An Indoor Navigation Robot Using Augmented Reality

Austin Corotan
College of Science and Engineering
Western Washington University
Bellingham, Washington
e-mail: corotaa@wwu.edu

Jianna Jian Zhang Irgen-Gioro
College of Science and Engineering
Western Washington University
Bellingham, Washington
e-mail: Jianna.Zhang@wwu.edu

Abstract—**In this paper, we address the issue of autonomous robotic navigation in an indoor environment. An Augmented Reality (AR) based navigation system is created using the JAQL robot and an Google Pixel 2 Android smart phone. In developing this system, we aim to investigate the current capability of augmented reality as an all-in-one solution to indoor routing, localization, and object detection. Using Google's ARcore, we have developed a mobile application, which was used as both the sensor and the controller for the system. We highlight the features of ARcore and how they were applied to each of our navigation tasks. We evaluate the effectiveness of our system and present results from a set of experiments in a real environment under varying conditions. We conclude by discussing the strengths and limitations of our system and how we can make improvements in the future.**

Keywords-augmented reality; in-door navigation robot; Q-learning; ARcore; Arduino; Google Pixel 2 smart phone; JAQL

I. Introduction

Augmented reality (AR) is a combination of the real-world environment with computer-generated information such as graphics, sounds, and other perceptual information. AR brings digital components into our perception of the world, and can be used to enhance our natural environment and provide more powerful insight on the world around us. This environmental information becomes digitally manipulable and can be used for a variety of tasks such as object detection or recognition, motion tracking, and environmental understanding. The use of AR has been much more prevalent in recent years. Several AR development kits, including Wikitude, ARkit, ARcore, and Vuforia have made AR much more accessible to developers and allow this technology to be applied across many fields. Smartphones are accessible to a significant portion of the population and AR technology turns our smart phones into powerful environmental sensors. If smart phones can be used in replacement of high-end devices such as light detection and ranging (LiDAR) sensors or auxiliary devices such as beacons, then they have the potential to reduce costs of systems significantly.

In this paper, an AR-based navigation system, using the *JAQL* (Jianna Q-Learning) robot [1], is created to provide users with a low-cost, reproducible framework for autonomous indoor routing and localization. The JAQL system is implemented using Google's ARCore platform on Android and deployed to a Pixel 2 phone attached to the robot. In developing this system, we aim to investigate the current capability of AR as a tool for autonomous navigation and indoor localization. The remainder of the paper is organized as follows: Section 2 discusses the ARcore framework and previous work. Section 3 presents our system architecture and components. Section 4 describes our methodology for indoor navigation using ARcore. Section 5 explains the preliminary testing results and evaluation setup, including discussion on the strengths and weaknesses of the JAQL system. Section 6 concludes the current work and gives some future directions.

II. Background

A. Previous Work

Winterhalter et al. [2] implemented an indoor navigation system for RGB-D smart phones given 2D floor plans. Their system uses a Monte-Carlo particle filter localization approach and calculations are done off-board by transmitting sensor data to a server. They showed that this approach can be used online and yields accurate results.

Boniardi et al. [3] implemented a navigation system that uses a sketch interface to allow the operator to draw a rough map of an indoor environment as well as the desired trajectory. They employed a theoretical framework for sketch interpretation enabling a robot to perform autonomous navigation and exploration when a full metrical description of the environment is not available. The system was able to perform navigation tasks more than 65% of the time and only a small population of users believe the minimal representation was inadequate for successful navigation.

Cho and Hong [4] used a laser range finder for navigation and localization of indoor environments. Their mobile robot was globally localized by finding matching patterns between given vertex patterns and vertices from the range data provided by the laser range finder.

Correa et al. [5] implemented a two part system for indoor autonomous navigation for surveillance robots. The first part is a navigation system in which the robot avoids obstacles using a distance sensor Kinect. The second part used a neural network to recognize different configurations of the environment, which was trained using data captured by the Kinect sensor.

Ko et al. [6] implemented an autonomous navigation system by building a grid based map using scanned range data. They used Dijkstra's algorithm for path planning and a particle filter to estimate robot position and orientation using scanned range data.

B. *ARCore*

Google’s ARCore platform was released on March 1, 2018. ARCore includes features such as motion tracking, environment understanding, and light estimation providing developers information that can be used for numerous automation tasks. Motion tracking allows the phone to track its position relative to its environment, environment understanding includes features that allow the phone to detect the size and location of surrounding surfaces, and light estimation allows the phone to estimate the current lighting conditions.

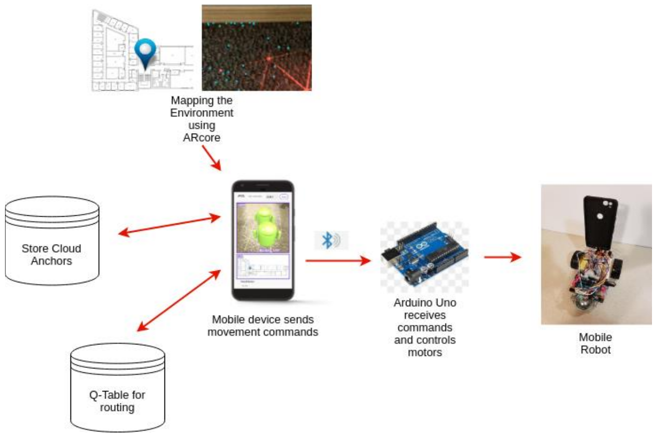
Mapping the Environment using ARcore
Store Cloud Anchors
Mobile device sends movement commands
Arduino Uno receives commands and controls motors
Mobile Robot
Q-Table for routing

Figure 1. System architecture diagram.

The ARcore motion tracking system can track the position and orientation of the camera relative to the world around it by tracking visually distinct features called feature points and using information from the inertial sensors on the mobile device. This technique is called *visual-inertial odometry* (VIO). More specifically, VIO is used to determine the camera’s pose. A pose describes a coordinate transformation one coordinate space to another, from an objects local frame to the world coordinate frame. Therefore the relative location of the feature points as well as their distance to the phone are constantly being tracked. Furthermore they can be displayed as a point cloud on the camera frame. ARcore also analyzes feature points that lie on a common surface which can then be displayed as planes. Virtual objects called anchors can be rendered and tracked while the phone is moving. Anchors represent fixed locations and orientations in the space. Cloud anchors can be saved to a remote server then rendered and tracked by the phone at a later time. Feature points are used to track the pose of both the camera and anchors in the world space.

ARCore comes with several benefits that made it a desirable candidate for our work. ARCore makes use of hardware that is already included with most smart phones, making it a great candidate toolkit for reaching the largest population while being efficient and low-cost. No additional sensors required. ARCore is available to all devices running Android 7.0 and later or iOS 11.0 and later. This covers 50% of all Android users and 89% of all iOS users [7, 8].

III. System Architecture

The goal of the JAQL system is to produce a framework that is completely controlled by AR on a mobile device. Our system is comprised of two main parts, the JAQL robot and a Google Pixel 2 smart phone. The mobile device is both the sensor and the controller of the robot. Therefore, the robot will be solely reliant on the device camera. A detailed diagram of the JAQL system architecture is shown in Fig. 1.

A. *JAQL Robot Design*

The JAQL robot used for our project is powered by an Arduino Uno with two Cylewet 12V DC geared motors. Each motor attached to a 68 mm diameter plastic wheel. A round ball bearings universal rotation caster is placed at the back of the robot to support and balance. On front top of the robot, a Google Pixel 2 smart phone is mounted. The communication between Arduino and the Google Pixel 2 is established via a HC 6 Bluetooth module. The current hardware total cost is about $40, and yet, it is smaller and much cheaper than that of our previous testing robot EGR1 [1].

B. *The Q-Learning Component Design*

The Q-learning component is implemented by the Q-team [9]. We use a 2-D building blue print of a floor plan as the input of navigation. This blue print, in pdf format, is parsed into 4 by 4 pixel square states. Then a general Q-learning program is used to generate a routing Q-table, which contains the learned shortest paths to different destinations.

C. *Mobile Application Design*

1) Implementation: The application was designed for Android 7.0 and later devices using the Android Studio SDK, the ARcore SDK, and the Open Graphics Library (OpenGL) written in Kotlin. SQLite was used to store our routing information and Firebase was used to store our cloud anchors.

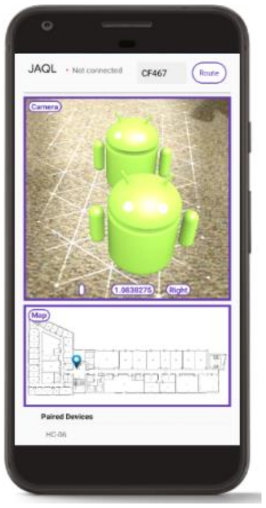

Figure 2. Interface.

2) User Interface: Fig. 2 shows the user interface for the phone application. The interface contains 3 main views. The uppermost view is the ARcore scene view. This will show all of the ARcore features rendered over the phones camera view using OpenGL. The middle view is the map view. This view shows the robot location over a blue print of our environment. This view features zooming and panning. The primary purpose of the map view is used to show the robot location within the environment. Further details on these views will be described in the subsequent sections. The bottom most view provides a list view so the user can chose which robot or Bluetooth device the phone should connect to. Once the robot is connect to the phone, the user can click on the drop-down menu at the top right of the screen to choose their desired location. They can then click on the "route" button to start the tracking and navigation toward their destination.

IV. JAQL Robot Approach

In creating JAQL system and investigating the capability of AR for indoor navigation, we focused on three major aspects of autonomous navigation: Routing, localization, and object detection. The goal of our system is to have a robot guide a user to a desired destination within a building. For our purposes, we limited the current JAQL system to a single floor. The starting location of the user is always assumed to be at the same location (home base). The robot will then guide the user to their desired destination without colliding with walls or objects in their path. The user will have information on their current location mapped in real-time. All of navigation and localization will be done on the mobile device.

A. Path Planning

In order to route the user to their desired location, the application must determine the optimal route from the lobby to the destination. Optimal route planning was done using a Q-learning algorithm.

```
Set $\gamma$ and initial state rewards in $M_r$
Initialize $M_Q$ to zero
**for** all possible initial states **do**
    Select a random initial state
    **while** the goal state hasn't been reached **do**
        Select a random action among all possible actions
            for the current state
        Get the state that this action points to
        Get the maximum Q value for this next state based
            on all possible actions
        Compute $Q(s, a) = r + \gamma \max_{a'} Q(s', a')$
        Set the next state as the current state
    **end while**
**end for**
```

Figure 3. Pseudocode for the Q-learning.

1) Q-learning: Q-learning is a reinforcement learning algorithm that can determine optimal routing paths from a start state to a goal state. The algorithm learns the optimal path by choosing random routes from random start states to the final goal states. An image parser was used to break up a blueprint of the floor into an adjacency list of states. Each state is a 4 pixel by 4 pixel region on the blueprint. There were a total of 25,217 states. Any states representing walls were removed from the list of states. States called *islands* also had to be removed. *Islands* in this context are states which have no link to the rest of the map. Initial state rewards were set such that the goal state had a reward equal to the number of total states and the rest of the states were set to zero. Once the Q-learning algorithm has converged, each state-action pair is assigned a value in a Q-table. The optimal route is determined by referencing the Q-table, choosing the action with the highest reward for each state along the route. Details of the algorithm is shown in Fig. 3.

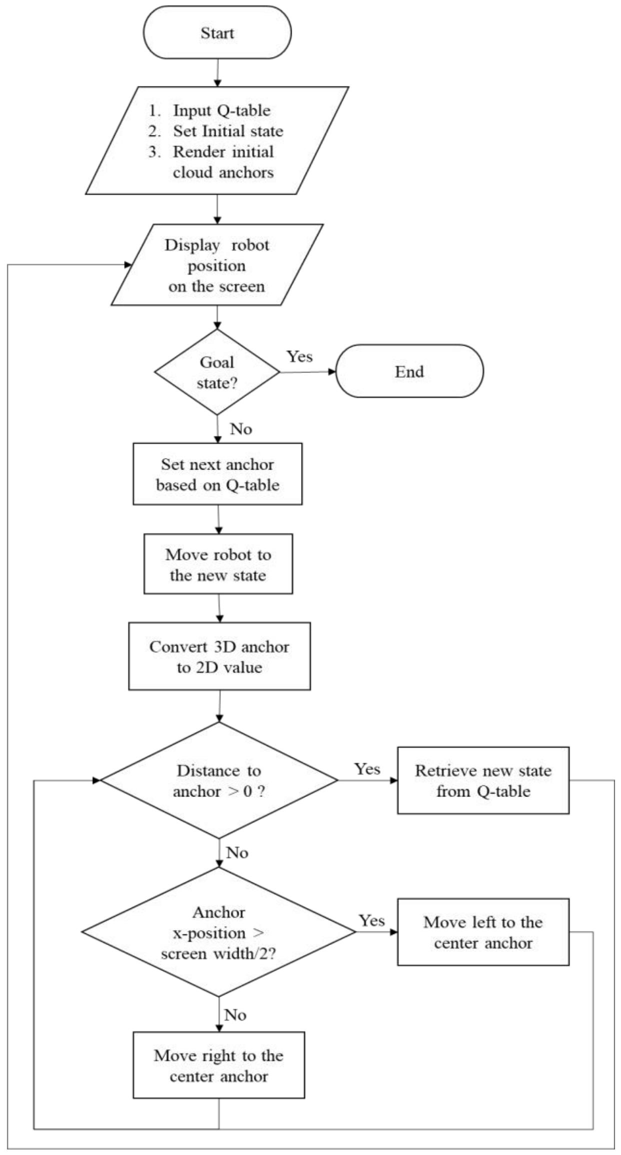
Start
1. Input Q-table
2. Set Initial state
3. Render initial cloud anchors
Display robot position on the screen
Goal state?
Yes
End
No
Set next anchor based on Q-table
Move robot to the new state
Convert 3D anchor to 2D value
Distance to anchor > 0 ?
Yes
Retrieve new state from Q-table
No
Anchor x-position > screen width/2?
Yes
Move left to the center anchor
No
Move right to the center anchor

Figure 4. JAQL routing system flowchart.

2) Anchors: Once the Q-learning algorithm learns the optimal path, the route is broken down into a list of robot movement commands to follow the provided path. However the robot motors are not consistent enough for the robot to maintain its position accurately along its route. To address this problem, we initially investigated the use of ARcore's vertical plane detection to center the robot along its route to the destination. However, this was unreliable because most of the walls in the building are completely white and therefore have no features for the camera to detect.

However, ARcore's horizontal plane detection was working exceptionally well in our environment, and was much more reliable. Once the camera was able to identify horizontal planes, it could be used to render anchors navigation.

3) Routing Algorithm: Fig. 4 details the general process of our routing algorithm. The first step is to store a cloud anchor at the starting state. This starting state never changes therefore we can place a cloud anchor at that location such that the robot can use it as a reference point for its initial position. Once the starting anchor and the orientation of the mobile device is set, the robot can begin routing. The next anchor is placed based on the next action that should be taken by the robot according to the Q-table. The robot will do the specified action then move toward the current anchor while continuing to keep it centered. Once the robot reaches that anchor the next action is determined and the next anchor is placed. This process continues until the robot reaches its final destination. This methodology allows the AR system to assist the robot in maintaining its proper position and orientation throughout its entire routing process. The ability for the robot to navigate reliability to its destination is primarily dependent on the effectiveness of the ARcore framework in tracking the position of the mobile device in the environment.

4) Coordinate System Conversion: The core of our outing algorithm required a conversion of the 3D world coordinates of each anchor to be converted and scaled to the 2D screen coordinates of the phone. The reasoning behind this conversion was that no matter what action the robot is supposed to take next, the robot will always try to keep the current anchor in the center of the phone screen. The anchors then act as waypoints for the robot to correct its position while it is a route to its destination.

We converted the world coordinates of each anchor to screen coordinates by using a matrix transformation. The matrix to transform the coordinates was calculated using the following equations:

$$V_{origin} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (1)$$

$$V_{ndc} = M_v \times V_p \times V_{origin} \quad (2)$$

$$x_{screen} = Screen_{width} * \left(\frac{\frac{V_{ndc,0}}{V_{ndc,3}} + 1}{2} \right) \quad (3)$$

$$y_{screen} = Screen_{height} * \left(\frac{1 - \frac{V_{ndc,0}}{V_{ndc,3}}}{2} \right) \quad (4)$$

The first step was to calculate the normalized device coordinates (NDC). This was done by multiplying the frame's view matrix (M_v) by its projection matrix (M_p) then by V_{origin} as seen in (2). Once the NDC coordinates are determined, they were converted to screen coordinates. This conversion is shown in (3) and (4).

B. Localization

A major issue with indoor navigation is precise localization. GPS technology doesn't quite have the precision necessary for indoor environments. Furthermore, issues with signal strength can have an effect on localization as well. Our approach for indoor localization includes the use of the ARcore motion tracking system combined with a scaled blueprint of the environment.

Using the camera pose information retrieved from this algorithm, we can track the change of the robot's translation in the world space over time relative to where tracking initially began. The device translation from frame to frame can be retrieved then properly scaled and used to track movement on the building blueprint in the map view.

Using the robot's position on the blueprint, we can estimate and verify the robots current state during its routing. Furthermore, the user will be able to identify where they are in the building as the robot is guiding them to their destination. Fig. 5 describes the algorithm used to track the motion of the robot on the blueprint map of the floor.

C. Object Avoidance

While the robot is making its way toward its desired location, it needs to be able to avoid any possible obstacles. As mentioned previously, ARcore can track visually distinct feature points which can be rendered using OpenGL. Using VIO, ARcore tracks the change in position of feature points relative to the camera. As the camera pose changes, feature points closer to the camera will move less than those farther away. Using this information, ARcore can calculate the relative distance of each feature point from the camera. This information can be filtered and tracked to try and determine whether there is some object near the robot.

```
Set the initial position of the camera on the blueprint
Set initial pose to 0
for each frame do
    Retrieve the camera pose and set to current pose
    $x = \propto (initPose[0] - currentPose[0])$
    $y = \propto (initPose[1] - currentPose[1])$
    Use scaled $x$ and $y$ translations to translate map marker
    initPose = currentPose
end for
```

Figure 5. Pseudocode for localization using the camera pose.

For our application, we only care about objects in front of the robot. The range of the ARcore frame extends past the width of the building hallways and therefore feature points on both the floor and the hallway walls could be tracked. If these feature points are accounted for, then they would interfere with object detection and produce false positives. In regards to our object detection mechanism, we chose not to include any feature points lying on the horizontal plane or more than one robot length (about one foot) away from the robot. All feature points within this range are tracked. Positions and distances to these points are stored and recalculated with each change in the camera frame. Feature points are sorted and colored differently depending on distance. If there are any feature points detected that is within 1 robot length of the robot, then the robot stops its movement and waits until there are no feature points detected before resuming movement.

For the scope of our work, we decided not to have the robot take evasive action and instead just halt all movement. We realized that the hallways are much too busy for the robot to account for all obstacles at any given moment. For example, consider the situation where there are two people walking down the hallway toward the robot on either side of it, the robot would be unable to veer left or right in this situation. Furthermore, human obstacles are most likely moving much quicker than the robot therefore the only situation where this type of action would be necessary would be to avoid static obstacles. We will leave evasive maneuvers for future work and solely focus on the evaluation of object detection in general.

V. EXPERIMENT SETUP

To evaluate the JAQL system, preliminary experiments were conducted in a real world environment. Details of the environment are shown in Fig. 6. The environment is a single floor in a university building. The environment consists of a lobby, professor offices, as well as classrooms. Our routing system assumes the user is starting from the same starting location (home base) every time. This location is marked by the blue marker. The destination from our longest experiment is marked by the red marker. There are 57 possible destinations the user can request. Each destination is a office or classroom on the floor. The red path is the optimal route determined by the Q-learning algorithm. The JAQL system consists of three major components: routing, localization, and object detection.

We set up three experiments, with each experiment having increasing routing distance. The reason for this setup was to analyze the performance degradation relative to distance and to pinpoint the limitations and capabilities of the application.

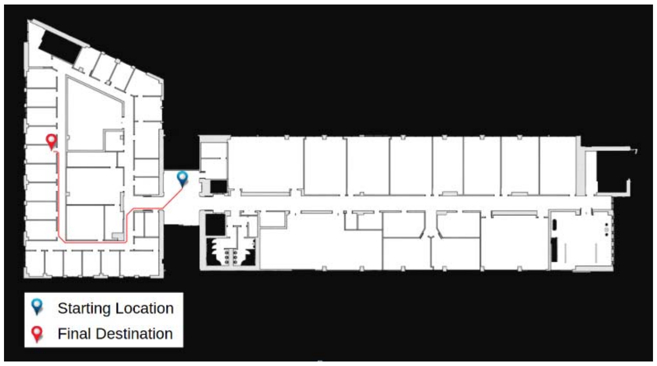

Figure 6. Blueprint of the test environment.

To measure effectiveness, we examined three different variables throughout the routing process: the final location of the robot based on the Q-learning algorithm, the final location of the robot according to the JAQL system, and the final real-world location of the robot. The final location of the robot based on the Q-learning algorithm represented the true location. Therefore, for each experiment, we measured the final offset between the true location, and the real-world position of the robot, and the final offset between the true location and the location of the robot according to the JAQL system. The JAQL system's routing is accurate if the real-world position offset is small and effective at localization if the system offset was small. We tested the system 10 times for each experiment and took the average offset values of the 10 tests.

TABLE I. AVERAGE OFFSETS FOR EACH EXPERIMENT

Route Location	Metrics		
	Total Distance	*Real-World Offset*	System Offset
CF420	15m	0.14m	0.08m
CF494	26m	0.21m	0.13m
CF477	45m	0.58m	0.91m

To evaluate the effectiveness of object detection we placed various objects along the path of the robot as it was routing to its destination. We wanted to look at how well the JAQL system did depending on the object and the environmental conditions. Data that we recorded were the type of object, if the object was static or dynamic, the lighting conditions, and the detection percentage across all experiments.

VI. PRELIMINARY RESULTS

The results for the offsets were as expected. In general, the JAQL system was able to route and localize well for both the short and intermediate routes. However, significant performance drops occurred for the longest route. The routing offset was lower at the closest two locations, however it was much worse than the real-world offset at the farthest location.

TABLE II. AVERAGE OFFSETS FOR EACH EXPERIMENT

Object Type	Features		
	Type	*Lighting Condition*	Detection
Person	dynamic	good	21%
Cardboard Box	static	good	93%
Dark Chair	static	good	84%
Person	dynamic	dark	0%
Cardboard Box	static	dark	60%
Dark Chair	static	dark	33%
Person	dynamic	bright	13%
Cardboard Box	static	bright	87%
Dark Chair	static	bright	87%

In regards to the object detection, the JAQL system was very sensitive to the lighting condition of the environment, experiencing a significant drop in object detection capability in darker environments. Furthermore, the JAQL system had a difficult time detecting a moving person, and currently could not detect a person in environments with poor lighting. This is informative since people would be the most likely obstruction for the robot as it is navigating.

VII. DISCUSSION

Results from Table I show that the JAQL system has more consistent routing capability across all routes. However, it was able to localize slightly better while routing to CF420 and CF494. Perhaps this is due to the drift that occurs in the tracking of feature points as the sensor moves further away from them. When objects obstructed the view of the camera they seemed to shift the tracking planes slightly. This obstruction may have played a role in the drop in performance. Any shifts in tracking early on would affect the position of the robot greater the farther it has to travel. Using AR motion tracking for indoor navigation appears to work well for routing and localizing at short and intermediate distances, however more work needs to be done for routing long distances more precisely.

Results from Table II combined with observation during our experimentation have led us to conclude that using only ARcore feature points is not enough for consistent reliable detection. Furthermore, the JAQL system is entirely dependent on one sensor, the camera. If the sensor is obstructed in any way, it will lose its current tracking and therefore its routing and localization capabilities. Therefore, for object detection to work effectively with JAQL system, we may need to add some supplementary computer vision technique that specializes in detecting various objects in different lighting condition.

VIII. CONCLUTION AND FUTURE WORK

This paper presented an AR-based navigation system, JAQL System, to provide users with a low-cost, reproducible framework for autonomous indoor navigation. In developing and testing the JAQL system, we evaluated the capability of AR technology for this domain. Our primary focuses were routing, localization, and object detection. Our system took advantage of the ARcore motion tracking feature and used anchors as way-points to improve accuracy. Localization utilized a blueprint of the building and the camera's pose in the world space. Object detection was implemented by tracking the positions of certain feature points detected by ARcore. Based on our evaluation, we determined that augmented reality is a powerful technology that has the potential to be applied in similar domains.

We would like to incorporate other computer vision technologies to enhance the JAQL system for better object avoidance. Another important benefit of ARcore is that the robot can grab the raw image information from the scene on fly. This information can then can be passed on to custom computer vision models which specialize in certain tasks such as object recognition. Identifying whether an object is static or dynamic, and its location relative to the robot would be an enhancement to the JAQL system. Lastly, we would like to incorporate natural language processing feature to the JAQL system, that way the robot can interact with its environment through sound. It would be interesting to look into object interaction by listening and identifying certain noises.

REFERENCES

[1] Jianna Jian Zhang Irge-Gioro, "Making an Affordable Educational Guide Robot: EGR1", In the Proceedings of the of the International Conference on Artificial Intelligence: Techniques and Applications, Shanghai, China, pp. 337-342, September 2016.

[2] W. Winterhalter, F. Fleckenstein, B. Steder, L. Spinello, and W. Burgard, "Accurate indoor localization for rgb-d smartphones and tablets given 2d floor plans," in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sept 2015, pp. 3138–3143.

[3] F. Boniardi, A. Valada, W. Burgard, and G. D. Tipaldi, "Autonomous indoor robot navigation using a sketch interface for drawing maps and routes," in 2016 IEEE International Conference on Robotics and Automation (ICRA), May 2016, pp. 2896–2901.

[4] S. H. Cho and S. Hong, "Map based indoor robot navigation and localization using laser range finder," in 2010 11th International Conference on Control Automation Robotics Vision, Dec 2010, pp. 1559–1564.

[5] D. S. O. Correa, D. F. Sciotti, M. G. Prado, D. O. Sales, D. F. Wolf, and F. S. Osorio, "Mobile robots navigation in indoor environments using kinect sensor," in 2012 Second Brazilian Conference on Critical Embedded Systems, May 2012, pp. 36–41.

[6] N. Y. Ko, S. W. Noh, and Y. S. Moon, "Implementing indoor navigation of a mobile robot," in 2013 13th International Conference on Control, Automation and Systems (ICCAS 2013), Oct 2013, pp. 198–200.

[7] G. LLC. (2018) Distribution dashboard. [Online]. Available: https://developer.android.com/about/dashboards/

[8] A. Inc.App store: https://developer.apple.com/support/app-store/

[9] Alyis Clark, Christopher Roybal, David Carew, and Jayce Brewer. JAQL Project, Advisor: Dr. Jianna Jian Zhang Irgen-Gioro, College of Science and Engineering, Computer Science Department, fall 2017- fall 2018.