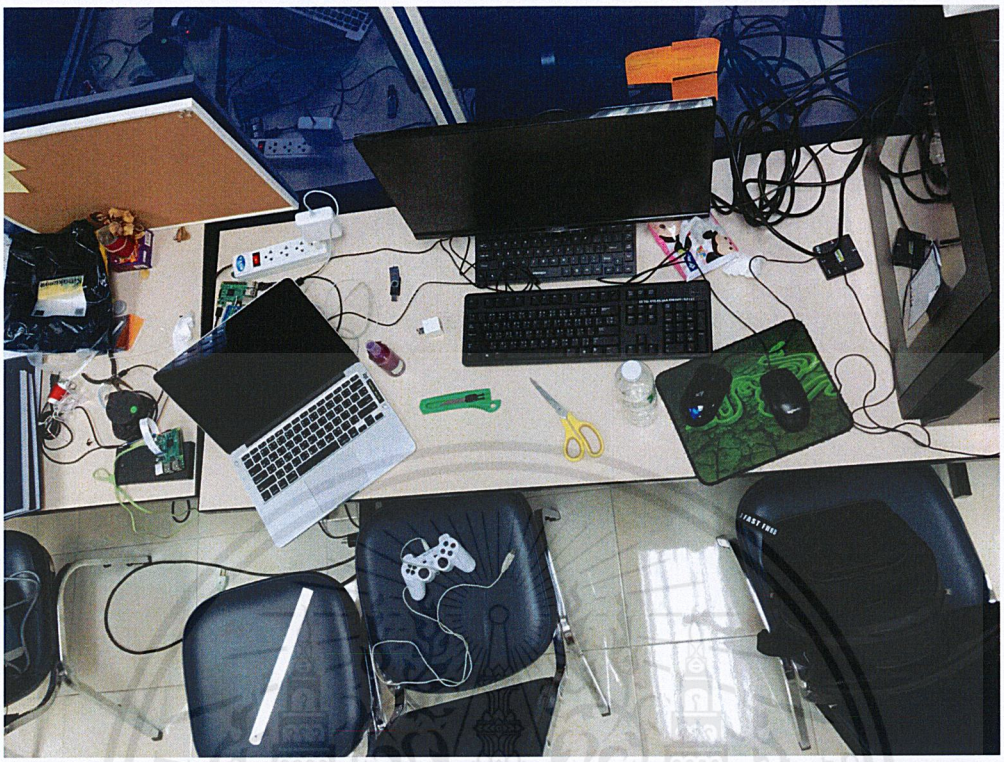


Figure B-37. Input from camera stream.

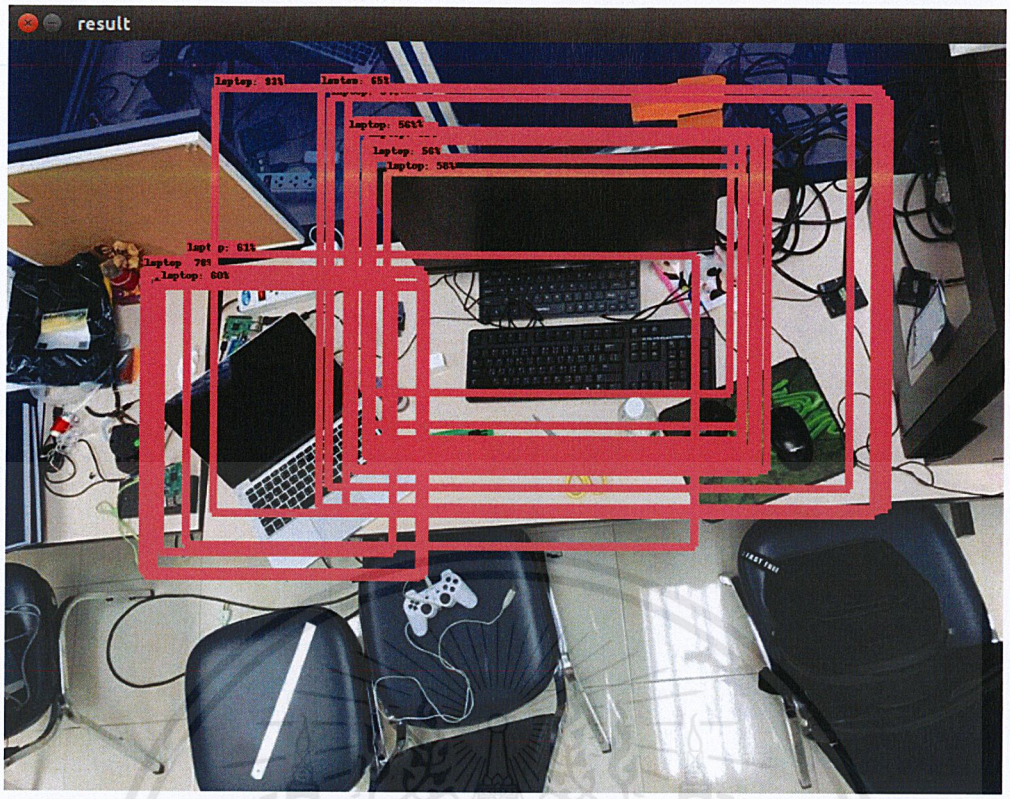


Figure B-38. Classification result using SSD MobileNet V1 model.

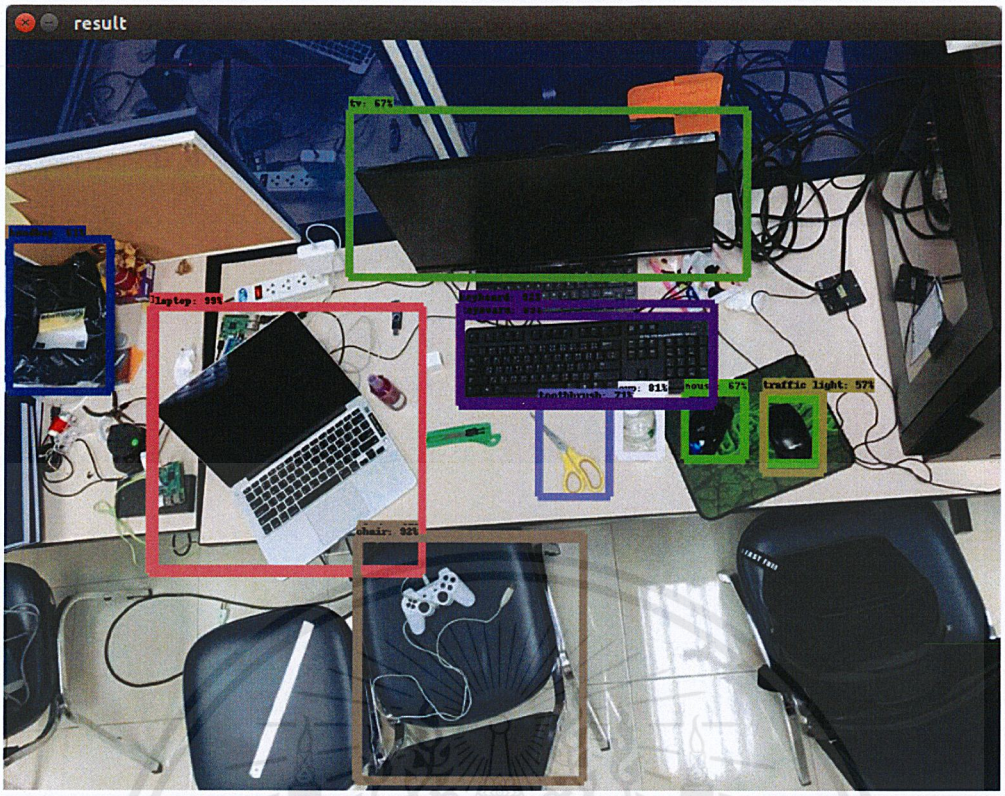


Figure B-39. Classification result using Faster R-CNN ResNet-101 model.

Figure B-40. Classification result using Faster R-CNN Inception v2 model.

From Figures B-37 to B-40, SSD MobileNet V1 model detected two objects.

TABLE B.21
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	coral	laptop	78
2	coral	laptop	65

Faster R-CNN ResNet-101 model detected nine objects.

TABLE B.22
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	navy	handbag	61
2	coral	laptop	99
3	green	tv	67
4	brown	chair	92
5	purple	keyboard	92
6	light purple	toothbrush	71
7	white	cup	81
8	light green	mouse	67
9	olive	traffic light	57

Faster R-CNN Inception v2 model detected six objects.

TABLE B.23
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	coral	laptop	57
2	brown	chair	59
3	brown	chair	74
4	brown	chsir	53
5	purple	keyboard	84
6	blue	scissors	63

B.11 Image 11

Figure B-41. Input from camera stream

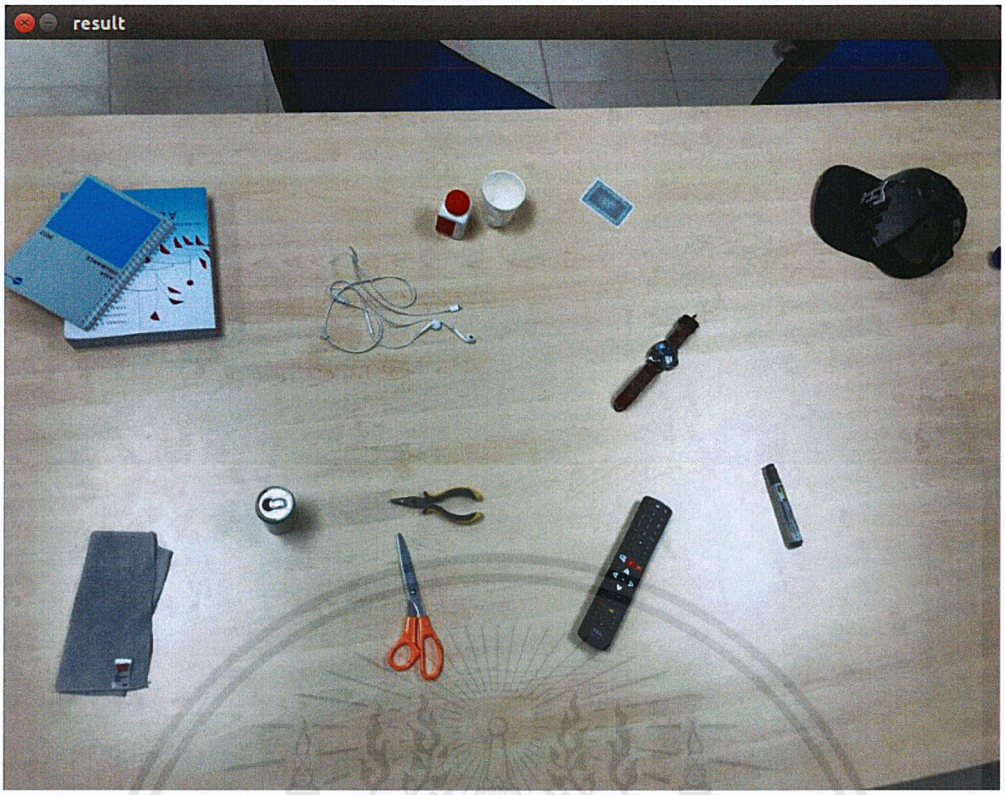


Figure B-42. Classification result using SSD MobileNet V1 model

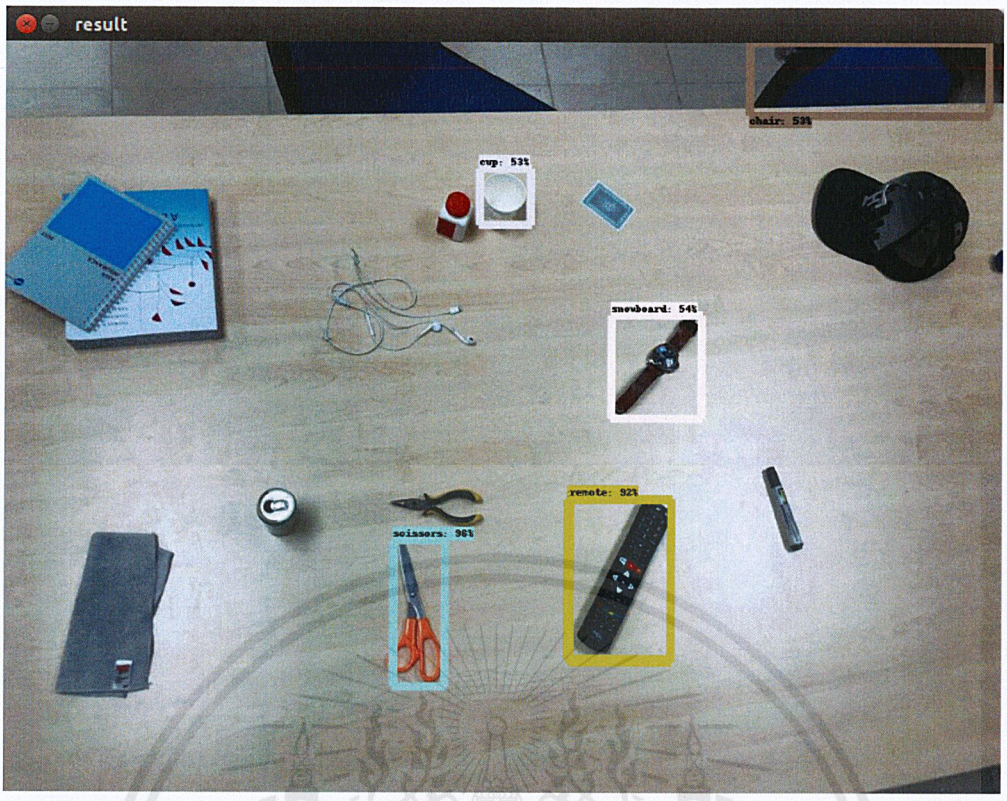


Figure B-43. Classification result using Faster R-CNN ResNet-101 model.

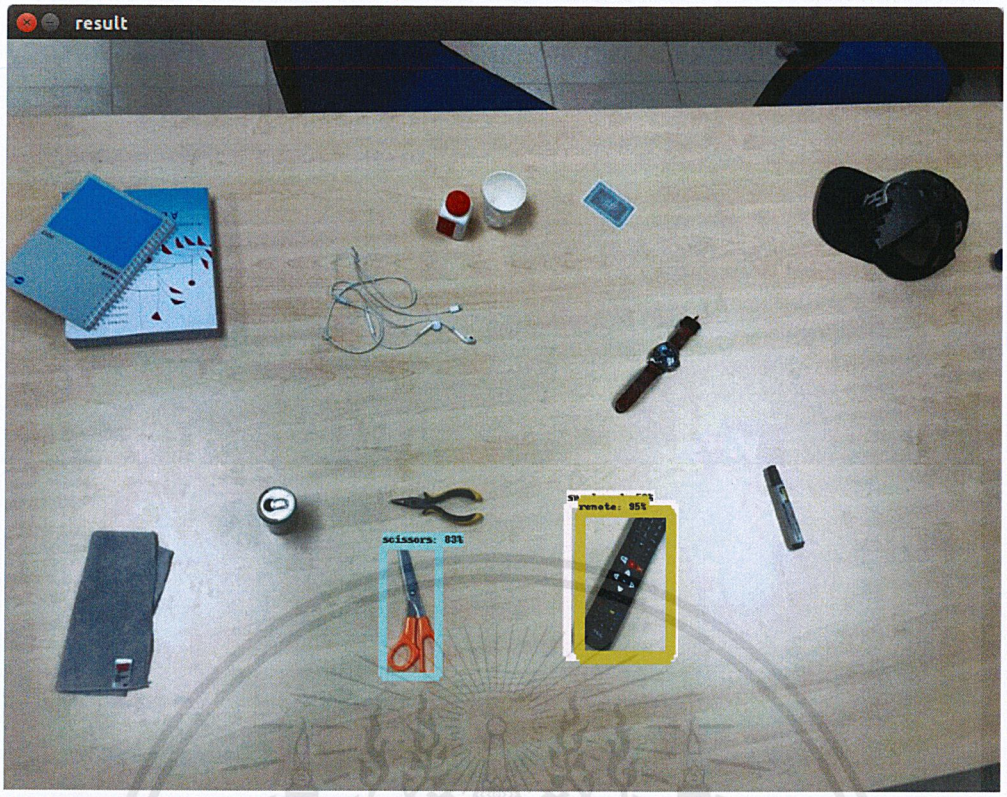


Figure B-44. Classification result using Faster R-CNN Inception v2 model.

From Figures B-41 to B-44, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected five objects.

TABLE B.24
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	blue	scissors	96
2	pink	cup	53
3	green	remote	92
4	light cream	snowboard	54
5	brown	chair	53

Faster R-CNN Inception v2 model detected two objects.

TABLE B.25
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	blue	scissors	83
2	green	remote	95

B.12 Image 12

Figure B-45. Input from camera stream.

Figure B-46. Classification result using SSD MobileNet V1 model.

Figure B-47. Classification result using Faster R-CNN ResNet-101 model.

Figure B-48. Classification result using Faster R-CNN Inception v2 model.

From Figures B-45 to B-48, SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected eleven objects.

TABLE B.26
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	white	dining table	75
2	blue	book	56
3	brown	chair	56
4	blue	book	73
5	pink	cup	87
6	brown	chair	83
7	lime	person	81
8	blue	scissors	97
9	green	remote	52
10	green	remote	93
11	bottle	white	86

Faster R-CNN Inception v2 model detected six objects.

TABLE B.27
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	blue	book	82
2	brown	chair	70
3	brown	chair	81
4	brown	chair	67
5	green	remote	78
6	pink	cup	68

B.13 Image 13

Figure B-49. Input from camera stream.

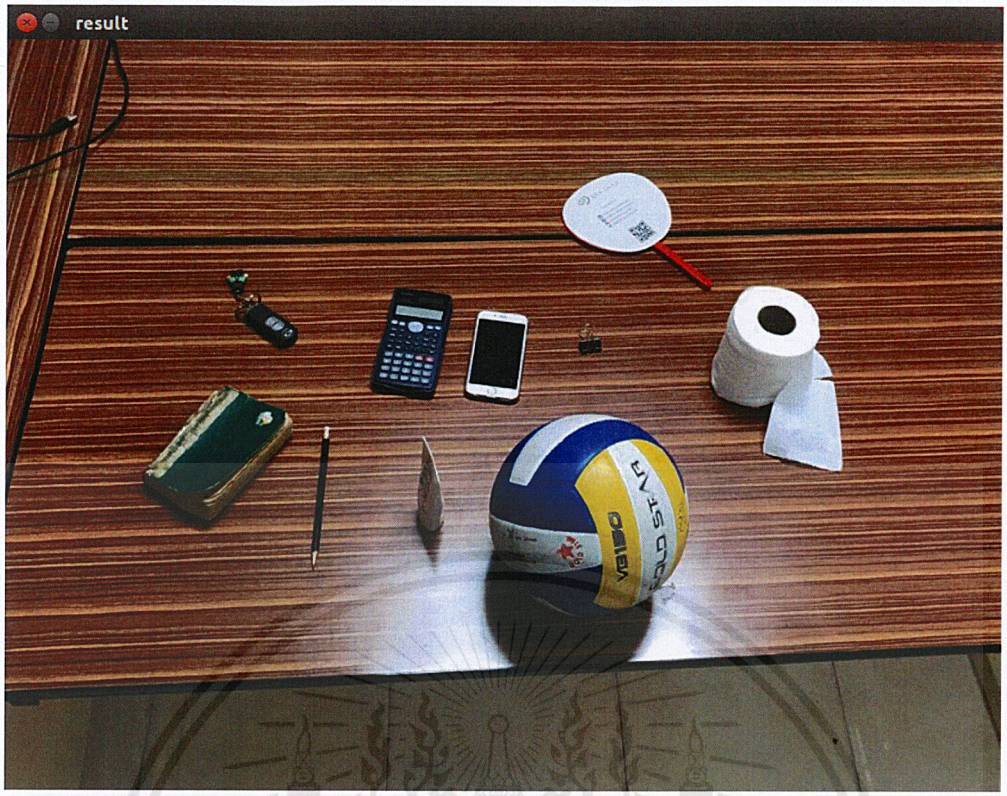


Figure B-50. Classification result using SSD MobileNet V1 model.

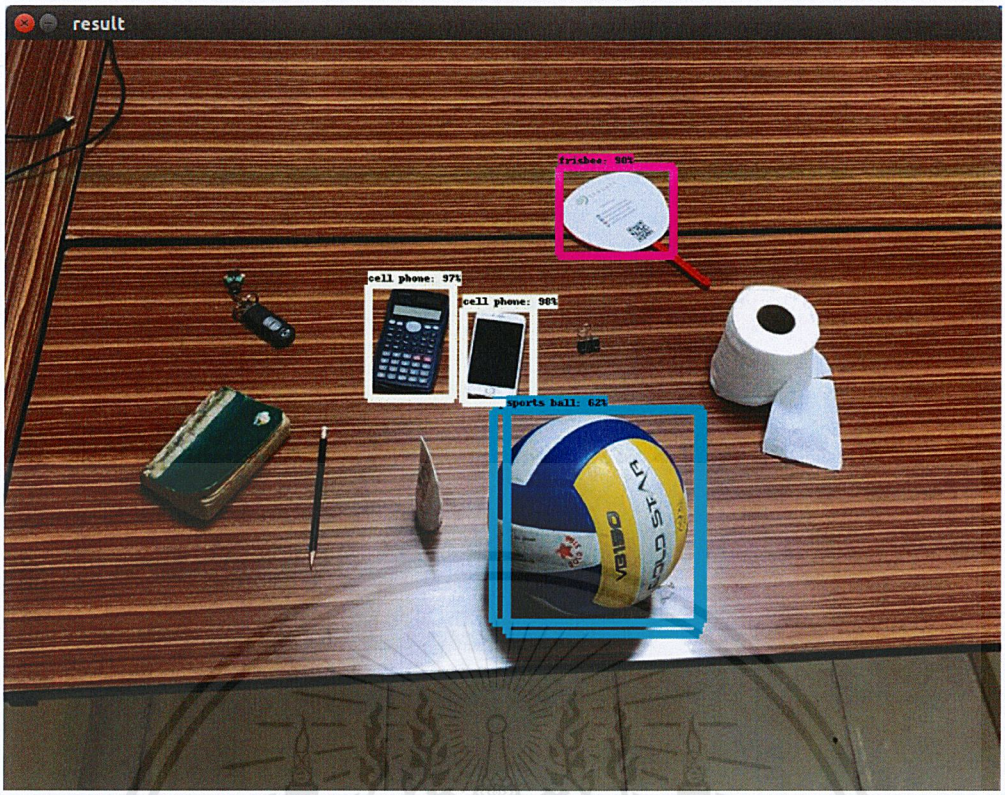


Figure B-51. Classification result using Faster R-CNN ResNet-101 model.

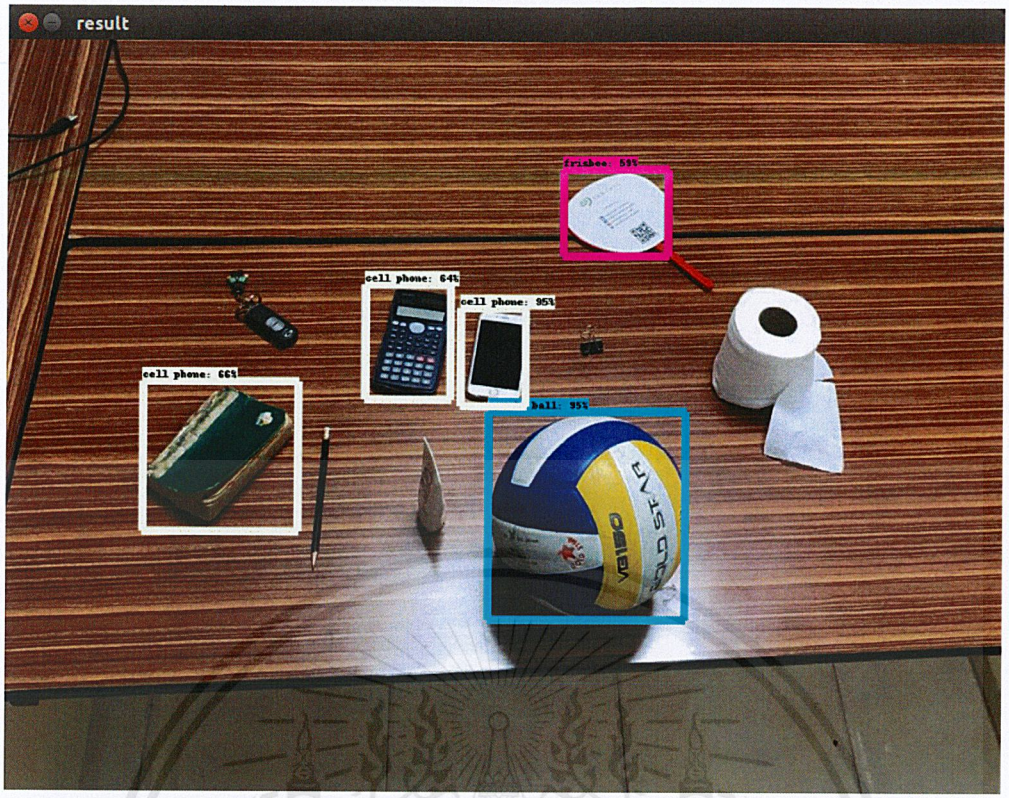


Figure B-52. Classification result using Faster R-CNN Inception v2 model.

From Figures B-49 to B-52 , SSD MobileNet V1 model detected no object. Faster R-CNN ResNet-101 model detected four objects.

TABLE B.28
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	light cream	cell phone	97
2	light cream	cell phone	98
3	blue	sports ball	62
4	magenta	frisbee	90

Faster R-CNN Inception v2 model detected five objects.

TABLE B.29
CLASSIFICATION RESULT FROM FASTER R-CNN INCEPTION V2 MODEL.

Bounding box	Color	Label	Probability (%)
1	light cream	cell phone	66
2	light cream	cell phone	64
3	light cream	cell phone	95
4	blue	sports ball	95
5	magenta	frisbee	59

B.14 Image 14

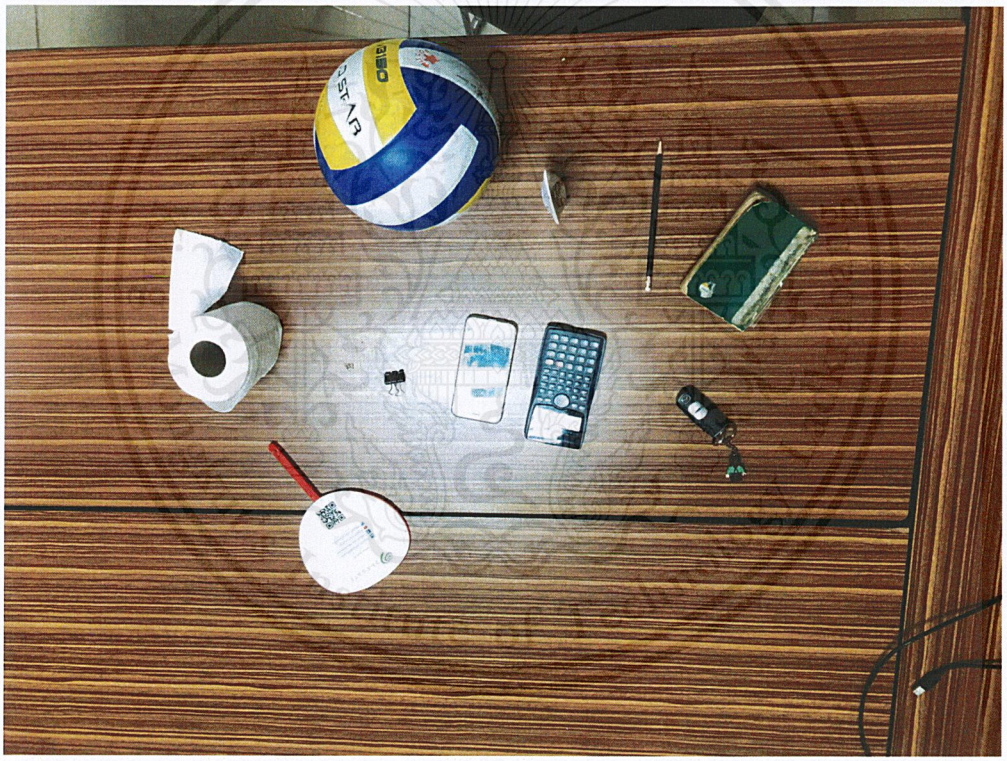


Figure B-53. Input from camera stream.

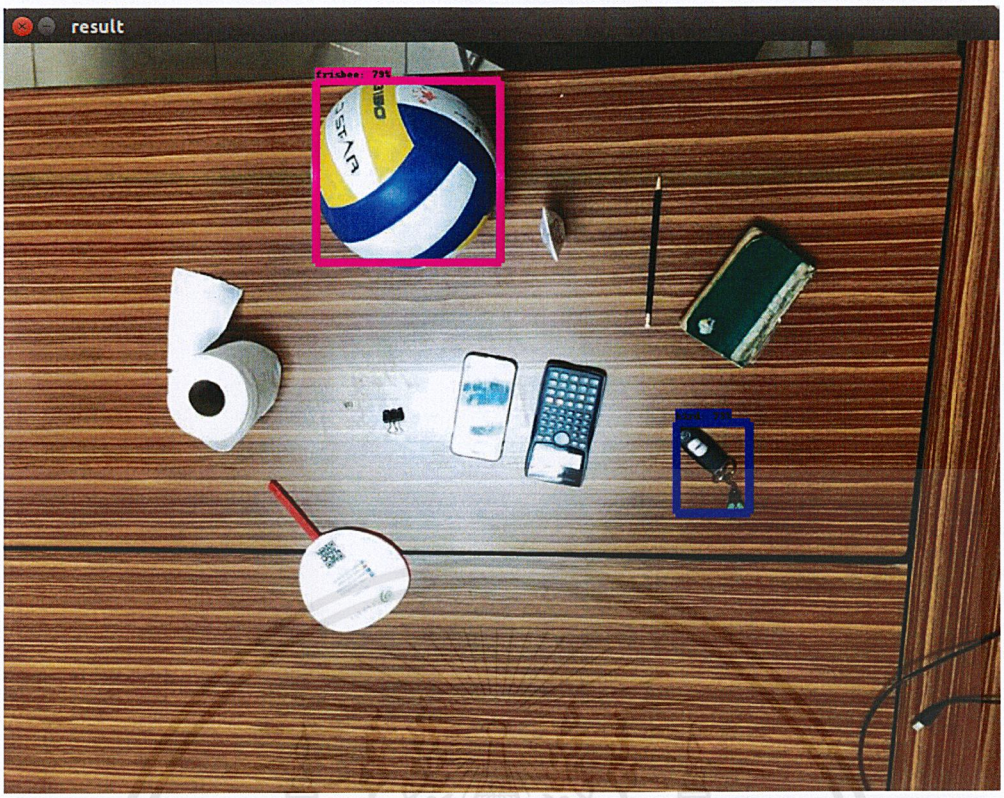


Figure B-54. Classification result using SSD MobileNet V1 model.

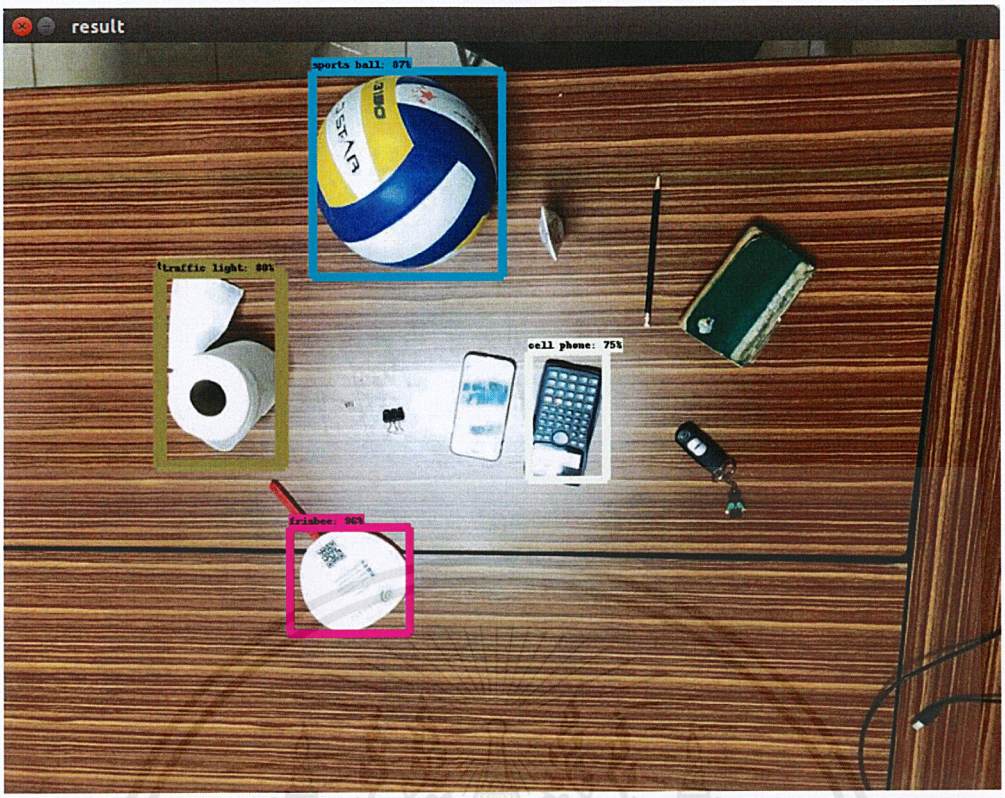


Figure B-55. Classification result using Faster R-CNN ResNet-101 model.

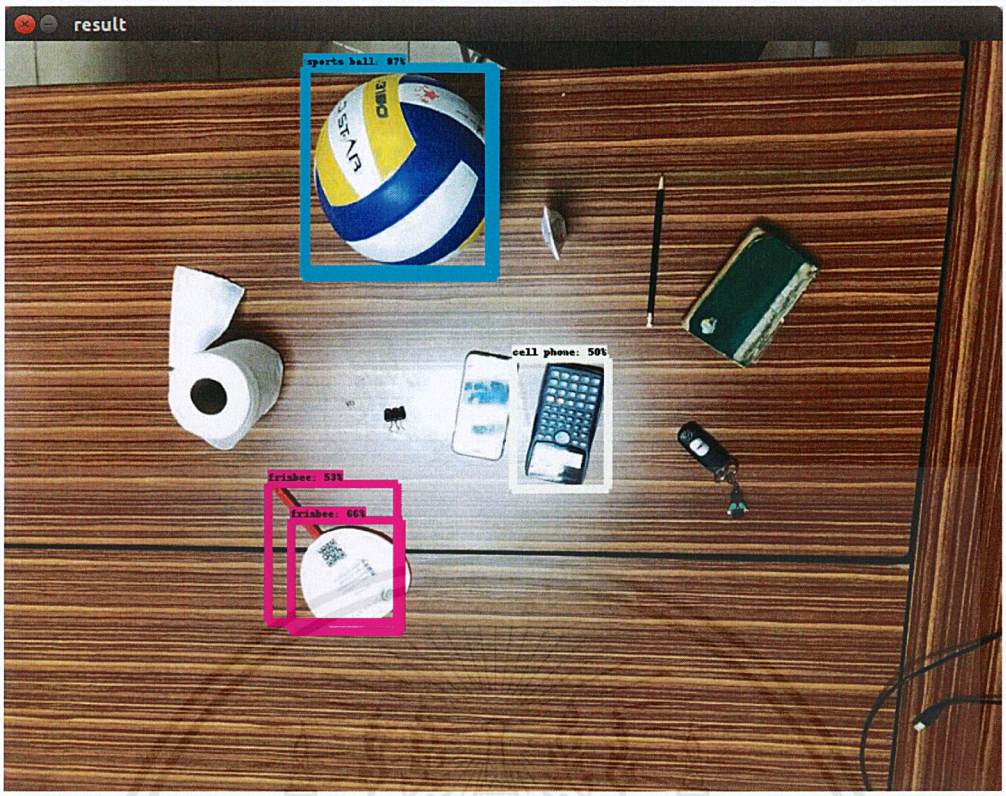


Figure B-56. Classification result using Faster R-CNN Inception v2 model.

From Figures B-53 to B-56, SSD MobileNet V1 model detected two objects.

TABLE B.30
CLASSIFICATION RESULT FROM FASTER R-CNN RESNET-101 MODEL.

Bounding box	Color	Label	Probability (%)
1	magenta	frisbee	79
2	navy	bird	73

Faster R-CNN ResNet-101 model detected four objects.

Table B.31
Classification result from Faster R-CNN ResNet-101 model.

Bounding box	Color	Label	Probability (%)
1	olive	traffic light	80
2	magenta	frisbee	96
3	blue	sports ball	87
4	light cream	cell phone	75

Faster R-CNN Inception v2 model detected three objects.

Table B.32
Classification result from Faster R-CNN Inception v2 model.

Bounding box	Color	Label	Probability (%)
1	magenta	frisbee	66
2	blue	sports ball	87
3	light cream	cell phone	50

Bibliography

[1] A. Kitanov and K. Lenac, et al., "Fast Planar Surface 3D SLAM Using LIDAR", *Elsevier B.V.*, pp.198-218, 2017.

[2] A. Bachrach, N. Roy, and R. He., "Autonomous Flight in Unknown Indoor Environments", *International Journal of Micro Air Vehicles*, vol.1, no.4, pp.217-228, 2009, url=https://doi.org/10.1260/175682909790291492.

[3] A. Farhadi and J. Redmon., "YOLO9000: Better, Faster, Stronger", *arXiv preprint arXiv:1612.08242*, pp.1-8, 2016.

[4] C. Novel and O. Manuel, et al., "Autonomous 2D SLAM and 3D Mapping of An Environment Using a Single 2D LIDAR and ROS", *Conference: 2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, 2017.

[5] C. Szegedy, et al., "Going Deeper With Convolutions", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.1-9, 2015.

[6] C. Szegedy, et al., "Rethinking the Inception Architecture for Computer Vision", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.2818-2826, 2016.

[7] D. Estler., "Path Planning and Optimization on SLAM-Based Maps", *Annals of Mathematics and Artificial Intelligence*, 2016.

[8] E. Michael, L. Benson, and V. Régis., "Comparison Of Indoor Robot Localization Techniques In The Absence of GPS", *Proceedings of SPIE - The International Society for Optical Engineering*, 2010.

[9] E. Galceran and M. Carreras., "A Survey On Coverage Path Planning For Robotics", *Robotics and Autonomous Systems*, vol.61, pp.1258-1276, 2013.

[10] E. Rimon and Y. Gabriely., "Spanning-Tree Based Coverage of Continuous Areas by a Mobile Robot", *Annals of Mathematics and Artificial Intelligence*, vol.31, pp.77-98, 2001.

[11] F.N. Landola, et al., "SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <0.5 MB Model Size", *arXiv preprint arXiv:1602.07360*, 2016

[12] F. Wang and J. Cui, et al., "A Comprehensive UAV Indoor Navigation System Based on Vision Optical Flow and Laser FastSLAM", *Acta Automatica Sinica*, vol.39, no.11, pp.1889-1899, 2013, ISSN=1874-1029, url=http://www.sciencedirect.com/science/article/pii/S1874102913600804.

[13] F. Wang and K. Wang, et al., "An efficient UAV navigation solution for confined but partially known indoor environments", *IEEE International Conference on Control and Automation, ICCA*, pp.1351-1356, June 2014, ISBN=978-1-4799-2837-8.

[14] G. A. Borges and M. J. Aldon., "A Split-and-merge Segmentation Algorithm for Line Extraction in 2D Range Images", *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol.1, pp.441-444, 2000, ISSN=1051-4651.

[15] G. Grisetti, et al., "A Tutorial on Graph-Based SLAM", *IEEE Intelligent Transportation Systems Magazine*, vol.2, pp.31-43, 2010.

[16] H. Choset., "Coverage for Robotics - A Survey of Recent Results", *Annals Of Mathematics and Artificial Intelligence*, vol.31, pp.113-126, 2001.

[17] H. Durrant-Whyte and T. Bailey., "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms", *Proceedings of the IEEE*, pp.1-8, 2006.

[18] J. Engel., "Autonomous Camera-based Navigation of a Quadrocopter", *University of Munich Master's Thesis*, pp.1-88, 2011.

[19] J. Lentin, "Programming with ROS", *In book: Robot Operating System for Absolute Beginners*, 2018.

[20] J. Lee and J. Wang, et al., "Real-Time Object Detection for Unmanned Aerial Vehicles based on Cloud-based Convolutional Neural Networks", *School of Informatics and Computing, Indiana University Thesis*, pp.1-6, 2017.

[21] J. Leonard and P. Newman, et al., "Explore and Return: Experimental Validation of Real Time Concurrent Mapping and Localization", *IEEE Int. Conf. Robotics and Automation*, pp.1802-1809, 2002.

[22] J. Liu and R. Chen., "Sequential Monte Carlo Methods for Dynamic Systems", *Journal of the American Statistical Association*, vol.93, May 2004.

[23] J. Meyer and S. Kohlbrecher, et al., "A Flexible and Scalable SLAM System with Full 3D Motion Estimation", *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pp.1802-1809, November 2011.

[24] J. Škoda and R. Barták., "Camera-Based Localization and Stabilization of a Flying Drone", *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference*, 2015.

[25] K. He, et al., "Deep Residual Learning for Image Recognition", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.770-778, 2016.

[26] M. Beckler and P. Bailey, et al., "2D Simultaneous Localization And Mapping", *Matthew Beckler*, 2008.

[27] M.D. Zeiler and R. Fergus., "Visualizing and Understanding Convolutional Networks", *European Conference on Computer Vision*, pp.818-833, 2014.

[28] M. Montemerlo and S. Thrun, et al., "FastSLAM: An Efficient Solution to the Simultaneous Localization And Mapping Problem with Unknown Data", *Journal of Machine Learning Research*, vol.4, May 2004.

[29] M.R. Blas and S. Riisgaard., "SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping", *MIT OpenCourseWare*, vol.22, 2003.

[30] N. Dijkshoorn., "Simultaneous Localization And Mapping With The AR.Drone", *University of Amsterdam*, pp.372-376, 2012.

[31] P. David, R. Rui, and S. João., "An Evaluation of 2D SLAM Techniques Available in Robot Operating System", *11th IEEE Int. Symp. on Safety, Security, and Rescue Robotics*, 2013.

[32] R. Girshick, et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation" *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.580-587, 2014.

[33] R. Szeliski., "Computer Vision: Algorithms and Applications", *Springer Science & Business Media*, 2010.

[34] S. Kohlbrecher, et al., "Hector SLAM for Robust Mapping in USAR Environments", *ROS RoboCup Rescue Summer School Graz*, 2012.

[35] S. Ren, et al., "Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks", *Advances in Neural Information Processing Systems*, pp.91-99, 2015.

[36] S. Ruder., "An Overview of Gradient Descent Optimization Algorithms", *arXiv preprint arXiv:1609.04747*, 2016.

[37] Y. LeCun, et al., "Backpropagation Applied to Handwritten Zip Code Recognition.", *Journal:MIT Press*, vol.1, pp.541-551, 1989.