RoboCare - A Human Robot Communication Framework To Care For The Sick

Tanmay Agarwal Kathan Shah Muntaqim Mehtaz

***Abstract*—There is an increasing need to care for sick individuals in their homes and allow for contact-free media of interaction. We hope to create a human-robot communication framework using the Tiago robot to create functionalities to fetch and deliver objects. With the goal of developing hand gestures and voice instructions as methods of communication with the robot, we discuss our results and further work with the simulation, vision and voice components of the project.**

I. Introduction

Care-taking robots have been around for a long time and their functions have varied from caring for the elderly, doing common household tasks and cleaning/vacuuming the house. With the pandemic outbreak of the novel Coronavirus, as hospital infrastructure gets overloaded, many potentially infected individuals have to quarantine in their homes. This poses a problem as some members living in the apartment might be healthy, but coming into frequent contact with the infected members will increase their risk of contracting the virus. Furthermore, due to an unprecedented shortage of Personal Protective Equipment (PPE) for hospital staff, increased contact with patients poses an increasing risk of staff contracting the disease themselves. Therefore, there is a need to create methods for contact-free care in hospitals by automating the most basic tasks of fetching and delivering objects so as to keep only the essential tasks for the hospital staff to perform in-person.

In order to solve this problem, our goal is to create a human-robot communication framework that can be used in the above-described settings to deliver items to the infected member, fetch items from the infected member and search for items as instructed. In order to communicate with the robot, we hope to use hand gestures and voice communication. By using an Amazon Alexa device to handle the voice communication requests, we not only eliminate the need to create a Natural Language Processing architecture, but also allow for remote operation of the robot. This way, an individual living in a particular location could use an Alexa device to communicate with the robot and care for another individual in a different geographic location. Secondly, by using hand-gestures to communicate with the robot allows for a quick, non-verbal mode of communication that is also independent of any network communication. Such a medium of human-robot communication would be particularly helpful in remote areas where an internet connection is unreliable or absent.

II. Methodology

A. Task

We used a Tiago robot simulation in Gazebo for our purposes since it has a gripper, possesses the capability to move around and is well-documented with tutorials on basic navigation, Moveit! and OpenCV, all three of which would be required for our project. We have identified two primary methods for communication in our project, voice commands and visual commands. Packages used include Moveit!, OpenCV and Alexa-app along with the associated repositories for the Tiago robot, cloud communication and the ROS Navigation stack. In entirety, the repository uses C++, Python and Node.js to implement the outlined HRI framework.

B. Simulation configuration

1) Simulation Environment: In order to simulate a suitable simulation environment, we modeled an apartment with four rooms in Gazebo using its Building Editor. Models for tables, a person and objects for pick up were also added to the simulation to serve as common objects and obstacles that the robot would be expected to encounter, and also serve as objects for detection and manipulation to be used later. While intending to model the house after an apartment, we also created the structure to be generalizable to other environments, such as nursing homes, houses or hospital wings.

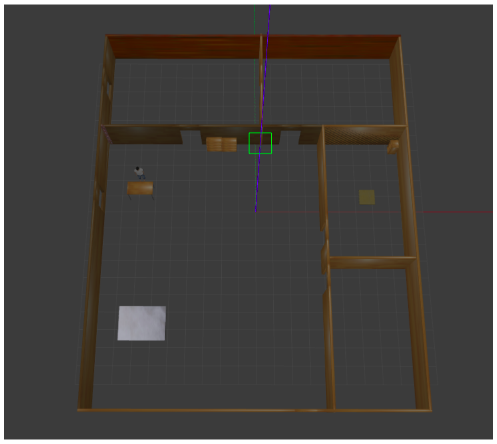

Fig. 1. Simulation Environment

2) Communication System: While creating a system of ROS Nodes and topics to implement the outlined HRI framework, it was essential to create a system that could:

- Take inputs from both hand gestures and voice commands
- Queue multiple instructions appropriately and prevent overlapping in execution
- Be modular so as to allow any future modes of communication to be added in easily

In order to achieve the above objectives, we created a system represented in “Fig. 2”. Each mode of communication subscribes to a topic that carries messages of its own unique type. A callback function tailored to handle this particular data type then handles this request and calls another function called manageInputs. The manageInputs function takes an integer representing a room number to go to, a string representing an object to search for, and another integer representing a room number to go to. Finally, this function then converts these inputs into navigation or search commands and communicates with Tiago’s internal nodes to carry out the intended tasks. All of these subscribers are then encapsulated into one node, named room_nav in our implementation.

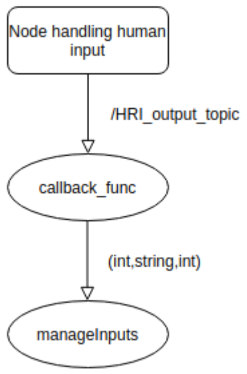

Fig. 2. Communication Schematic

This system has the following advantages:

- Since any mode of communication eventually deconstructs the input to (int,string,int) and calls the manageInputs function, this system is able to take inputs from any mode of communication, so long as its subscriber’s callback function is capable of deconstructing it into (int,string,int) format.
- Since any communication method’s callback function eventually calls the manageInputs function, multiple communication requests are queued with respective subscribers and they call manageInputs in the same order as the requests were received .Furthermore, since execution won’t stop till manageInputs has stopped executing in entirety, while multiple requests can queue with subscribers, they cannot overlap in execution with other requests
- It is simple to add a new mode of communication to this framework. All we would need is a node to parse the request, publish it on a topic with its unique message type. We would then add a new subscriber to this topic and a callback function that deconstructs this request into (int,string,int) format and calls manageInputs with these values.

“Fig. 3” diagrammatically shows this communication framework model

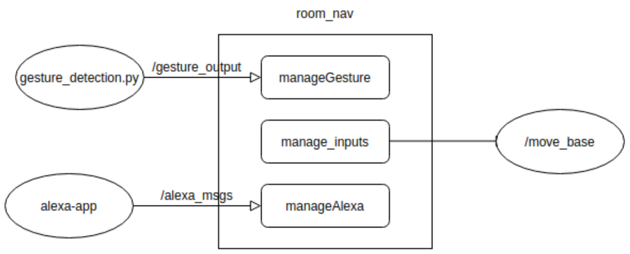

Fig. 3. Node Communication Schematic

3) Navigation: A critical component of the implementation is for the robot to move from one location to another as indicated by the human interactor. In order to accomplish this, we created a dispatch mechanism that takes an integer room number as input, decodes that into specific co-ordinates and sends the navigation request to the ROS navigation stack. These co-ordinates are predefined and serve as locations that are easy to reach inside a room, which once accomplished, can be followed by more complex tasks in the same room. This enables a path to easily exist between rooms and the ROS Navigation stack does not fail in finding a path to the room itself (whilst it might fail to perform the subsequent tasks). We chose to utilize the ROS Navigation stack since it implements sophisticated algorithms for path planning and obstacle avoidance that utilize the robot’s on board LiDAR and RGB-D camera. We create a client in the room_nav node that sends these requests to the move_base server and checks if the goal has been reached or not.

4) Object Search: Another aspect of the framework is to search for a particular object and report if it exists or not. In order to simplify object detection, we chose to place Aruco tags on search objects and instead of searching for the objects, we search for a particular marker_id on an Aruco tag. For simplicity, our current implementation only searches for aruco tags with marker_id of 582 (model named aruco_cube_5cm). Another aruco cube (model named aruco_cube_8cm) is provided in the Gazebo simulation, which has a tag with marker_id = 26. This serves to simulate an object that we want to detect, and another that we do not intend to detect. The system can then be tested on false positive or false negative detection too. Furthermore, using Aruco tags has benefits for object grasping which will be discussed later.
For the scope of our project, we limit our search space to the two tables in the central room. The algorithm is structured as a loop, best illustrated in “Fig. 4”.

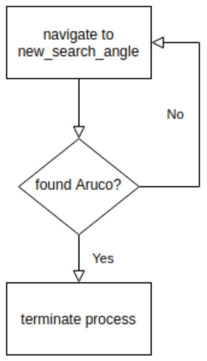

Fig. 4. Aruco Tag Loop Structure

In this process, we visit a table and search for the object. If the object is not found, we visit the table at a different angle. If it is still not found then we move to the next table and repeat the process. We encountered some challenges in this part of the implementation, particularly with detecting aruco tags. Three variables determine if the object will be detected or not in our framework:

- Object height relative to Tiago's camera
  For the aruco tag to be detected, it has to be visible to Tiago's camera comfortably. This includes not just its position in the camera's field of view but its orientation with respect to the camera too since the aruco tag is placed on the top face of the object. Therefore, when the object is kept on a table and itself has some height, if it is tall enough then it is possible that the tag is inside the camera's field of view but is too oblique to be detected
- Detection during or after motion
  In our current model, we only check if the aruco tag is found or not once we reach a particular position around the table. However, detection of the aruco tag occurs constantly through the execution of another node. This can create a situation in which the tag was found in a previous location, but since the robot was navigating to another location, it only realizes that a tag was found once it reaches that new position. Since the output from the aruco tag detector is a pose relative to Tiago's camera, there is no way to know where in the map the tag was detected.
- Position of the object on the table
  Depending on where the object is placed on the table itself, we can run into the issues described above. Since search positions around the table are pre-defined, detection of the aruco tag is inherently dependent on the position of the object on the table too

Given the challenges above, while the current system is capable of carrying out basic object detection, there is a large scope for error and the system needs to be adapted to account for a varied number of scenarios.

5) Object Manipulation: The last piece of the project includes grasping an object after detection. Having Aruco tags on the objects simplifies the inputs by removing the need for contour/size detection and provides an easy method for calculating the pose of the tag relative to the robot's camera. The disadvantage to this system is that the exact dimensions of the object need to be known beforehand, without which, the aruco tag itself does not provide sufficient information for object manipulation. In order to realize this, we implemented Tiago's Pick and place tutorial in our simulation environment but encountered challenges due to which the robot was never able to grasp the object

- Errors in localization and pose estimation
  Contrary to the tutorial, our robot not only manipulates objects in its environment but also moves in the environment itself. This leads to errors in the robot's localization which compounds with the pose estimation of the aruco tag. As a result, even though an Aruco tag is detected, its pose estimated and the inverse kinematics solved, the robot is not able to grasp the object since it miscalculated the pose.
- Randomness and non-uniformity in simulation environment
  In the pick and place tutorial, before picking up an object, the robot moves its arm into a predefined position that is "suitable" for grasping objects in front of it. This movement does not take into account any obstacles that might be in front of it. This can lead to situations in which the robot's arm collides with nearby obstacles and disrupts the environment. "Fig. 5" shows an example in which the robot's arm got stuck under the table while it was trying to move to the initial grasping location. Therefore, this system is not fully generalizable

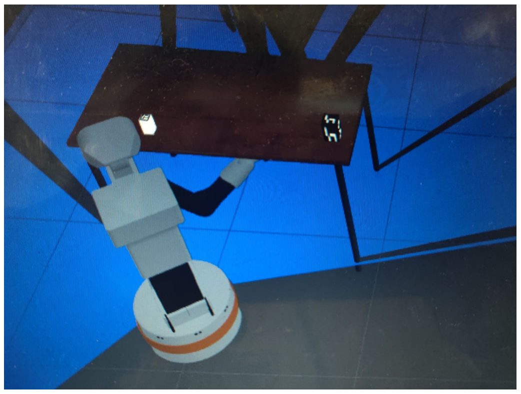

Fig. 5. Robot's arm stuck under the table

Given the system requirements and the challenges above, object manipulation is theoretically possible and the code to do so is also included in the repository, however grasping the objects in the simulation environment proved to be a difficult task.

C. Alexa Commands

An alexa app [2] API was used to create a new alexa skill that recognises commands and intent requests. The app functionality exists in index.js file. Currently it can handle inputs like room number and object name. Alexa app recognises the intent and then publishes to a topic, then the room_nav node subscribes to this topic which in turn translates the message into an instruction consumed by the robot. A sample command would be “Alexa, ask robo care to go to room 3”, this will cause the robot to move and navigate to a room which is designated the number 3. The communication happens over a port. A rosbridge websocket is initialized on which the alexa-app-server can latch onto, to transmit messages. This simple server-client system is enough if we want to test the app’s functionality locally. If we want to take advantage of Alexa’s cloud functionality we will need to connect the alexa skills kit (running remotely) with the alexa-app server (running locally). For this purpose Bespoke tunnel [3] was used to forward the local server with a public URL. You can also use a ngrok tunnel to achieve the same functionality but Bespoke was chosen due its better Alexa related documentation. Once the communication is set up we can send voice commands to our robot from anywhere in the world as long as the Alexa and Alexa Skills Kit(ASK) are running on the same account. In “Fig. 6” you can see a visual schematic on how this communication works.

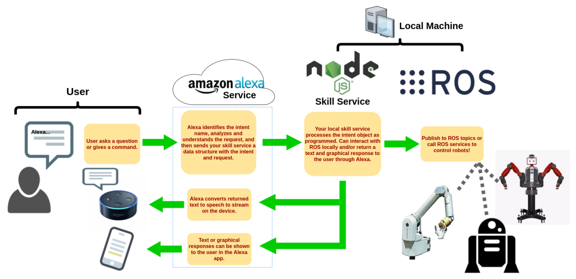
Local Machine
User
amazon alexa Service
node JS
Skill Service
ROS
Alexa...
User asks a question or gives a command.
Alexa identifies the intent name, analyzes and understands the request, and then sends your skill service a data structure with the intent and request.
Your local skill service processes the intent object as programmed. Can interact with ROS locally and/or return a text and graphical response to the user through Alexa.
Publish to ROS topics or call ROS services to control robots!
Alexa converts returned text to speech to stream on the device.
Text or graphical responses can be shown to the user in the Alexa app.

Fig. 6. Alexa Communication Schematic

D. Hand Gestures

Computer vision algorithm was utilized to detect hand gestures with a goal of having the robot understand that holding up one finger translates to integer one, inviting the robot to “Go to room one”. The gesture detection algorithm was rendered to a ROS node that initiates it through the webcam that publishes an integer room number as output to a topic. The algorithm essentially works by counting the number of contours in the input image. While this approach is theoretically correct for detecting the number of fingers, since the algorithm does not distinguish between what is a hand and what is not, it also reports output from counting contours in any object. Furthermore, lighting, background uniformity and camera quality play an essential role in the accurate determination of hand gestures.

III. Results

A. Voice Commands

We are taking advantage of Amazon Alexa’s Natural Language Processing(NLP) prowess. It can recognise any intent from a list of specified utterances. Some of the examples of utterances that our Alexa skill can handle currently include “Alexa, ask robo care to go to room number”, “Alexa, ask robo care to find object”, “Alexa, ask robo care to go to room number and find object and deliver to room number”. The only foreseeable downside is Alexa’s understanding of accents. It sometimes mis-resolves words spoken in a non American accent. Furthermore, edge cases have not been handled in our project, i.e., if a room or an object does not exist, Alexa does not understand how to respond to such a scenario. Alexa has a timeout counter of eight seconds, which means if it does not receive a response back in eight seconds, it will timeout and close the app. This may cause some issues when the internet speed is slow and the message takes longer to travel between the cloud and the local server.

B. Vision Accuracy

“Fig. 7” shows some examples of correct and incorrect detection of hand gestures by our vision algorithm. Accuracy is fairly variable and depends on multiple factors. The contour detection method has some success and can be considered for situations where computational power is limited as the overhead for this algorithm is small.

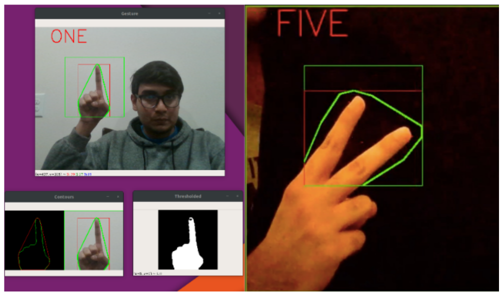

Fig. 7. Correct vs Incorrect Hand Gesture Detection

C. Navigation and Object Detection

The navigation nodes were able to work appropriately with the hand gesture and Alexa communication nodes. Most importantly, we were able to achieve cloud communication between an alexa device and the gazebo simulation, therefore allowing for the robot to be controlled remotely. This was tested through a zoom call in which one member gave an alexa command on their device, and 1 mile away, our robot received the request and navigated to the appropriate room. Communication with the hand gesture node works well too, but accuracy is inherently dependent on the accuracy of the gesture detection algorithm.
The object detection algorithm is also functioning well when the conditions for object height, orientations and position on the table are regulated. The manipulation procedure also initiates but fails to grasp the object. When the above parameters are not controlled, success with object detection is varied and will need to be assessed on a case by case basis. It was also observed that running the navigation nodes and object detection node along with the manipulation node was

computationally too expensive for the available machine and results were fairly inconsistent. Due to the absence of a capable graphics driver, RViz would either fail to load entirely, or some of its components would fail to load independently. As a result, identifying, troubleshooting and solving errors was made fairly difficult, as a result of which we decided to keep object manipulation as a part of further work to be done when more computational power is available.

IV. Conclusion

Our project's primary objective was to create an interface for contact-less operation of a robot. In light of the global pandemic due an infectious virus contact-less interaction has become of paramount importance. This task requires numerous considerations such as safety of human beings in the robot's vicinity, robustness and portability of the interface. This project mainly focused on creating an interface that can understand voice and gesture commands and relay the instructions to a robot. In a household setting this type of an interface can be used to safely provide food, water and other items to someone who is sick without putting other members of the family at risk. In a larger setting this interface can be extended to multiple robots that can deliver or retrieve objects from quarantined patients. We have been able to demonstrate that Alexa voice commands can be used to remotely instruct the Tiago robot to navigate to different locations. Hand-gestures can also be achieved to do the same and would be useful in situations where network connections are unreliable. Lastly, detecting objects with Aruco tags has some success but better models can account for more generalizable solutions.

V. Further Work

While we were able to successfully demonstrate robot navigation through human-robot communication, more work needs to be done on detecting objects and manipulating them. While a better system for object search will allow for a greater number of situations to result in successful object detection, we can even consider not using Aruco tags, but instead utilizing deep learning vision algorithms to detect objects. In this case, the onboard RGB-D camera or the LiDAR sensor can be used to estimate object depth. Secondly, in the presence of sufficient computational resources, object manipulation can be explored further. The suggested path would be to not follow the Tiago tutorial on picking and placing objects as that system will have limited success in many simulation environments. Instead, creating a node from scratch that utilizes Moveit! to find inverse kinematics solutions and does not move into a predefined grasp position could prove to be more successful. For voice commands, we will need to publish the Alexa skill to Amazon so that it can get approved after meeting all the security criteria so that anyone can simply download the skill and start using it right away without much hassle. Thirdly, in order to improve hand-gesture detection, one can test using deep learning vision based algorithms that train on images of particular hand gestures so that the algorithm works solely on images of hands. Furthermore, in depth research needs to be done to find out whether such a system is safe to operate and will it actually be effective in slowing the spread of the virus in real life. On top of the research, a more scalable and robust system needs to be created that can handle such important tasks in a bigger setting, such as hospitals and nursing homes.

References

[1] M. Deyo, ”Alexa App Server”, https://github.com/mdeyo/alexa-node-ROS, January 2020

[2] ”Bespoken Proxy Tunnel”, https://bespoken.io/

[3] ”TIAGo bot documentation”, http://wiki.ros.org/Robots/TIAGo

[4] ”Hand Gesture Recognition Repository”, https://github.com/codeingschool/Hand-Gestures