VisualBot: A Smart Indoor Robot to Guide Visually Impaired Individuals

Yasmin Abouhelwo, Ali Mohsen, Michel Pasquier, Salam Dhou
Computer Science and Engineering Department
American University of Sharjah
Sharjah, UAE

***Abstract*—Navigating in indoor spaces has always posed a major challenge for visually imparied individuals. With the advancements in computer technology, there has been a continuous effort from researchers to develop and design assistive devices to aid the visually impaired. However, the task of indoor navigation still remains widely unexplored since indoor spaces are significantly more complex than outdoor environments. Therefore, we have presented an assistive robot, VisualBot, which is designed to guide visually imparired individuals in indoor spaces such as academic buildings. The robot is able to guide individuals to different locations in the building using voice assistance and recognition technology to deliver real-time guided help. Our robot includes a laser distance sensor used for mapping the environment and planning a path for the robot to assist the user to the chosen destination. The system is also capable of detecting different obstacles such as people, chairs and doors, and communicate that information reliably with the user. The proposed prototype aims to integrate a more comprehensive autonomous system to provide better independence and self-sufficiency to visually impaired individuals.**

***Index Terms*—Robotics, visual impairments, Path Planning, indoor navigation , Object Detection**

I. Introduction

WHO estimates that at least 2.2 billion people globally have vision impairment [1]. Visual impairment is involved with various conditions and is generally understood as the loss of vision acuity and can differ in stages and severity. However, it is a general truth that losing ones' sight causes a lot of psychological and physical difficulties in life, from everyday simple tasks to working or performing activities in indoor or outdoor environments. A study conducted by Li and Lee found that visually impaired people spend around 80 to 90 percent of their time indoors [2]. The study indicated that navigating or going through unfamiliar places independently can be an extremely challenging task for the visually impaired that reduces their confidence in independent navigation. Visually impaired students find difficulty navigating around campus buildings because of furniture or sometimes equipment in the hallways. In malls, some find it difficult to shop because of carts and people. They also felt it was hard for them to navigate the mall to find the shops they needed to visit [2].

Now with the significant estimate of visually impaired, the need for effective solutions to help them in navigation and orientation is dire. The main aim of an assistive device built for visually blind is to make them capable, independent, and self-sufficient. Assistive technologies developed between the years 1990- present day, like NavBelt, smart cane, Smart Glass, ISANA, etc., used different combinations of hardware, sensors, image processing and AI based solutions and were all evaluated based on different parameters like Power Consumption, Weight, Economic, and User-Friendliness [3]. Such devices still lack accuracy in navigation and object recognition, ease of use, adaptability, efficiency in real-world applications and integration of features [3].

This paper will explore designing a robotic guide system to be used for guiding users just as a guide dog would be used. The robot will guide blind people in indoor environments. The platform includes a module for navigation, path planning and mapping where obstacle detection and SLAM (Simultaneous localization and mapping) are integrated. Moreover, the platform integrates voice assistance and recognition to provide real-time guided assistance along the journey. Through expanding the functionalities of existing indoor navigating robots, we aim to aid visually impaired individuals by guiding them through closed environments using an autonomously operated robotic guide. The proposed system includes a TurtleBot3 Burger robot that can be attached to a durable leash. The user can hold the leash, inform the robot of his/her desired requests and be guided to the requested destination. The design includes using Nvidia's GPU for processing and a Bluetooth connected headset to analyze information acquired from environments surrounding the robot, and audibly inform individuals of critical information they may need.

II. Literature Review

Different applications and smart devices have been proposed to guide visually impaired in indoor and outdoor environments. In this section, we describe the most relevant systems designed for navigation purposes by researchers.

CaBot is a suitcase shaped autonomous navigation robot designed by researchers. It is designed to guide blind people and includes the use of a vibro-tactile handle for providing hepatic feedback to the user [4]. However, the use of voice recognition and assistance is not provided which increases the difficulty for the user. Likewise, a guide robot was tested in Kanagawa Rehabilitation Hospital in Japan. The user chooses a destination to navigate to by pressing certain buttons and the

Identify applicable funding agency here. If none, delete this.

velocity of the robot is determined by the force exerted from the user on the handle [5].

In [6], researchers proposed a prototype robotic system which uses amazon cloud for voice recognition. The system is designed to detect the state of traffic lights and guide the user accordingly. Researchers proposed and designed a local navigation robot to guide a blind user to a landmark object [7]. They make use of YOLOv3 for object detection and ROS for mapping. In another research, the quadrupedal Mini Cheetah robot was used and connected to a leash to allow the robot to change its intrinsic dimension and fit into narrow spaces. The robot can guide humans by having the leash go slack or taut [8]. A robotic guide dog system, which is tied to a leash, has also been proposed to learn and follow man-made trails. An NVIDIA Jetson TX1 embedded system was used along with motors and cameras [9]. Another group of researchers proposed Eyedog, which is a prototype assistive-guide robot. It provides vision-based navigation and laser-based obstacle avoidance in specific outdoor settings [10]. The system only offers a GUI for the user to interact with; however, this will not be feasible system for a visually impaired individual to interact with or use.

Other systems have been designed such as wearable modules and smartphone applications. Researchers designed NavCog3, a Smartphone application to help visually impaired in indoor navigation. The app also makes use of the built-in IMU sensors in smartphones and Bluetooth Low Energy (BLE) beacons to provide turn-by-turn instructions and immediate feedback for users [11]. A wearable module was designed for the user to position on his/her leg. A microcontroller and RFID reader were embedded in the system used for localization and navigation [12]. In another research, a small quadcopter tied to leash is used to guide visually impaired in indoor environments. The guidance system is based on the emitted sound from the quadcopter and the haptic stimulus provided from the leash [13].

Based on the conducted research and review of different systems of various designs, our proposed prototype offers a well-rounded integration of mapping, navigation, voice recognition, voice assistance and object detection which provides the user with a better ability to navigate in indoor spaces. This design even includes detecting doors and their status when reporting back to the user with voice assistance. Using hepatic feedback was not used like in [4] [5] [8] and voice assistance and recognition are improved instead to provide real time guided assistance.

III. Description of Methods

In this section, the explanation of some major concepts and algorithms used in the system is introduced. This includes: ROS, SLAM, AMCL, PocketSphinx and DeepSpeech.

A. *Robotic Operating System (ROS)*

The main interface that we used to manage and program our robot is ROS. It is a framework that consists of various tools and libraries, some of which are used to create maps, localize, and plan the motion of the robot. The framework also allows us to utilize the peer-to-peer architecture for designing nodes and systems that can communicate synchronously or asynchronously as required.

Nodes are used to allow for communication between different systems in the robot. The implementation is made efficient and reusable by building nodes and giving them specific functions. All nodes can be either subscribers and/or publishers. This allows for the exchange of messages using topics and services.

- Subscribers: a node that registers its information and topic with a master node (roscore) and receives information over the topic it "subscribed" for.
- Publishers: a node that registers its information and topic with a master node (roscore) and sends a message to connected subscriber nodes that are subscribed on the same topic

B. *Simultaneous Localization and Mapping (SLAM)*

To create a map of an indoor environment, SLAM is required. SLAM is a process which involves the use of sensors to simultaneously localize and keep track of a robot while constructing or updating a map of the surrounding environment. This process can be completed by using either visual-based or LiDAR based systems. Visual-based SLAM was found inefficient when the surrounding environment lacks texture or sufficient light [15]. Therefore, a laser sensor will be used since it captures the environment better as the robot is moving faster and provides better localization while creating the map.

Gmapping is a SLAM algorithm which works by utilizing a Rao-Blackwellized particle filter to create a grid based-map [14]. Each particle will represent a possible orientation and position for the robot. Such particles will be randomly distributed at first and will eventually converge to give the most likely estimate to where the robot is in the map. The result of using the Gmapping algorithm will be the generation of an an occupancy grid used to represent a map of the environment using an evenly spaced field of binary random variables. Each variable will show the existence of an obstacle at that location in the environment.

C. *AMCL and Path Planning*

AMCL will also be required for localization. Monte Carlo localization is a type of particle filter used for representation of multi-modal distribution of probability. The Adaptive version of MCL was designed to have an improved real-time performance by lowering the execution time and having less samples. The information required for this process includes: the position and orientation (x, y,) of the robot at time t (xt), distance information obtained from the distance sensor (z0...t), and the movement information obtained from the encoder up to time t (u0...t).

D. *PocketSphinx*

PocketSphinx is an open-source toolkit which is used for voice recognition. It is designed to receive the audio and decode it by file or stream looking for phonomones and mononoms that match what is contained in its dictionary and language models. This will allow for keyword searches and transcriptions of spoken audio. The use of PocketSphinx is only limited for the recognition of the activation word which will allow the user to request the relevant command from the robot. PocketSphinx uses language models and dictionaries for voice recognition. The language model is a pre-trained statistical model, and the phonetic dictionary provides the mapping between words to a sequence of phenomes.

E. *DeepSpeech*

DeepSpeech is another open source toolkit used for Speech-To-Text transcription [16]. It was first a neural network architecture published by a group of researchers at Badiu, and then it was implemented by Mozilla. A pre-trained model is provided and can be used for building.

IV. System Architecture

A. *Hardware system*

The design of the robot includes using a TurtleBot3 burger robot with the addition of other hardware components. The TurtleBot3 is ROS-based mobile robot that is affordable, scalable and has the necessary sensors and boards that can be used for AI and autonomous applications. As can be seen in figure 1, the main components that come with the robot which are used are: a 2D-LIDAR, a Raspberry Pi 3B+, an OpenCR microcontroller, 2 Wheels, 3 3D printed chassis, 2 DYNAMIXEL motors and a Li-Po battery. An NVIDIA Jetson AGX Xavier was used as the central computing board, and it comes with a 512-core Volta GPU with Tensor Cores and an 8-core ARM v8.2 64-bit CPU. A webcam was used and connected to the GPU to stream video directly and allow for object detection and recognition. For voice recognition and communication, a headset is used for communication with the robot and a Bluetooth speaker is used to enable the robot to communicate back with the user. A Bluetooth USB adapter was used to connect the GPU to a wireless pair of headphones and speakers.

To power up the components, 2 main batteries are used. The Li-Po battery which is provided for the TurtleBot3 robot was used to power the OpenCR board and then the Raspberry Pi was connected to the OpenCR using the GPIO pins. As for the GPU, a 11.1v 52000 mAh battery was used and connected directly to the DC barrel jack of the device. The GPU was connected to 4 different components. This includes the camera, USB Bluetooth dongle, the RPI and the wireless adapter. The raspberry pi was connected to the GPU using a USB cable that is used as a network adapter. The OpenCR board is connected to the LIDAR sensor and the wheels.

Since the GPU runs on 18.04 LTS, the Melodic Version of ROS was built on both the RPI and GPU to ensure compatibility. Furthermore, the wireless adapter connected to the GPU is used to allow us to view and control the processes occurring and test the design. By using a router, the GPU is connected to the created network, and we were able to VNC, and view control the ongoing processes from a monitor.

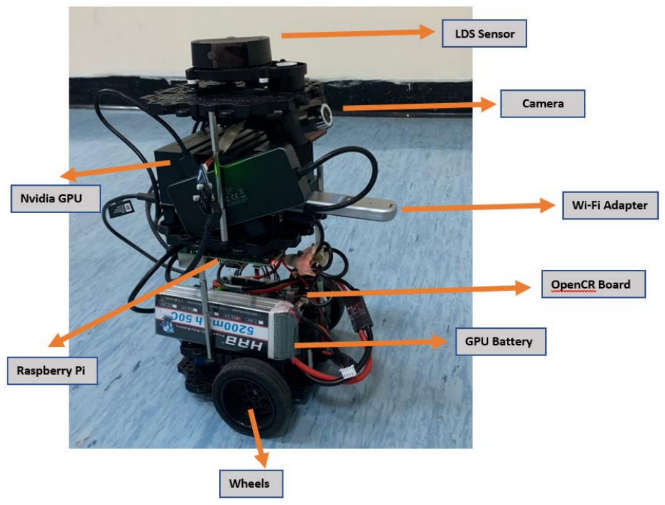


Fig. 1. TurtleBot3 with all connected hardware components

B. *ROS Nodes Architecture*

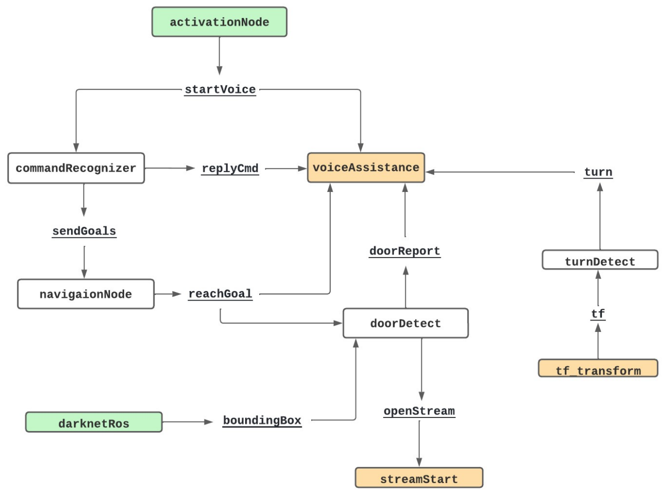


Fig. 2. Implementation of ROS nodes

To implement the system, a different set of nodes are created and each act as either a subscriber, publisher or both a publisher and subscriber. Figure 2 indicates the topics, which act as channels for nodes to communicate on. An arrow feeding into a node indicates that this node is subscribed to that topic and an arrow coming out of a node indicates that a node is publishing on that topic.

1) The activationNode publishes messages on the topic startVoice when the right input is received. The user will typically start with using the activation word, "Hello Robot", to wake up the robot and allow for receiving further instructions. The commandRecognizer and voice-Assistance nodes are both subscribers listening to the

activationNode. When the user mentions the keyword, the commandRecognizer node will listen and open the DeepSpeech voice recognition stream to detect the commands that the user wants the robot to execute. Using DeepSpeech enables the recognition of longer sentences with better transcription accuracy.

2) The commandRecognizer is also a publisher which will send the relevant x and y coordinates on the map of the location the user requested to the navigationNode. The system will generate a path and plan the motion of the robot as is it is guiding the user to the destination. The navigationNode will also publish a message informing the doorDetect node that the location has been reached and the system needs to detect the presence of a door and whether its open or closed.
3) The voiceAssistance node is a subscriber continuously listening on 5 different topics. The user must be continuously updated and informed of the status of his request, object in the way, doors and turns. The robot will be able to respond back with the appropriate command for the situations listed below:
   - when activation word is detected
   - when user requests to be guided to any location
   - when a door is detected
   - when an object is detected
   - when a turn is detected
4) The doorDetect node is responsible for publishing to the "openStream" topic and activating the startStream node when the robot reaches the destination. The startStream node is used to start the camera stream for a period of 40 seconds and apply YOLO door detection. This node will also be listening to the boundingBox topic to know what objects are being detected.
5) The darkNetROS node is responsible for publishing information about the bounding box of the detected object, such as: class, id, xmin, xmax. When the object is detected, this node will receive a string with the name of the object and will publish to the voiceAssistance node that there was an object detected (Closed/open Doors) and the user will hear specific instructions about the status of the door.
6) Finally, the turnDetect node is a publisher and subscriber. It listens to the position of the robot from the tf transforms. As the robot is moving, the position of the robot with respect to the map is changing and getting updated to this node. When there is change in the orientation and X,Y positions, the robot will detect that this translation indicates a turn. Thus, this node will publish to the voice recognition module that there is a turn, and then the robot will inform the user.

C. Mapping

To test our design, a map of a certain area in the engineering building is created. The information needed for the process of mapping with SLAM consists of both the distance data collected from the LDS mounted on the robot and the pose value which is dependent on the odomotery information. In ROS, the distance measured is known as "scan" and the pose is known as "tf or transform" because the position and orientation of the robot changes as the robot is moving.

The process starts with the LDS sensor node sending the "/scan" data to the gmapping algorithm. As the TurtleBot is controlled manually by the keyboard, translational and rotational speed instructions are sent to both"cmd_vel" and "TurtleBot_core" nodes. The "TurtleBot_core" node executes the commands to move the robot and at the same time publishes the odometry information and the relative transform coordinates "tf". The gmapping nodes receives the required information and creates the map. Finally, the "map_saver" node creates the pgm and yaml of the map and saves the created map. The process of launching and maintaining the nodes is done manually.

As can be seen in Figures 3 and 4, the map created in ROS is represented by a 2D Occupancy Grid Map (OGM). Each grid corresponds to a value which is indicative of how likely this area is to be occupied. White corresponds to the free area where the robot can move, black is the occupied area where the robot cannot move, and gray is the unknown area. The greyscale values can range from 0-255

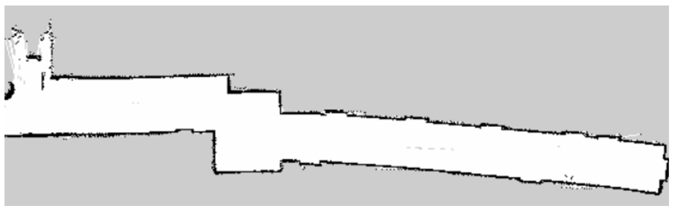


Fig. 3. original SLAM-based map generated for corridor

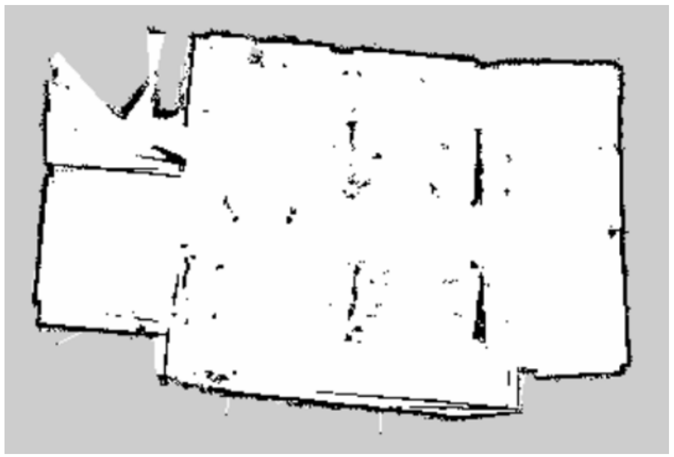


Fig. 4. Original SLAM-based map generated for lab room

The map for 2 setups was created. Figure 3 shows the map created for one hallway that is located on the side of the room shown in figure 4. However, the map generated from SLAM is not perfect given that there is sensor noise and errors from the laser sensors and odometry calculations. As can be seen in

Figure 4, the robot was not able to accurately create the map for the tables since they have slim and narrow corners and the laser sensor could only map such objects as dots. Therefore, a software tool [Krita], which is an image editor, was used to improve the quality of the map and combine the 2 maps together to be used in later stages of navigation. The maps of both the hallway and the lab room were combined to create one map for the robot to use which is shown in Figure 5. In Figure 5, we decided to increase the width of the lines and make them more defined since it allowed the robot to localize and navigate better to the desired location. We increased the width of the tables which are represented by vertical bars. Increasing the width allowed the robot to walk past the tables safely without colliding with them.

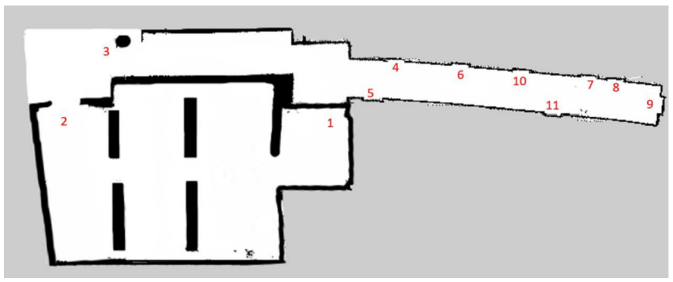


Fig. 5. Modified final map of building used for testing

To assign locations on the map and allow the user to specify his/her desired location, we used the transform functionality of ROS to know the relative x and y coordinates of the locations with respect to the map. It defines offsets in terms of both translation and rotation between different coordinate frames. To do that we used the transform from the /map to /base_link. The numbers notes in figure 5 in the map corresponds to a location in the original building. Each number is noted with the x and y coordinates of that location and the name of the room or area in table I.

TABLE I
TABLE TYPE STYLES

Point on Map	Assigned Location Name	X and Y coordinates
2	Door lab	-20.84, -0.847
3	Vending machine	-18.065, 3,454
4	Room 1237	-3.618, 2.548
5	Room 1234	-4.785, 2.144
6	Room 1457	-0.765, 2.577
7	Room 1459	6.29, 1.927
8	Room 1659	7.904, 1.97
9	Exit door	9.604, 1.41
10	Room 1678	2.735, 2.163
11	Room 3458	4.27, 1.40

D. Path planning and localization

As can be seen in figure 6 on the right , the robot physically exists in a certain location within the map. The green space in the figures stands for the “particles.” Each particle is an arrow that represents a set of directions and orientations that the robot can have.

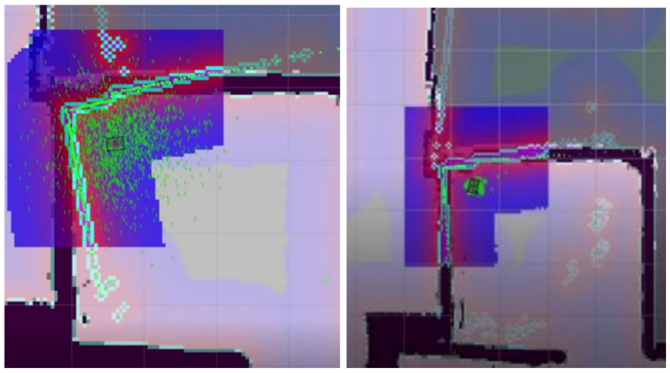


Fig. 6. Localization process by AMCL

For the robot to obtain a better estimate of its current position with respect to the map, it is required that the robot rotates and moves a short distance around the initial location. The best settings and options that we found for obtaining the best estimate and results in localization are as follows:

1) Rotate the robot at around 0.1m/s in place using teleoperation commands
2) Move the robot a short distance of 100m or less around the starting location. Figure 6 on the left shows the results after the robot was able move around the starting point. The green arrows considerably decreased and converged to the most probable location that the robot will be in.

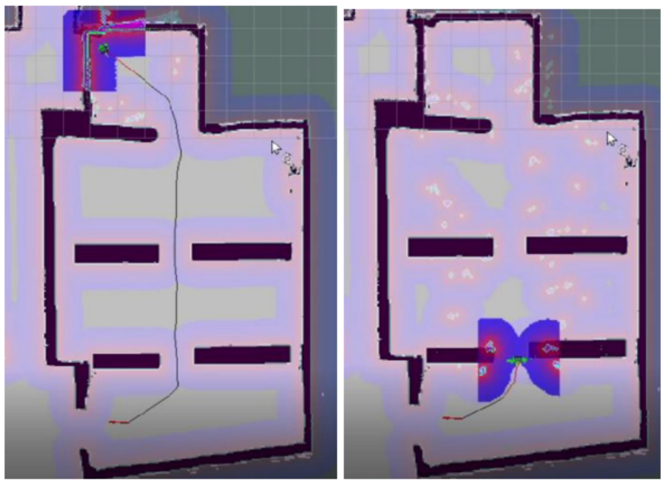


Fig. 7. Generation and execution of planned path to destination

The destination is then marked or specified by the user and the path is generated. The algorithm used to create the initial plan as shown in figure 7 on the right is Dijkstra’s algorithm. The robot then executes the global plan and begins navigating in the real world to the destination. However, due to the existence of obstacles and the changing conditions around the robot, this change needs to be accounted for by the local planner and local costmap. The local 2D costmaps can be seen in the purple and pink layers surrounding the robot in figure

7 and it is used to calculate the obstacle area, collision area, and the possible movable area for the robot.

Costmaps are represented with values between ‘0’ and ‘255’, with 0 representing the free space and 255 represents an occupied area. Thus, in figures 7-10 the pink area is the actual obstacle that the robot will collide with, and the blue/purple shows the point where the center of the robot approaches this area. The costmaps are constantly updated and changed where obstacles are either cleared or marked depending on the environment and the data collected from sensors.

The local planner in coordination with the data collected from the local costmaps and the global plan recalculates a part of the plan with a configurable look-ahead distance. This can be seen clearly in Figures 7 on the left with the small red line that overlaps with the original black line representing the whole path. The robot updates this small part of the path and executes this update in a small area around the robot.

E. Object Detection and Door State Recognition

Door detection is an integral module in our system. Once visually impaired individuals are successfully guided to a room, having information about their entry path, or door, gives the visually impaired assurance on their decision on whether to proceed in entering a room or to ask for assistance. In our door detection module, the Darknet version of YOLOv3 was used , which is a deep learning convolutional neural network algorithm.Darknet was chosen rather than other YOLO Models such as Keras. This was due to its reliability and its compatibility with ROS, as it has a Darknet ROS version that makes use of the subscriber-publisher based system for communicating with other nodes. YOLOv3 detects objects from live feeds in real time through the Darknet ROS framework. Furthermore, it differs from other convolutional neural network models as it is trained to do both classification and bounding box regression at the same time.

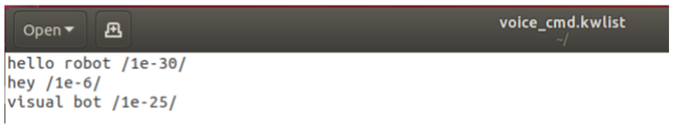
To train our network,about 1500 images were labeled from an open source dataset (DeepDoors2). The dataset used contained 3000 testing, training, and validating image data for closed, semi-open, and open doors of 640x480 dimensions. The chosen dataset was a good choice as it contained objects in front of the door such as people, animals, and other inanimate objects; this increased the chances of finding the door indoors. For labeling the data, an image annotation tool was used (LabelImg) that saves labels as XML files in PASCAL VOC format. After getting the labels from the image annotation tool, the XML files were converted to text files that only have the image name and the bounding boxes of the door in the image. This was done since Darknet ROS only supports text files for data annotation.

The initial training of the network was done on a GTX 1060 GPU with 64 batches over 6000 iterations. The process of training took around 10 hours. In training the network, the accuracies obtained ranged from 80The following parameters were used to train the network: batch=64, subdivisions=16, width=416, height=416, channels=3, momentum=0.9, decay=0.0005, angle=0, saturation = 1.5, exposure = 1.5, hue=.1, learning rate=0.001, burn in=1000, max batches = 6000, policy=steps, steps=4800,5400, scales=.1,.1

As for detecting other objects a pretrained model based on the COCO dataset was used. The COCO data contains 80 detectable objects including persons, coffee tables, chairs, and other furniture pieces.

F. Voice recognition

Voice recognition is used in communicating the needs of the user and translating them into actions the robot can take. With headphones or other Bluetooth connected deices, the user can activate the robot and select a desired location in the building to go to.



```
hello robot /1e-30/
hey /1e-6/
visual bot /1e-25/
```

Fig. 8. Keyword list and the assigned threshold values

PocketSphinx is used to detect the activation word to start the robot. The keyword spotting mode is used to detect a set of specified keywords. Every keyword has a threshold value that allows for the word to be detected in continuous speech. As shown in the figure below, the activation word(s) is “hello robot” and it is given a higher threshold compared to other since its is a longer key phrase. This threshold was attenuated and changed based on conducted tests in order to balance between “false alarms and missed detections”[1].

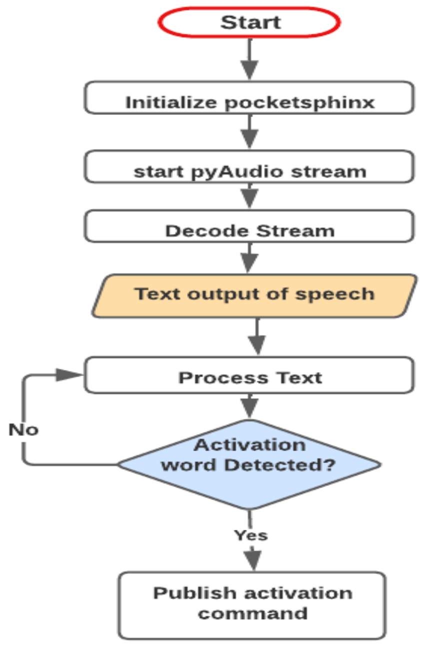


Fig. 9. Flowchart demonstrating sequence of PocketSphinx implementation

The flowchart in figure 9 shows the logic sequence implemented for using PocketSphinx. First, the language models, dictionary, and keyword list. Next, PyAudio is a set of python bindings used for playback and recording audio. The Pocketsphinx decoder is used to transcribe the collected raw audio data. The generated text is then used for further processing to detect the whether the activation word “Hello robot” was used by the user. This stream is continuously running to constantly be able to detect the user’s command.

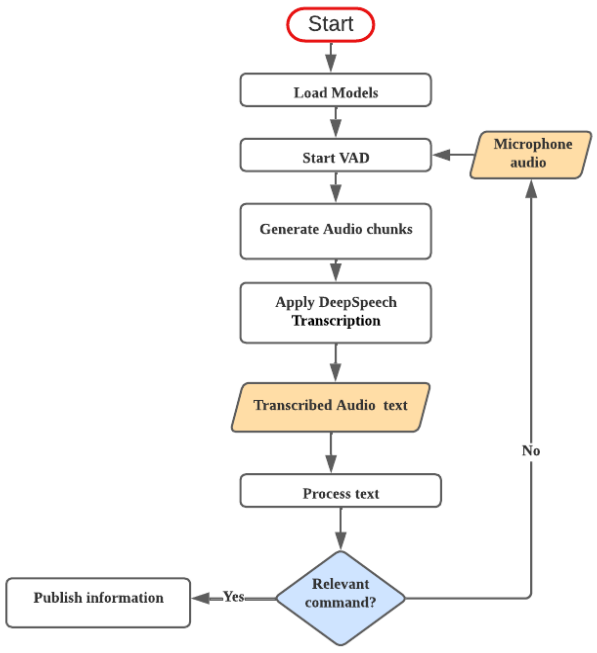


Fig. 10. Flowchart demonstrating sequence of DeepSpeech implementation

The flowchart in figure 10 shows the implemented structure with DeepSpeech. Voice recognition with DeepSpeech is used after the activation word has been detected. The user would instruct the robot to go to any location or room. The first step includes loading the acoustic and language models. Next, an instance of class VAD audio will be initialized to take in raw microphone input. The audio data will then be broken into chunks to enable real-time transcription. The transcribed audio data will be converted to text and passed to another function to check the relevance of the command issued by the user. This process includes:

- Checking if string has indication of user wanting to go to location
- If true, checking the location specified (room number or another location)

Finally, results will be published to several nodes for path planning, navigation and voice assistance. The desired location will be transmitted to the navigationNode to mark the destination and the voiceAssistance node will also update the user of the status of the request.

G. *Voice Assistance*

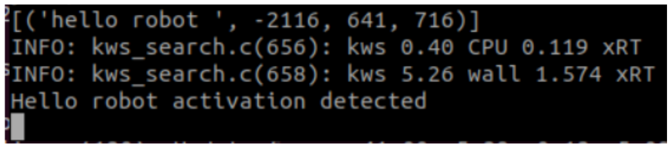
The robot is designed to reply to the user, not only execute the command. This will help the user know that the robot listened to the command and allows the robot to give further instruction about navigation. Espeak is used for the implementation of voice assistance and it showed the best performance and results on Linux based systems. The voice assistance node is continuously listening to 4 different nodes to give the user the relevant feedback. Once a request or message is received by any of those topics, the module will process the request to find the suitable text output that the robot will have to speak. Finally, the text output will be passed to the Espeak command line utility to execute and translate the text to the speech.

V. Results

In this section, the results and testing of various procedures is explained. The robot was tested on 4 different settings in the building. Various other voice recognition and assistance tools where tested and the results will be explained.

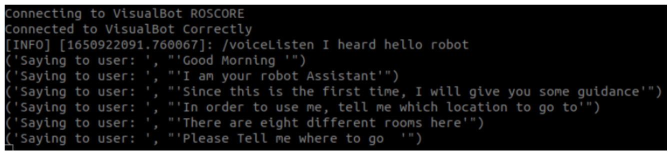
A. *Testing Scenario*

The user is initially inside the lab and wants to go to the exit door of the lab. Therefore, the user will start with activating the robot as shown in the transcription output in figure 11. The robot will greet the user and give some introduction as shown in figure 12. The user will then proceed to tell the robot of the desired location and the DeepSpeech module will recognize the command as shown in figure 13.



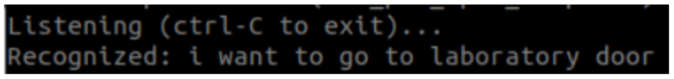
```
[('hello robot ', -2116, 641, 716)]
INFO: kws_search.c(656): kws 0.40 CPU 0.119 xRT
INFO: kws_search.c(658): kws 5.26 wall 1.574 xRT
Hello robot activation detected
```

Fig. 11. Transcription output of using activation word



```
Connecting to VisualBot ROSCORE
Connected to VisualBot Correctly
[INFO] [1650922091.760067]: /voiceListen I heard hello robot
('Saying to user: ', "'Good Morning '")
('Saying to user: ', "'I am your robot Assistant'")
('Saying to user: ', "'Since this is the first time, I will give you some guidance'")
('Saying to user: ', "'In order to use me, tell me which location to go to'")
('Saying to user: ', "'There are eight different rooms here'")
('Saying to user: ', "'Please Tell me where to go  '")
```

Fig. 12. Voice assistance output when interacting with user



```
Listening (ctrl-C to exit)...
Recognized: i want to go to laboratory door
```

Fig. 13. Transcription output using DeepSpeech

A plan will be generated to navigate to the location as shown in figure 14 and the robot will also alert the user to the detected turn and objects detected along the path as shown in figure 15.

Once the robot reaches the location, the CNN model and weights for object detection are loaded and the camera stream

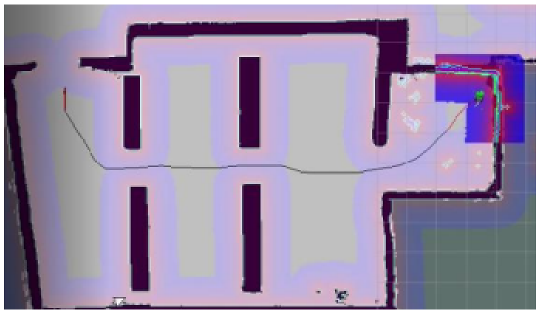
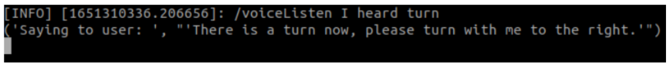


Fig. 14. Path generated for chosen destination on map



```
[INFO] [1651310336.206656]: /voiceListen I heard turn
('Saying to user: ', "'There is a turn now, please turn with me to the right.'")
```

Fig. 15. Voice assistance output informing the user of a turn detected

is opened. The GPU took 20 seconds to load and detect the object. The door is detected as closed as shown in figure 16.

The robot finally instructs the user that a closed door was detected and that the user should proceed to open the door as shown in figure 17.

VI. Conclusion

In conclusion, we designed and tested a prototype for an autonomous robot capable of navigating and avoiding obstacles in a SLAM based map based on SLAM. The robot can recognize voice instructions from the user and execute the commands to guide the user to his/her destination. Moreover, the robot aids the user through voice to tell him/her about turns, objects, obstacles in the way, and the status of doors. All is done using Bluetooth speakers and microphones and an offline mode which allows our design to be more flexible and less dependent on WiFi for functionality. In future work, we want to use a larger and more stable robot design such as the MIT mini-cheetah or the TurtleBot2 robot.To improve the quality and reliability of voice assistance, there might be a need to create our own separate models from DeepSpeech and PocketSphinx to train the system to better recognize the indented communication. Finally, a camera can be added to help track the state of the user and dynamically plan to stop and ask the user to follow.

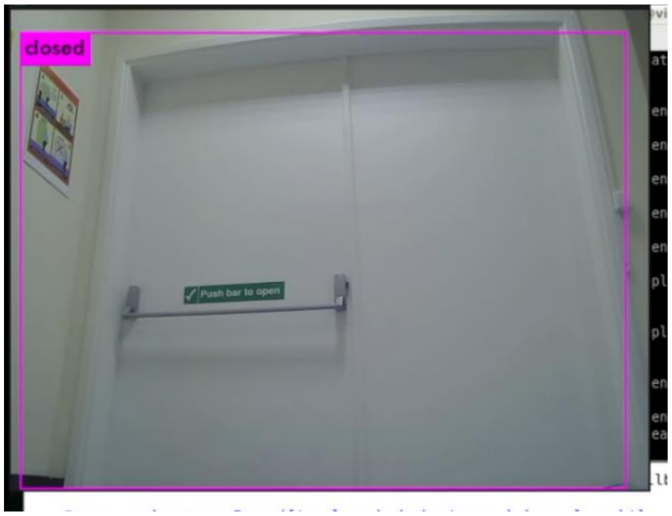
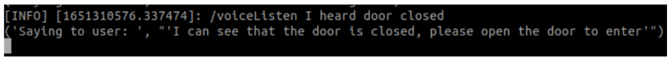


Fig. 16. Detected object with the bounding box indicating the status



```
[INFO] [1651310576.337474]: /voiceListen I heard door closed
('Saying to user: ', "'I can see that the door is closed, please open the door to enter'")
```

Fig. 17. Voice assistance output informing the user of the closed door

References

[1] World Health Organization, "Blindness and Vision Impairment," Who.int, Oct. 11, 2021. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.

[2] W.Jeamwatthanachai, M. Wald, and G. Wills, "Indoor navigation by blind people: Behaviors and challenges in unfamiliar spaces and buildings," British Journal of Visual Impairment, vol. 37, no. 2, pp. 140–153, 2019. Available online: https://timothysun.com/resources/navindor/indoor-navigation-bvji.pdf

[3] Kanak Manjari, Madhushi Verma, Gaurav Singal, A survey on Assistive Technology for visually impaired, Internet of Things, Volume 11, 2020, 100188, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2020.100188.

[4] J. Guerreiro, D. Sato, S. Asakawa, H. Dong, K. M. Kitani, and C. Asakawa, "CaBot: Designing and Evaluating an Autonomous Navigation Robot for Blind People," The 21st International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS '19, 2019, doi: 10.1145/3308561.3353771.

[5] K. Tobita, K. Sagayama, and H. Ogawa, "Examination of a Guidance Robot for Visually Impaired People," Journal of Robotics and Mechatronics, vol. 29, no. 4, pp. 720–727, Aug. 2017, doi: 10.20965/jrm.2017.p0720.

[6] J. Zhu et al., "An Edge Computing Platform of Guide-dog Robot for Visually Impaired," 2019 IEEE 14th International Symposium on Autonomous Decentralized System (ISADS), Apr. 2019, doi: 10.1109/isads45777.2019.9155620.

[7] S. Kayukawa, T. Ishihara, H. Takagi, S. Morishima, and C. Asakawa, "BlindPilot: A Robotic Local Navigation System that Leads Blind People to a Landmark Object," Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, Apr. 2020, doi: 10.1145/3334480.3382925.

[8] A. Xiao, W. Tong, L. Yang, J. Zeng, Z. Li, and K. Sreenath, "Robotic Guide Dog: Leading a human with leash-guided hybrid physical interaction," arXiv.org, 28-Jun-2021. [Online]. Available: https://arxiv.org/abs/2103.14300. [Accessed: 04-Oct-2021].

[9] T.-K. Chuang et al., "Deep Trail-Following Robotic Guide Dog in Pedestrian Environments for People who are Blind and Visually Impaired - Learning from Virtual and Real Worlds," 2018 IEEE International Conference on Robotics and Automation (ICRA), May 2018, doi: 10.1109/icra.2018.8460994.

[10] G. Galatas, C. McMurrough, G. L. Mariottini, and F. Makedon, "eyeDog," Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments - PETRA '11, 2011, doi: 10.1145/2141622.2141691.

[11] D. Sato et al., "NavCog3 in the Wild," ACM Transactions on Accessible Computing, vol. 12, no. 3, pp. 1–30, Sep. 2019, doi: 10.1145/3340319.

[12] C. Tsirmpas, A. Rompas, O. Fokou, and D. Koutsouris, "An indoor navigation system for visually impaired and elderly people based on Radio Frequency Identification (RFID)," Information Sciences, vol. 320, pp. 288–305, Nov. 2015, doi:10.1016/j.ins.2014.08.011.

[13] M. Avila Soto, M. Funk, M. Hoppe, R. Boldt, K. Wolf, and N. Henze, "DroneNavigator," Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility, Oct. 2017, doi: 10.1145/3132525.3132556.

[14] M. Filipenko and I. Afanasyev, "Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment," IEEE Xplore, 2018. https://ieeexplore.ieee.org/abstract/document/8710464/

[15] Q. Zou, Q. Sun, L. Chen, B. Nie, and Q. Li, "A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles," IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, pp. 6907–6921, Jul. 2022, doi: 10.1109/tits.2021.3063477.

[16] mozilla, "mozilla/DeepSpeech," GitHub, Dec. 13, 2019. https://github.com/mozilla/DeepSpeech