EchoLocation Bot: Synthesis of existing technology for commercial personal robots

Carsten Binz[1], Carl Hildebrandt[1] and Meriel Stein[1]

***Abstract*— The popularity of Amazon Alexa devices suggests the need for a user-friendly robot with voice-recognition and voice-locating abilities. In this paper, we present a proof-of-concept for an Alexa device supported by a mobile robotic base that can follow its user's voice while recognizing and avoiding obstacles in its path. This device has the potential for many different threads of future work.**

I. INTRODUCTION

The increasing average age of the world population[8] in recent years has effected a growing need to delegate care giving. Thankfully, robots are becoming more and more ubiquitous in daily life. With the advent of helper bots like the Amazon Echo, Roomba, Care-o-bot, and PR2 appearing in peoples homes, the opportunity presents itself to combine these technologies into more powerful aids, allowing physically impaired users to eschew the need for assisted living facilities or reliance on friends and family members to conduct their affairs [5] [1].

Combinations of these available technologies could have numerous applications. In this paper we present EchoLocation, a mobile robot which tracks and follows a sound source. The robot does this while avoiding obstacles. This robot could find application in search and rescue, aiding people with physical impairments, and other assistive and care-giving fields.

II. DESIGN

The original proposal had to be modified as development of the robot progressed. The modifications were for numerous reasons, such as limitations in Amazons API. The original overview and final overview of the design are discussed below in Section II-A and II-B respectively.

A. Initial Overview

The plan at the outset was to tap into the Amazon Echo Dot API for its voice recognition and voice locator utilities to allow the Turtlebot to listen, follow, and take commands from a nearby user. Its voice recognition processes natural language into recognized commands. Its voice locator uses a circular microphone array to find the angle the voice is coming from, then displays a white light inside a blue light ring indicating that direction. Ideally, Turtlebot would hear the user from across a room or even several rooms away and follow the voice to provide assistance, filtering out noise in the voice locator measurement and avoiding obstacles in its path.

Originally the user would be able to give voice commands to the robot. In order to achieve this one needs to create a custom skill on Amazon Web Services (AWS) [3]. During development of creating a custom skill it was noted that in order to send data outside of the AWS ecosystem, one needs an AWS developer account to create an AWS Lambda[4]. AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of the Amazon Web Services. On top of the cost and limited API, Amazon also requires that data be sent to AWS in regular intervals for security. Constantly uploading data to AWS hinders our ability to continuously listen. For these reasons, and the fact this was beyond the scope of the course, creating a continuously listening voice command skill was removed from the project plan.

B. Final Overview

In the course of modifying the project plan to fit design constraints, the most limiting aspect of the design became the Amazon Echo itself. To replace getting the direction of the voice from the AWS API, a camera was used. Image processing and recognition was employed to process the blue light display and retrieve the angle of the users location in relation to the Echo.

The continuous listening skill was replaced with an infinite loop skill which asks a user the same question on an infinite loop. This did not remove the entire problem as after a set amount of time, the skill still had to upload the data causing the blue ring to light up completely (removing the ability to track the source of the sound). Additionally, the voice locator was prone to noise. Because it was designed as a user feedback interface and not meant to be a source data for controls feedback, its accuracy presented a challenge. The capabilities of the Echo and access to them had the largest effect on our final implementation.

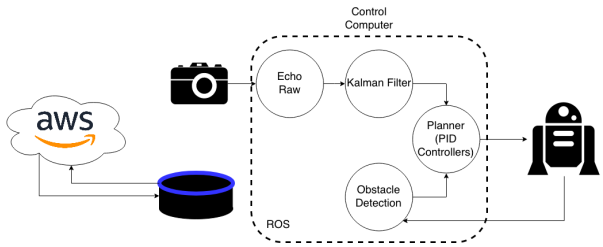

Fig. 1. The final design overview

In an effort to correct the inaccuracies from the Amazon Echo, a Kalman filter was added to the voice locator. The Kalman filter tracked the voice and gave a better approximation of the angle. The original proposal included the

robot avoiding obstacles. This was implemented and tested in the final design. The original proposal also used PID controllers to control the angle and velocity. Both of these controllers were implemented, tested, and integrated into the final design.

III. IMPLEMENTATION

A. Physical Design

In order to process the voice locator of the Amazon Echo Dot, a camera was attached to a structure build over the Amazon Echo Dot. This structure allowed the camera to be a constant height and position above the Amazon Echo Dot.

The camera was connected to the computer running the "echo_raw" and "filter" nodes using a USB cable. This computer ran all code needed to do image processing. The speed the angle of the sounds location was generated was tantamount to the robots success. Thus image processing and filtering were run on a faster Macbook Pro. The Amazon Echo Dot requires a $2A$, $5V$ power source to run, so a normal laptop USB port could not supply the current required to power the device. Thus the Amazon Echo Dot was connected to an onboard USB power supply.

Testing noted that the light intensity given off by the Amazon Echo Dot overwhelmed the camera. Thus digital filtering could not differentiate between the difference in brightness of various points on the LED ring. A physical filter was thus installed between the camera lens and Amazon Echo Dot. An inexpensive pair of sunglasses retrofitted to the camera provided the physical filter required.

B. Echo Raw

The "echo_raw" node was designed to take a camera image and convert it to a raw angle. Capturing the raw image was done using OpenCV. OpenCV is an open source image processing library popular both with the image processing and robotics communities [7]. The goal of this node was to track the brighter LED. Once the location of the brighter LED was located we could then use that to calculate an angle. Removing noise and unwanted regions of the image were done in deliberate stages described below:

Stage 1: *Removing excess blue*. The Amazon Echo Dot emits a blue ring when it tracks sound. It however emits a brighter white led where the sound is coming from. Thus we wanted to remove as much of the periphery blue ring as possible to remove noise from the final image. Due to the ring being blue we removed the blue color by setting the blue matrix in the RGB image to 0.

Stage 2: *Blur the image*. The next step was done in order to further reduce noise. We did not want to have any small patches identified as bright light. We also knew that the final light source would contain a large section of bright light. Thus to remove any small points in the image identified as bright light the image was blurred.

Stage 3: *Convert the image to Grayscale*. The next step was to convert the final image to an image with only pixel values between 0 and 255. That would allow us to easily identify between bright and dark sections in the image.

Stage 4: *Threshold the binary image*. The final stage was passing the image through a threshold to create a binary image. Values below the threshold were 0 and values above the threshold were 1. The threshold did not have to be set dynamically as we had built the camera onto a custom frame and the robot was always used in the same lighting conditions.

Stage 5: *Calculating the center of the bright region*. The final stage was calculating the center point of a group of pixels of unknown shape. This was done by finding the X-Y coordinates of each pixel identified as bright and saving it to a list. The average of this list could then be computed, which gave the center of the X-Y coordinates. A graphical representation of how this works is shown below in Figure 2:

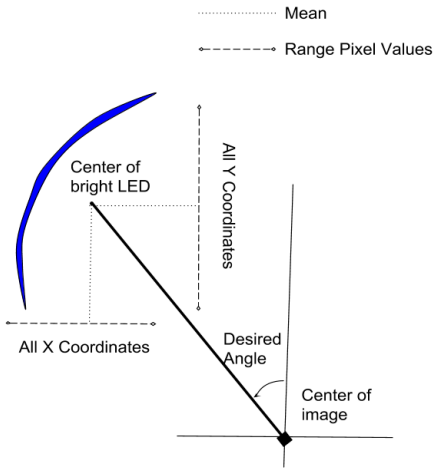

Fig. 2. Calculating of the LED on the Amazon Dot

Now that the image processing was done and the center of the bright pixels found, a final step of calculating the angle was required. This can be done by converting the given probelm to a trigonometry problem and using the arc-tangent.

C. Kalman Filter

Since the camera readings as well as the indicated direction from the amazon echo are noisy we implemented a Kalman Filter. The Kalman filter gave us a better approximation of the angle generated by the "echo_raw" node described in Section III-B. The Kalman Filter estimates the current angle based on the raw value it receives from the "echo_raw" node and the current angular velocity according to the prediction equation 1. Once the prediction is done, the Kalman filter updates its prediction using the update equations shown in Equation 2

$$\begin{aligned} \hat{X}^{-} &= A\hat{X}_{k-1} + Bu_k \\ P_k^{-} &= AP_{k-1}A^T + Q \end{aligned} \tag{1}$$

$$\begin{aligned} K_k &= P_k^{-}H^T\left(HP_k^{-}H^T + R\right)^{-1} \\ \hat{X}_k &= \hat{X}_k^{-} + K_k\left(z_k - H\hat{x}_k^{-}\right) \\ P_k &= \left(I - K_kH\right)P_k^{-} \end{aligned} \tag{2}$$

For the first initialization we set the angle xk-to zero.

D. Obstacle Detection

The obstacle detection node subscribes to the laser data given by the Turtlebot. This data is used to determine distance and angle of all solid-body obstacles. The distance of the obstacles were distilled into three categories represented by an enum: **FAR** for a distance of greater than 1.1m, **CLOSE** for a distance smaller than 0.8m, and **MEDIUM** for anything in the 0.8m to 1.1m range.

Nearby objects were given a greater priority of avoidance. The angle of obstacles was determined by the place in the data array the values were coming from. The angle of the obstacle was done by finding which side of the array the obstacle falls. Once this was known, the obstacle detection node output this using three categories represented by an enum: **LEFT**, **RIGHT** and **STRAIGHT**.

The obstacle detector encounters difficulty with detecting and avoiding semi-permeable objects, such as nets like the one hanging in the Autosoft Lab. In boundary cases like these, the Turtlebot's onboard 3D infrared camera cannot be counted on for reliable obstacle detection.

E. Planner

The planner took in three values in the form of two messages. It took in a filtered angle from the Kalman filter in the form of a Float64 standard ROS message. It also took in two integer values both of the form Int8. Both Int8 values were grouped together into a single custom message called "ObstacleData". This allowed for cleaner communication between the Planner and the obstacle avoidance.

The planner used this input to calculate two outputs, namely angular velocity and linear velocity. Both the angular and the linear velocity were implemented using PID controllers. That allowed the outputs to be smooth and continuous. The implementations of both controllers is described below:

1) Angular Controller: The angular controller was designed to minimize the error between the goal heading and the desired heading. This seems simple, however difficulties arose when considering the different frames of the robot and Amazon Echo Dot. The robot calculated its current heading using data received from the odometry sensor data. The robots kept two positions, its current position and previous position. Converting this to a trigonometry problem, the angle between the two points can be calculated using a right angle triangle and arc-tangent function.

The desired heading was calculated as described in Section III-B. Simply minimizing the error between the current heading and goal heading using a PID did not work. This was due to the two angles not sharing the same frame of reference. This problem is depicted below in the Figure 3. Given $time = 0$ the error between current heading and goal heading would be 45, causing the PID to minimize this error. Thus the robot would correctly turn the required 45 degrees. At $time = 1$ the robot would continue to drive straight rather than turn, as the error between current heading and goal heading would be 0. This is caused by the difference in frame. The Amazon Echo Dot was attached to the robot, and as the robot rotated the Amazons reference frame would rotate to.

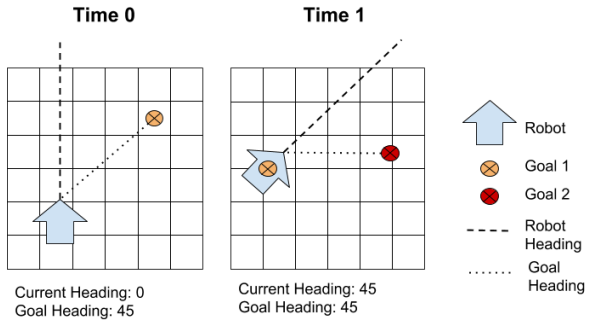

Fig. 3. Determining current heading and desired heading according to local frame

The fix for this was simple. Due to the Amazon Dot being attached to the top of the robot, the current heading of the robot is always 0 relative to the Amazon Dot. Thus the current heading of the robot could be set to 0.

Given an obstacle to avoid, the planner would add 45 degrees to the goal heading. This was done using Equation 3 below:

$$goal\_heading = goal\_heading + (direction \times 45) \quad (3)$$

In Equation 3 the "goal_heading" is calculated as "goal_heading" added with the direction. This equation worked due to the Direction enum implementation. Left was set to the value -1, right was set to 1 and straight set to 0.

2) Velocity PID: The velocity PID controller was much simpler. The velocity controller minimizes the error between the current speed of the robot and the desired speed. The current speed was taken from the robots odometry sensors. The robots desired speed was set based off the obstacle distance given by the obstacle detection node. The pseudocode for the desired speed is given below in Algorithm 1:

Algorithm 1: Pseudo code for getting the desired velocity

```
if Obstacle Distance ≥ 1.7 then
    Desired Speed = 0.3;
else if Obstacle Distance ≤ 0.7 then
    Desired Speed = 0.1;
else
    Desired Speed = 0.2;
end
```

F. Network Design

The original design included connecting the AWS in order to get information from the Amazon dot. We would have

been able to circumvent getting the angle through image processing by simply getting the angle through Amazons API. However, it was noted that Amazon does not give access to the angle information [2].

Progress towards the original network design had already been made and is briefly described below. The original implementation is shown below in Figure 4. The Amazon Echo Dot would connect to the self implemented skill on the AWS. The AWS would then send a JSON file to a web server. This web server would respond with the appropriate JSON information to tell Alexa what to say back to the user. The webserver would forward any useful commands to the control computer.

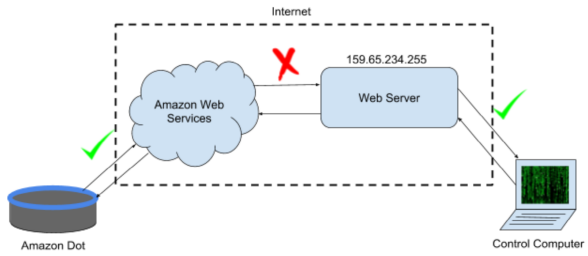

Fig. 4. Original network design marked with what was complete and what was not

A Linux server was created on Digital Ocean a cloud hosting company [6]. A simple web server was created in Node-JS which opened a socket on port 4242. This web server was run on the Linux server created on Digital Ocean. A python script was created which was able to connect to this webserver on port 4242 and exchange information.

For the communication between the Amazon Dot and AWS, a custom skill was started. The skill was a simple hello world skill. However in order to allow for communication to be sent outside of AWS, an account needed to be created with AWS. Due to the recent finding that the angle API was not available, this approach was put on hold, and the image processing in Section III-B developed.

IV. RESULTS

A. Raw Readings

Using image processing to determine the where the LED's where brightest worked as expected.

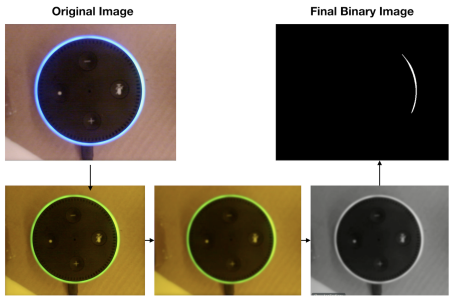

Fig. 5. All the steps of the image processing starting from the original image and ending with the binary image.

The first step in the image processing was getting the binary image of the light. Each of the steps used to obtain the final binary image are shown in Figure 5

The binary image was then used to calculate the angle from the center of the image as described in Section III-B. Due to the actual angle not being given, no ground truth was established. Thus visual inspection was used to determine the accuracy. Examples from the final calculation are shown below in Figure 6.

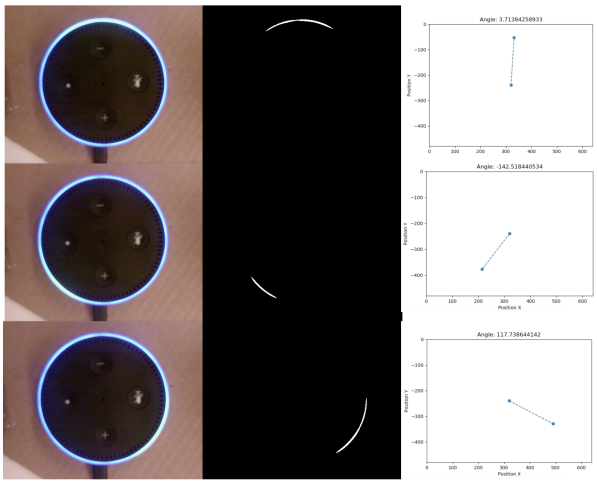

Fig. 6. Final angle calculation showing both the original image and binary image.

B. Kalman Filter

The plot in Figure 7 shows the estimation of the Kalman filter (blue line) as well as the raw noisy input signal (red line). We tested the Kalman filter using artifical input which simulated noise. The Kalman filter reduces the noisy parts effectively and drastically reduces the impact of outliers.

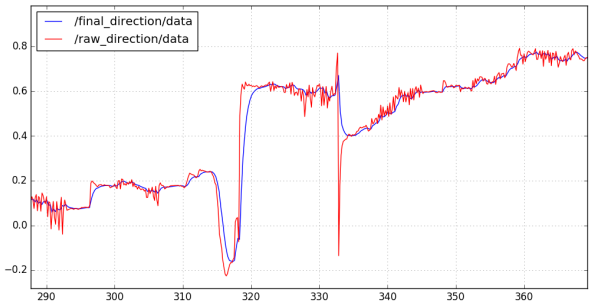

Fig. 7. Kalman filtered data

The graph in Figure 8 shows the effect of the Kalman filter on real data. For the test we walked around the Amazon Dot in a circular motion to see if it was able to correctly detect the direction from which the voice originated. As this graph makes evident, the red line of the raw data is noisy and the resolution is low. However, the jumps from -π to π are intentional. The Kalman filter is able to smooth the low resolution as well as the noise and simultaneously reacts fast

enough to the jumps with minimal oscillation. We realized that we could future reduce noise by evenly distributing the angles at an offset from zero ensures that the robot will not have to normalize its turning direction according to the angle given by the Amazon Echo Dot. This means that the jump from -π to π would be smooth. However this was not implemented in the final robot as walking behind the robot would be difficult as it would turn as you tried to get behind it.

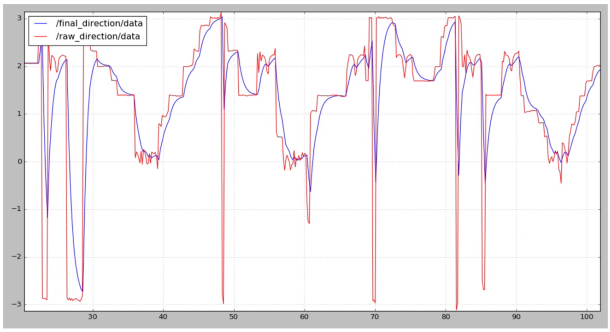

Fig. 8. Kalman filtered Amazon Echo Dot angle in practical deployment

C. Following Voice

A series of tests were performed in which the autonomous robot tracked and drove towards a source of sound. The autonomous robot was able to successfully navigate towards the sound source. The sound source did not need to be stationary and was able to move about freely. There were a few noted deficiencies, however they can be attributed to the Amazon Echo Dot. The Amazon Echo Dot is designed to upload information to AWS in regular intervals. This uploading sometimes threw the current goal angle of the robot off. Another deficiency was due to the rate at which the Amazon dot updates it's angle. Both of these however were out of our control and thus were ignored. The final results are shown in the video in Section VI.

D. Obstacle Avoidance

Obstacle avoidance was tested in isolation to determine if it was working correctly. A series of tests were developed in which the robots $goal\_heading$ was set to 0. That means the robot was told to drive straight continuously. We noted that the robot avoided most obstacles including walls. There were two scenarios in which the robots success was not guaranteed.

The first situation was when the robot found a semi-permeable obstacle such as a net. The robot would generate left and right commands at random as different parts of the net were detected. The continuous left and right movements resulted in a net straight motion. This could have been fixed by simply adding a flag for how long it should turn a certain direction. However due to this being a very specific environmental problem, all nets were simply removed.

The second situation in which the robot would fail was when the obstacle could not be avoided with a 45 degree turning angle. This could have been fixed by calibrating the the view angle and calculating the obstacles position. The larger the obstacle and closer to the center the harder the robot would turn. However due to this not being the main focus of our project we assumed that the obstacle would remain a constant size and that 45 degrees was sufficient.

Obstacle avoidance tests are shown in the video described in Section VI.

E. Full Robot

Once all sections of the robot had been tested in isolation they were combined. The robot was tested vigorously by having a person walk around and having the robot track the person. An obstacle was also randomly inserted and removed. The robot was mostly successful at tracking the human voice. Deficiencies were attributed to the Amazon Echo Dot generating an incorrect heading. This happened frequently when the robot drove near a wall and the humans voice echoed off the wall. The robot would then turn towards the new sound source (the echo from the wall), recognized the wall as an obstacle and turn away from the wall before correcting itself.

Other cases were when physical sounds such as a robot driving over a metal cap caused the robot to track an incorrect sound source. Due to these problems being related to Amazon Echo Dots tracking features, they were ignored. The results and corner cases are shown in the video described in Section VI.

V. CONCLUSION

A. Feasibility

This project can be thought of as a feasibility study for the integration of an existing, stationary "helper bot" – the Amazon Dot – into a mobile system. We give proof that Amazon Alexa integration is workable even in a hacked-together implementation, and that a mobile Alexa device is a logical next step for expanding its functionality.

Furthermore, this extension of the existing product has a high possibility for widespread adoption and frequent use, as Amazon is already so prevalent and sells secondary Alexa devices for multiple rooms in one house. While these secondary devices are not specifically marketed for people with disabilities, the EchoLocation robot would find a target market among those in need of assistance and/or unbroken connectivity to care-giving or lifesaving services regardless of where they are in their homes. A search-and-rescue robot with sound-following and voice-locating capabilities would be able to locate survivors in burning buildings or wreckage without jeopardizing the lives of rescuers. The wide range of use cases for a mobile Alexa extends to many corners of the already-existing market for Alexa devices as well as broadens it.

B. Future Work

The EchoLocation bot can be extended to take in commands from Alexa to manipulate the various features of the robot it rides on. For example, it could control the following

behaviors of its mobile base, such as how closely it follows its user, or whether it should enable the camera based on its location in the house. Additionally, it could manipulate a robot's grippers at a high level, such as those like PR2 has.

Incorporating the Alexa voice control into existing personal assistant robots such as PR2 and Care-O-Bot would arguably be the most useful contribution to the overall functionality of the robot, as well as the fact that the Alexa has access to plenty of its user's data already. So, functions like "call a person from the user's contacts" or "play music from playlist X" would not need to be implemented by the developer, and would be intuitive to interact with on the part of the user. This would help ease the steep learning curve that comes with new technology, especially technology involving new hardware. Additionally, the Amazon Echo Dot could be equipped with security functionality to supplement the "smart home" ideas it is based on and allow Amazon to compete with Nest and similar companies.

For these to happen, the research community would need cooperation with Amazon to open up the Alexa API for callbacks and extension. Of course, these robots could be extended using the principles of the Amazon Echo Dot. Given the attitude of Amazon towards its own proprietary information, it may be easier for robotics community to simply construct their own microphone arrays to track the voice so that the Amazon Dot isnt needed anymore.

VI. APPENDIX

All code can be found in the git repository at the URL:

`https://github.com/hildebrandt-carl/EchoLocation`

A video containing the final results can be found on youtube at the URL:

`https://youtu.be/jPWCccZzm4U`

REFERENCES

[1] Amazon Alexa - user guide. https://www.amazon.com/b?node=17934671011. Accessed: 2018-12-06.

[2] Amazon Developer Forums - how to get direction of voice. https://forums.developer.amazon.com/questions/163449/how-to-get-direction-of-voice.html?childToView=165518#answer-165518. Accessed: 2018-12-10.

[3] Amazon Web Services. https://aws.amazon.com/. Accessed: 2018-12-11.

[4] Amazon Web Services - lambda. https://aws.amazon.com/lambda/. Accessed: 2018-12-11.

[5] Business Insider - robot caregivers for the elderly could be just 10 years away. https://www.businessinsider.com/robot-caregivers-for-the-elderly-10-years-away-2017-8. Accessed: 2018-12-09.

[6] Digital Ocean - cloud computing. https://www.digitalocean.com/. Accessed: 2018-12-10.

[7] OpenCV - open source image processing. https://opencv.org. Accessed: 2018-12-06.

[8] World Population Growth. https://ourworldindata.org/world-population-growth. Accessed: 2018-12-11.