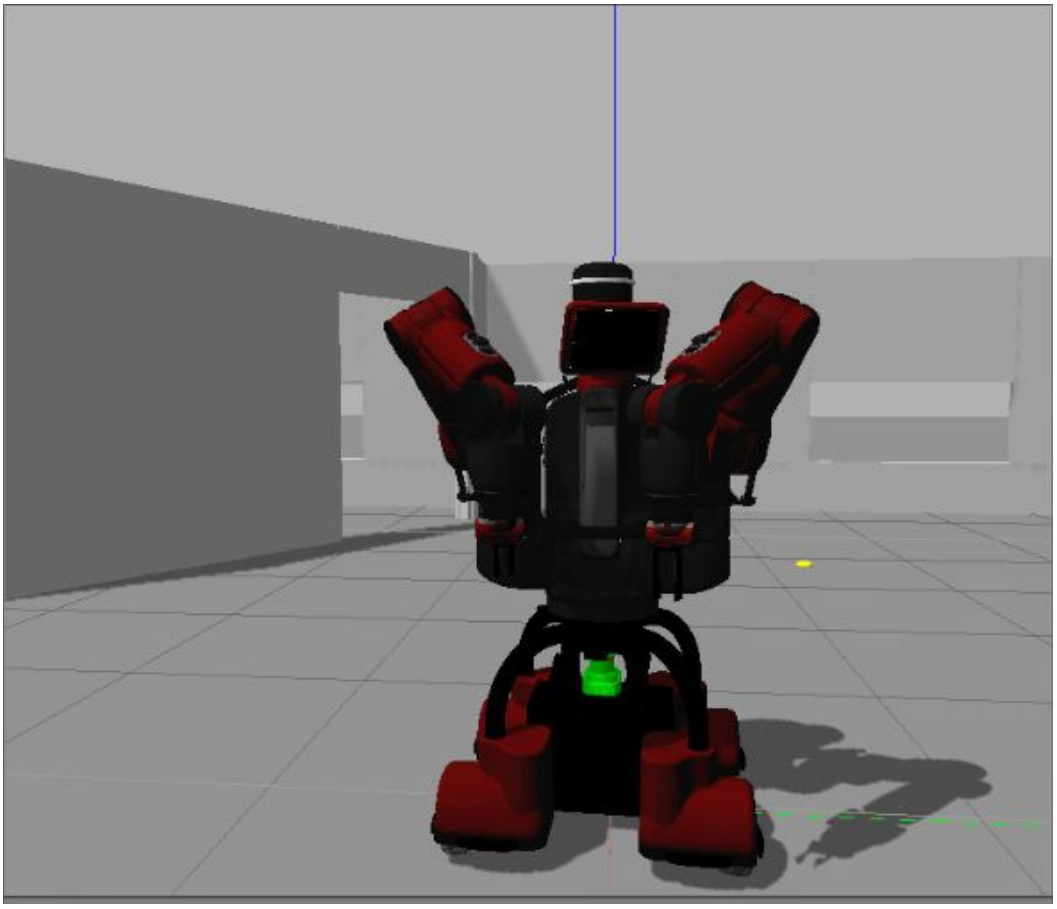
Robot Delivery Team - Report



Version: Draft01
Date: 25 May 2020
Sponsor: James Harland
Supervisor: Ian Peake
Author: Simon Lay s3658769
Timothy Gunardi Teguh s3639374
Samual Brown s3545852

Table of Contents

Robot Delivery Team - Project Charter

Team Members: Simon (s3658769), Timothy (s3639374), Sam (s3545852)
Supervisors: Ian Peake, James Harland

Purpose of Project

We have approached Ian Peake and James Harland to create a connection between Rosie, the Baxter robot attached to a mobility base, and Blue, the MIR100 robot, for the purpose of delivery. However, a different project was proposed by us. Instead of a delivery team, the goal of this project will be to create a team of doormen. Rosie and Blue will act as doormen for VXLab's guests.

Project Sponsor

The Project sponsor is Professor James Harland from Computer Science and Software Engineering, director of the Virtual Experiences Laboratory. Our Supervisor and support is Ian Peake is the Technical Manager of the Virtual Experiences Laboratory

Objective

The Team has brainstormed and has decided to operate Rosie, the Baxter robot, with an attached mobility base for movement, to open the door for any guest to the VX lab via voice. It is important to note that when we refer to Rosie, we mean the integration of the Baxter robot itself with this mobility base. For brevity, we will continue to refer to this specific integration of the Baxter robot and the mobility base as Rosie. Blue, the MiR100 robot can also be told to look out for any guests who may be outside and notify Rosie to open the door, as well as be able to carry the guest's bag and follow them until it is dismissed.

Risks

- Rosie might not be able to open the door and just topple over if not calibrated correctly.
- Blue could be in the way of Rosie while she's moving to open the door for the guest.
- The door button might not work as intended, which may cause several bugs
- The NLP may not be properly configured and a manual intervention might be required through the command line.
- Coronavirus
- Rosie might fall on someone
- Rosie might hit something by accident
- Blue might collide into someone
- We may find that the sensors on Rosie and Blue cannot operate reliably through the door glass. In this case we will have to alter the objective slightly so that the doormen (Rosie and Blue) can only be instructed to open the door from inside the VXLab.

Benefits

- Blue and Rosie will be able to automatically welcome a guest to the VXLab via permission from a human.
- This will become an autonomous process which can allow people in the VXLab to just stay seated while a robot attends the door.
- Can relieve guests from back strain from the bag they have been carrying if Blue can carry and follow the guest until it is dismissed.

Overview

The Scope of this project is rather large. Therefore, the team has decided to split the work into phases. We are planning on starting the first sprint in week 4. However as the team is very new to the ROS technology we cannot guarantee the solution to be completely robust in all situations.

We are going to be working on a minimal viable product to serve as a proof of concept for our project.

Purpose of Project - Revisited

We have approached Ian Peake and James Harland to create a connection between Rosie, the Baxter robot and Blue, the MiR100 robot, for the purpose of delivery. However, a different project was proposed by us. Instead of a delivery team, the goal of this project will be to create a team of doormen. Rosie and Blue will act as doormen for VXLab's guests.

The implementation of this doormen for the VXLab's guests seemed like an achievable task. So we also thought of goals beyond Rosie and Blue becoming a team of doormen, and utilizing blue as a bag handler who could carry the bag of the guest if they accepted the request from Rosie. This would showcase the communication between the two robots and thus completing one of the tasks Dr. Ian Peake, James Harland and the team were expecting to complete. The process of communicating between the two robots required being able to be physically present as the team would be able to get a more accurate result from the environmental behaviours of the VXLab.

As of the recent outbreak of COVID-19. The method to our solution was no longer possible to complete the project through being physically present. Instead the team had a discussion to work on the project online through the simulator that Ian Peake has showcased us as one of the options to work on at home then implement them to rosie when we use the lab again. However the only way to progress with the project under the new restrictions was to work in the simulator completely with the hope we'd get access to the lab again in the future.

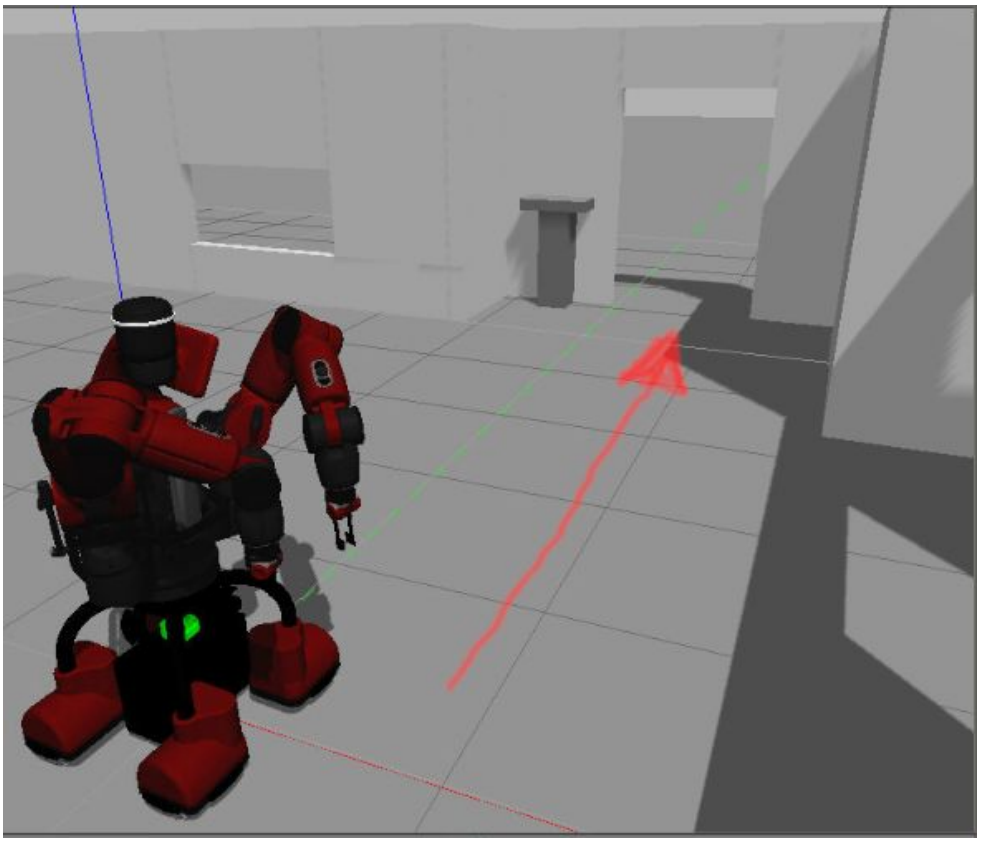
The shift in our workflow impacts our initial plan of practice and has caused us to pivot with the initial requirements we proposed. The team had to do another analysis of the project plans and goals and then present them to our stakeholders once more.

The purpose of this project has slightly changed to make a process of opening doors without needing human intervention to be done through the simulator. Throughout the project development, the team has discovered that the simulator wasn't able to completely replicate reality which posed further problems. Therefore, after discussing with our supervisor and client they stated that they are satisfied with the end product of Rosie being able to approach the door alone and push the button inside the simulator.

Objective - Revisited

Rosie, the Baxter robot, is humanoid, while Blue, the MiR100 robot, is essentially a coffee table with wheels. Due to this difference in shape, we believe they are able to complement each other in this task. Each has a wide set of strengths and weaknesses. In general, Rosie is capable of performing more complex tasks, such as pushing a button, but is not very mobile with her mobility base. In contrast, Blue is quite mobile and compact, and is capable of getting to different places quickly and safely compared to Rosie, but Blue lacks the arms to push the button, and is also too short to do anything to the button. Not to mention the fact that blue can't pick anything up. Hence, we believe they both compliment each other.

The Team has brainstormed about each of their strengths and weaknesses, and has decided that Rosie, the Baxter robot, should open the door for any guest to the VX lab via voice, by pushing a button. Blue, the MiR100 robot can also be told to look out for any guests who may be outside and notify Rosie to open the door, as well as be able to carry the guest's bag and follow them until it is dismissed.



Risks - Revisited

- Rosie might not be able to open the door and just topple over if not calibrated correctly, as the humanoid shape is rather imbalanced when compared to Blue.
- Blue could be in the way of Rosie while she's moving to open the door for the guest.
- The door button might not work as intended, which may cause several bugs
- The NLP may not be properly configured and a manual intervention might be required through the command line.
- Coronavirus restrictions
- Rosie might fall on someone
- Rosie might hit something by accident
- Blue might collide into someone, as it is short and sometimes unnoticed by people.
- We may find that the sensors on Rosie and Blue cannot operate reliably through the door glass. In this case we will have to alter the objective slightly so that the doormen (Rosie and Blue) can only be instructed to open the door from inside the VXLab.

Benefits - Revisited

- Blue and Rosie will be able to automatically welcome a guest to the VXLab via permission from a human.
- This will become an autonomous process which can allow people in the VXLab to just stay seated while a robot attends the door.
- Can relieve guests from back strain from the bag they have been carrying if Blue can carry and follow the guest until it is dismissed.
- Ability to work on the VXLab projects without physically being there

Stakeholders and End Users

The Baxter bot that runs in the simulator called “Gazebo” is still in a development stage where not all of the functionality exists. To compensate for the functionality that’s been missing, our supervisor Ian Peake has been committing his time to improving the simulator which assists us in completing our project.
The Stakeholders include:

- James Harland
- Ian Peake
- Future Students who will be using the simulator in future if there is a similar scenario like the pandemic
- Other future users who might want to use the baxter bot through the simulator

Project Team Members

Simon Lay	Development Team Member
Timothy Gunardi Teguh	Development Team Member
Samual Brown	Development Team Member

Overview - Revisited

The Scope of this project is rather large. Therefore the team has decided to split the work into phases. We are planning on starting the first sprint in week 4. However, as the team is very new to the ROS technology we cannot guarantee the solution to be completely robust in all situations.

We are going to be working on a minimal viable product to serve as a proof of concept for our project.

Methodology and Approach

The Project will be using an agile scrum methodology which enables us to move fast and react to changes in scope. This allows the team to make changes when we encounter issues with the simulator.

Project Governance

There is a weekly meeting with the supervisor and client where we discuss the project direction and current stages of the implementations. We use this time to bring up any issues that we've discovered when working on our tasks and publish our tasks to the trello board that we use to keep track of our progress. After this meeting, we usually hold an after meeting between the team members. We discuss what each person will be doing for the week, and what should our goals be for the next meeting, as well as what needs to be done before then.

As the capability of working online has become the primary method of meeting the supervisor and client. We use Microsoft Teams, Slack, and Facebook Messenger to communicate and arrange meetings. Originally, we used Zoom for online meetings, but we changed to Microsoft Teams eventually due to the policy surrounding the use of Microsoft Teams over Zoom. Demoing our project is available through screen sharing.

Scope and Deliverables

Before the Pandemic we had a plan to complete the project, in broken down chunks called phases. The phases focus on the implementation of rosie and how the project would have expanded. The phases below as follows:

Original Phases:

- Phase 1: Rosie moves to door and opens the door
- Phase 2: Blue is able to wait for guests and notify Rosie if there are any
- Phase 3: Rosie greets and asks if the person would like to have their bag carried
- Phase 4: Blue would get the bag object placed on its dock and follow the user while Rosie moves back to her charging station.

We have taken the first phase of our scope and expanded it instead, due to the Pandemic forcing us to adjust our workspace, which will delay the delivery of this product. By focusing on one phase, the product will be more polished, especially given the current situation of the MiR100 robot model not being available in the simulator. Working with the simulator is also very different to working physically, and we have considered this fact. We believe that due to this change, our goals must be changed due to the amount of time that will be spent on adapting to the simulator.

New Phases:

- Phase 1: Rosie moves to door and opens the door
- Phase 2: Rosie waves and greets the person
- Phase 3: Rosie is able to converse with the person like a therapist

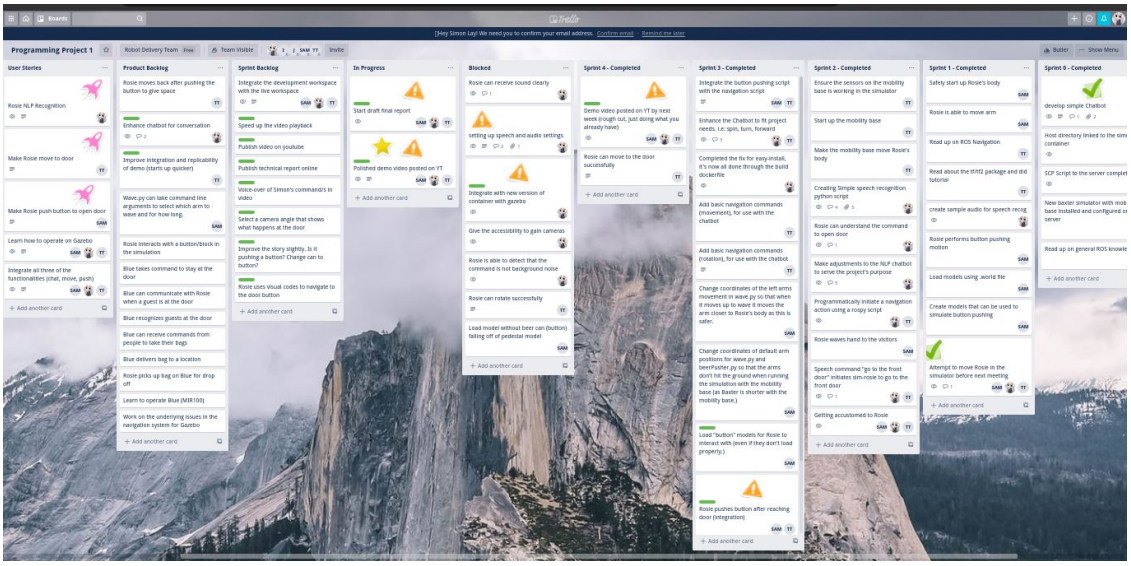
This project has 2 main deliverables that are specified by our client and supervisor. They include:

1. Rosie moving to the front of the door in the simulated vxlab
2. Rosie opening the door in the simulated vxlab

With these two core deliverables in place we can achieve the first phase of our proposed project and expand it with the new phases that we altered.

Infrastructure

Trello:
https://trello.com/b/PHEBrqCE/programming-project-1



Github: https://github.com/rmit-s3658769-Simon-Lay/RobotDeliveryTeam

Google Drive:
https://drive.google.com/drive/u/1/folders/1xCJMlgBEdHrb-J7c1HNshUqaGVfLbjFt

Technical Overview

Initial Tools Summary

Tool: Rosie, the Baxter robot
Description:
It is used for simple industrial jobs such as loading, unloading, sorting, and handling of materials. Designed and created by Rethink Robotics.

Baxter Robot

Tool: Mobility Base
Description:
A component designed to be attached to robots such as the Baxter robot so that it can move collectively. Designed and created by DataspeedInc.

Mobility Base

Tool: Blue the MiR100 robot
Description:
A cost-effective mobile robot that quickly automates your internal transportation and logistics. Designed and created by Mobile Industrial Robots. Together with the Baxter robot, they collectively form the Rosie we know.

MiR100 Robot

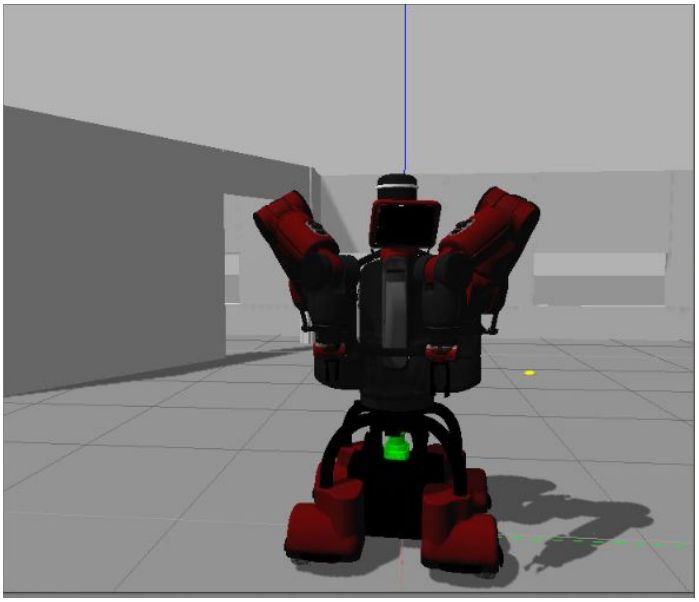
Current Tools Summary

Tool: Gazebo
Description:

An open source 3d Robotics simulator. This simulator enables Rosie to be simulated so the team can work on the project online.

Gazebo

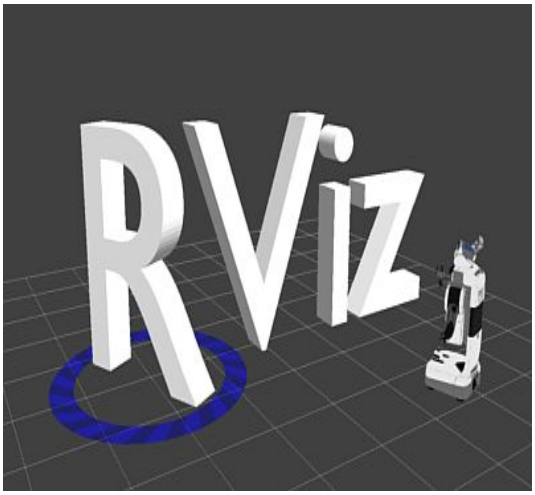
Tool: Simulated Baxter Bot
Description:
A simulated version of Rosie the Baxter Bot, including the mobility base, that's been put together by Dr. Ian Peake. This implementation of Rosie does not have its full functionality thus making it difficult for us to build the product.



The Baxter Robot with the Mobility Base in Gazebo

Tool: RViz
Description:
Visualization tool designed for Robot Operating System. This tool is used to help with the navigation of Rosie so then she won't collide into other objects such as the

VXLab wall.



RViz

Tool: Robot Operating System (ROS)
Description:
The Robot Operating System. It "is a set of software libraries and tools that help you build robot applications."[1] This is essentially what we use to interface with the robot's hardware. Development was done with the Kinetic Kame distribution of the Robot Operating System.

ROS Kinetic Kame

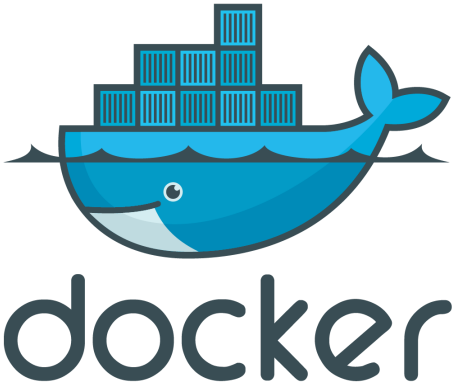
Tool: Emacs
Description:
Editor MACroS is a text editor used by some of the team members during the

[1] *https://www.ros.org/*

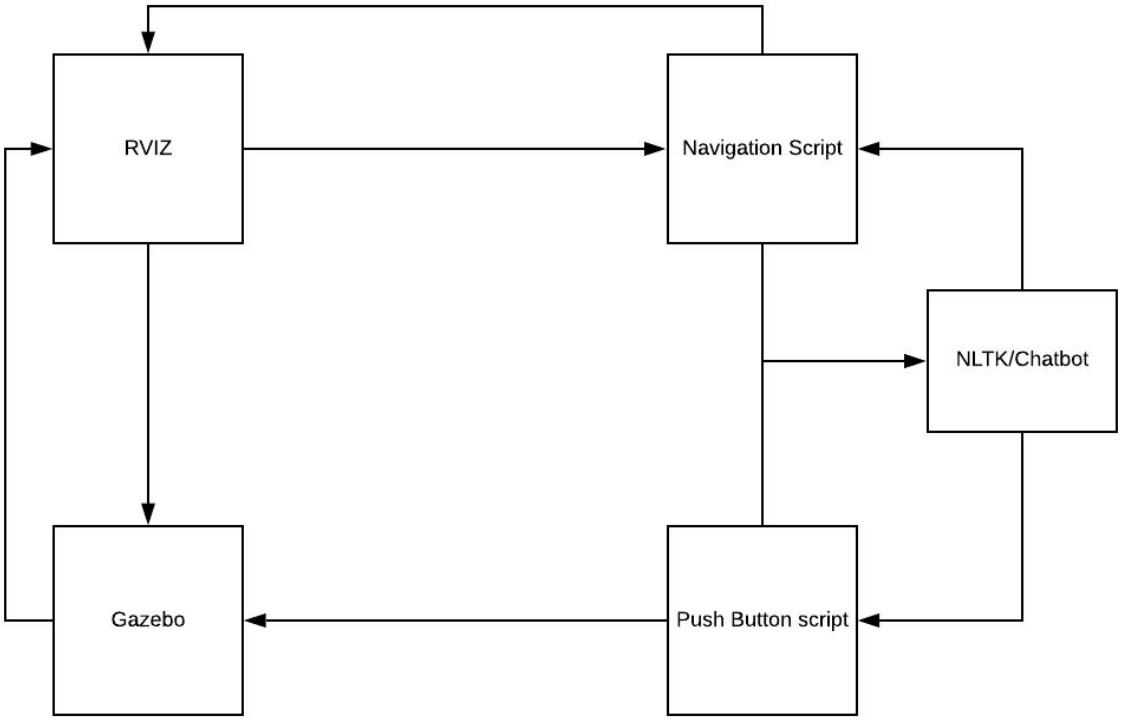
project. It is highly extensible and contains an emacs lisp interpreter.

Emacs

Tool: Docker
Description: A tool used to manage containers.



A Simplified Diagram of Components



This activity above is all executed in the docker container.

Deployment Guide

Access to the Development Server can be achieved by contacting Ian Peake. This is to enable development at remote locations.
The pull our project and follow the instructions in the README.md: https://github.com/rmit-s3658769-Simon-Lay/RobotDeliveryTeam

You would then need to configure your python scripts to be running python2 not python3 as it could cause compatibility issues with the current version of ros installed for the Gazebo simulator.

We used the following script to connect to the server, key (id_rsa) and the ip address may be different:

```
#!/bin/sh
ssh -i ./id_rsa robodeliver@131.170.250.237
```

Running the simulator:

Build the container and workspace using (Requires install of docker-ce on your platform to build and run container):

./build

Run using the container using:

./run

This will start two containers in the background: an X11 display container and a Baxter simulator. Once started, do:

docker ps

To find out the name of the docker container running the Baxter simulator (vxlab-baxter2) image.

Next, do:

docker exec -it vxlab-baxter2 bash

At the prompt, do:

source ./rosenv.sh
./simstart &

Finally, to see the Gazebo simulator window, run a browser on the same machine, with the URL (if you are running the sim on that machine):

http://localhost:8080/vnc_auto.html

However if the sim is running on a remote machine use something like this instead (the IP address may need to be changed.)

http://131.170.250.237:8080/vnc_auto.html

When in the docker container run the following to start the Wobbler example:[2]

./baxter.sh sim
rosrun baxter_examples joint_velocity_wobbler.py

The example scripts are stored in the following path. They may be informative to the reader.

/ws_baxter/src/baxter_examples/scripts/

And possibly most importantly (if the reader is working with a baxter bot) the baxter bot API documentation.

[2] *https://sdk.rethinkrobotics.com/wiki/Simulator_Installation*

docs.ros.org/hydro/api/baxter_interface/html/baxter_interface-module.html

This is the NLTK documentation. NLTK was used for maintaining a conversation with the client.

https://www.nltk.org/_modules/nltk/chat/eliza.html?fbclid=IwAR3RXWJsoAJGC1rOC3cskK0HBeTECdFv49XbXdcTstT4hS5lSqu5E89_qLo

This is the PocketSphinx documentation. PocketSphinx was used for speech recognition.

https://pypi.org/project/pocketsphinx/?fbclid=IwAR0niIPy-xUQuN4YbdawWIaB8HW4tGq7w6TZJIE1g0cGi456vQqtNDbXRec

This is the SpeechRecognition documentation. SpeechRecognition was used with PocketSphinx to convert WAV files to text.

https://pypi.org/project/SpeechRecognition/?fbclid=IwAR0U8kX0tOj2KUPIh7yA5auMONSkGYl50g2F_FEZTRITE2FeG1XItkLrVHY

Features Accepted

where you collect a short description of what were the features that you "accepted" into the backlog and aimed to implement. As for the project charter, it is definitely to your advantage to talk about "failures", that is technical tasks that were not able to be achieved that you worked on in the project;

The features we accepted were carefully monitored to ensure that we iteratively get closer to the project goal. We didn't really accept any features that would deviate this such as working on improvements to the environment. Although it would essentially be beneficial to us as developers. It wouldn't make us develop faster if all of our resources are focused on improving the environment and not the delivery of the project. With constant discussions with our stakeholders the main priorities were:

- Chatbot functionality
- Chatbot being more conversational
- Navigation functionality
- Pushing the button

These features are mentioned in our trello where they are broken up into iterative sprints.

The tasks which we failed in, which were mentioned in our project charter were:

- To have the camera's implemented

- Have audio being able to be recognised and not background noises.
- Rosie assisting the person with their bag
- Deliver the bag to MiR100
- MiR100's interaction with Rosie and the Person.

The failure to achieve these tasks were mainly due to time constraints and the environment not currently supporting the functionality to deliver them.

Testing

Testing was conducted for each of the following components:

Individual Component Testing:

NLTK Chatbot:

- The wav files were created and then tested on our development machine to ensure that the audio files were correct before feeding them to Rosie inside the simulator.
- Functionality of the chatbot was tested before uploading to the server, such as praising wav files correctly and using it to produce an appropriate output.
- Only the speech recognition and output of the wav files could be tested with the chatbot alone.

Navigation:

- Navigation was tested through rerunning the simulator after giving it a given command. The reason for rebuilding the docker container is to reset the variables to their origin.
- Navigation testing was required, as it allows us to transverse rosie towards the door while being aware of the walls of the vxlab.
- Scripts were also tested with the use of RVIZ and the Gazebo simulator. As testing these implementations were just running the scripts. It was acceptable from the team members that we would test like this, as there is no suitable testing harness for the kind of project goal we are trying to achieve.

Push button and wave:

- Similarly to pushing the button. The button was tested with Rosie having the navigation subsystem absent. As the button pushing motion did not rely on the navigation system. It was suitable to have it tested originally where it was developed.
- The testing was required to ensure that it's motion was suitable to push a button in a more realistic environment, so certain actions such as reverting back to the safety position was a priority.

- Testing for the wave to see if the arm does not get caught between a wall or other objects.

All individual testing was within the team's expectations. Including navigation as we know that there was an underlying problem in the navigation subsystem not being accurate. However with discussion with the stakeholders, a suggestion of implementation was the next step even if some components were not functioning perfectly.

Implementation Testing:

The approach we executed was testing the implementation between two components at a time and then finally all of the components together. The first attempt of implementation testing was with the NLTK and the navigation system. The process included the scripts to be located in a folder where the chatbot would be able to call it. When the wav file was fed to the chatbot it then initiated the movement scripts accordingly. If Rosie misses its destination point, a teardown of the environment is needed or a repositioning of Rosie in RVIZ is needed.

Next the implementation involved was Navigation and Push Button. It mainly consisted of testing the scripts together and then manually firing off each script after the next one ends. We also needed to rebuild the environment if Rosie's position did not reach its destination in RVIZ or Gazebo. Which is a similar issue we had to keep revisiting when we executed the scripts.

The final implementation of testing consisted of all 3 components working together. This was mainly handled by the NLTK chatbot, where it would receive the command from the wav file as input and then execute the opening door sequence. If it fails a rerun of the container is required if the Gazebo did not match RVIZ simulator. With the result of testing we were able to thoroughly understand the issues with certain components and delivered them to the client and supervisor, and when given with enough time, we were able to fix these issues to an acceptable state.

Development Guide

When developing remotely one should connect to an X11 desktop running in a container on the remote server (a VXLab blade server in our case) via noVNC. One should also connect to the server via SSH. This way commands can be issued on the remote server. The container can be stopped and started, Gazebo and ROS scripts can be stopped and started, files can be edited and Gazebo can be viewed via the noVNC connection.

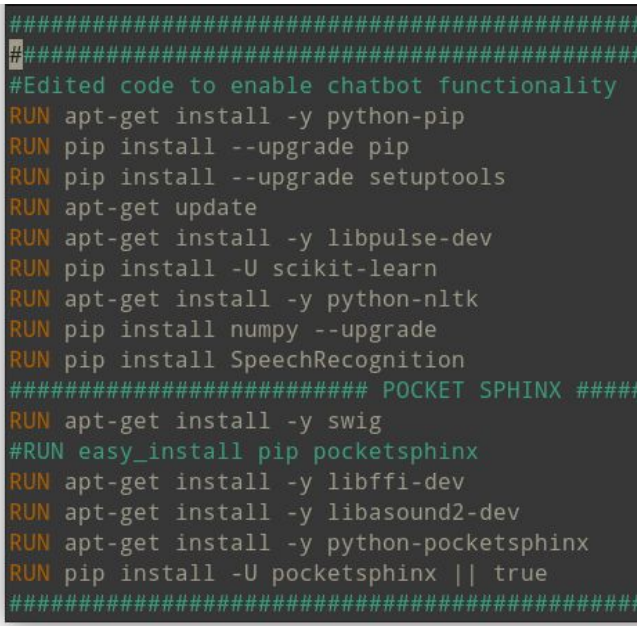
The ROS libraries are compatible with C++ and Python. We used python. Python scripts can be written that use the ROS libraries and then run using the rosrun command. When running scripts with the ros run command and using a set up like the one described above one should be able to see the effects of the script on the robot in the simulator.

Technical Solution

The product we have delivered is able to approach the door and simulate a push button command. The chatbot functionality also works but outputs it to the console.

NLTK Chatbot

A large portion of the team's time was to build and rebuild the dockerfile and ensure all libraries that worked in the client machine could work on the server side. This includes the implementation of audio so then the chatbot can parse and use it as an input. Below is the dockerfile described:

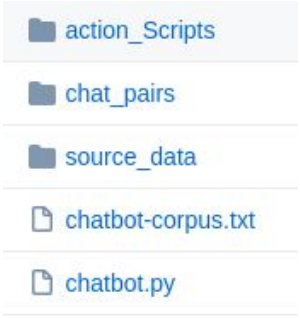


```
##############################################
#############################################
#Edited code to enable chatbot functionality
RUN apt-get install -y python-pip
RUN pip install --upgrade pip
RUN pip install --upgrade setuptools
RUN apt-get update
RUN apt-get install -y libpulse-dev
RUN pip install -U scikit-learn
RUN apt-get install -y python-nltk
RUN pip install numpy --upgrade
RUN pip install SpeechRecognition
######################### POCKET SPHINX #####
RUN apt-get install -y swig
#RUN easy_install pip pocketsphinx
RUN apt-get install -y libffi-dev
RUN apt-get install -y libasound2-dev
RUN apt-get install -y python-pocketsphinx
RUN pip install -U pocketsphinx || true
##############################################
```

The first step when designing the chatbot was to search for the available software that could support a chatbot in python 2.

The code location in Github to locate the NLTK chatbot is:
RobotDeliveryTeam/GithubFile/NLP/speech_recog

The files are split into folders depending on their functionality.



- The folder action_Scripts contains the scripts for Naivation and Pushing the Button.
- The folder chat_pairs contains the therapist chatbot code. The source has been mentioned in the file as the team has only made adjustments to the script to fit the project requirements.
- The folder source_data contains the wav files used to feed the NLTK chatbot.
- Chatbot-corpus.txt is used in the earlier stages of the chatbot which enabled it to user words and give responses based on the corpus

The code extract below outlines the method to parse the wav files from the source data. It is used as a method to parse multiple wav files as we would usually want more than 1 argument.

```
def extract_text_from_mv(name_source):
    r = sr.Recognizer()
    from_wav = sr.AudioFile(name_source)
    input_from_speech=''
    with from_wav as source:
        audio = r.record(source)
    try:
        s =  r.recognize_google(audio)
        return s
    except Exception as e:
        print("Exception: "+str(e))
```

The input controller uses an array for input which then returns an array to the chatbot to parse.

```
def inputs_controller():
    input_from_speech = []
    input_from_speech.append(extract_text_from_mv('source_data/sample_run_sample.wav'))
    input_from_speech.append("Hi")
    input_from_speech.append("I am doing good thanks")
    input_from_speech.append( "pretty good")
    input_from_speech.append(extract_text_from_mv('source_data/sample_bye.wav'))
    #input_from_speech = " ".join(input_from_speech)
    #print(input_from_speech)
    return input_from_speech
```

Baxter commands are handled by this method to execute them. It checks if it's part of a pre configured list of commands. If the script is specific such as EXECUTE random. It would need to be hardcoded. The other commands such as forward and spin are all given hard values for their functionality.

```
def baxter_commands(user_response):
    basic_commands_array = ['forward', 'stop', 'slow', 'spin', 'turn', 'execute']
    commands = user_response.split(" ")
    run_cmd = '*rospy cmd*'
    run_flag = False
    for i in commands:
        if run_flag == False:
            if i not in basic_commands_array:
                return False
            if ('forward' == i):
                print('moving forward....')
                os.system("python action_Scripts/move-original.py -x 5")
            if ('stop' == i):
                print('stopping.......')
                os.system("python action_Scripts/stop.py")
            if ('slow' == i):
                print('slowing down.....')
                os.system("python action_Scripts/slow.py -x 5")
            if ('spin' == i):
                print("spinning........")
                os.system("python action_Scripts/rotate.py -y 45")
            if ('turn' == i):
                print("turning.....")
                os.system("python action_Scripts/rotate.py -y 45")
                ##insert rospy command here
            if ('execute' == i):
                run_flag = True
        else:
            if i not in basic_commands_array:
                run_cmd = run_cmd+" "+i
            if i == commands[-1]:
                print("execute the rospy command: python "+run_cmd)
                #hard coded:
                os.system("python action_Scripts/move-to-door.py -p")
                os.system("python action_Scripts/beerPusher.py")
                os.system("python action_Scripts/waveLikeMade.py")
                os.system("sh ../../../../lift-arms")
                ##insert rospy command here
```

The function below enables the functionality of the chatbot as a whole. It checks if the argument given in the array is a baxter command and if false it uses the chatbot to generate a response. If the argument was a baxter command it will then attempt to trigger it if it is part of the preconfigured options.

```
def Enable_wav():
    print("************** Rosie **************")
    print("=" * 72)
    print("Hello I am Rosie.  How may i help you today?")
    for i in inputs_controller():
        user_response = str(i)
        user_response = user_response.lower()
        print("you said: "+user_response)
        #Insert Baxter bot commands#
        if baxter_commands(user_response) == False:
            ##Conversational
            if(user_response!='bye'):
                print("you said: "+user_response)
                chatbot.rosie_chat(user_response)
            else:
                print("Rosie: Bye! take care..")
                break
```

Pocket sphinx is used for speech recognition in the real world scenario. It was a feature that was already developed by the team. This function was still placed in the code in hopes that we would have access to the vxlab once more. The method can be used as a switch to enable the NLTK chatbot if you want to switch it between the chatbot using wav or live speech.

```
#implement script controls. e.g: rosie execute script move
def Enable_pocketsphinx():
    print("Rosie: If you want to exit, say Bye!")
    for phrase in LiveSpeech():
        print('you said:')
        print(phrase)

        user_response = str(phrase)
        user_response = user_response.lower()
        #Insert Baxter bot commands#
        if baxter_commands(user_response) == False:
            ##Conversational
            if(user_response!='bye'):
                print("you said: "+user_response)
                chatbot.rosie_chat(user_response)
            else:
                print("Rosie: Bye! take care..")
                break
```

The technical solution for the NLTK Chatbot mainly resides in the backend console. This is due to not having access to an output device in the server. So it is not connected to the view in Gazebo or RVIZ. The views would be primarily impacted by Navigation and the Push Button scripts.

Navigation

Developing the Navigation system does not require any notable outside packages. It does include the actionlib and move_base_msgs from the rospy packages. Development is quite dependent on the integration of the mobility base and robot itself. Unfortunately, we do not have the specifics as to how the mobility base was integrated with the Baxter robot. Otherwise, development is rather simple, as the SimpleActionClient from the actionlib takes care of everything. A MoveBaseGoal that you created will be sent to the ActionServer, which will then notify the related components as to what needs to be done. This is essentially calling the 2D Nav Goal in RViz programmatically, and functionally they are similar.

```
client = actionlib.SimpleActionClient('move_base',MoveBaseAction)
rospy.loginfo("Waiting for server...")
client.wait_for_server()

goal = MoveBaseGoal()
goal.target_pose.header.frame_id = "base_link"
goal.target_pose.header.stamp = rospy.Time.now()
if args.flag_cmd == "x":
    goal.target_pose.pose.position.x = 100.0;
    goal.target_pose.pose.position.y = args.y_axis;
elif args.flag_cmd == "y":
    goal.target_pose.pose.position.x = args.x_axis;
    goal.target_pose.pose.position.y = 100.0;
else:
    goal.target_pose.pose.position.x =  args.x_axis;
    goal.target_pose.pose.position.y =  args.y_axis;
goal.target_pose.pose.position.z =  0.0;
goal.target_pose.pose.orientation.x = 0.0;
goal.target_pose.pose.orientation.y = 0.0;
goal.target_pose.pose.orientation.z = 0.0;
goal.target_pose.pose.orientation.w = 1.0;
client.send_goal(goal)
rospy.loginfo("Waiting for response...")
wait = client.wait_for_result()
if not wait:
    rospy.logerr("Action server not available!")
    rospy.signal_shutdown("Action server not available!")
else:
    return client.get_result()
```

The frame_id of the goal represents the origin point. What this means is when setting a goal, the values will be relative to this location. This is important if you want to move to a set location like the door button, which means the frame_id must be set to the map origin point.

Most of the problems in the development come from outside factors such as faulty odometry code. The only known workaround to this is to get an improved version of the Gazebo simulator. You could also try fixing the problems manually, but that requires a great knowledge of the ROS navigation stack.

Developing the navigation script requires thorough use of RViz for debugging. Often the code is functionally sound but the outcome is unexpected. Outside factors play a huge role in determining whether or not the goal was successfully executed, and as such it is important to test the code and see the result in RViz and in Gazebo. The results of the two may determine what went wrong. Often, the output of Gazebo and RViz is contradictory, and this is usually because of bugs in the simulator, or RViz is ignoring certain factors that only exist within the simulator.

Push Button

The push button portion of the project required the development of two separate components that interact with each other, namely the beerPusher.py script and the button model implementation in gazebo. The beerPusher.py script is a python script that causes rosie to perform a motion approximating the movements a human would perform to push a button to open a door. The beerPusher.py script so named because of the nature of it's simulated target requires the inclusion of the ROS and baxter libraries (rospy and baxter_interface respectively). It also requires the inclusion of the time library, however this is not notable in the same way the other libraries are, with the exception of the aforementioned rospy and baxter_interface libraries the beerPusher.py script uses nothing more than python and it's standard libraries.

We will give a general outline of how the beerPusher.py script works below:

The script starts off in the main function as one would expect.
Firstly we instantiate the pusher variable with an instance of the Pusher class, we will get to what this does later (although I'm sure the reader probably already knows.)
Then we register the pusher.cleanShutdown() member function to be called upon program exit using the on_shutdown() member function. As with the Pusher class we will get to what this does at a later point, this will be a common motif in our explanations.
After this we call time.sleep() before finally calling the only other thing of real note in main() the pusher.moveArmsToSafetyPosition() member function.
We call the sleep() member function only to give a bit of aesthetic delay and break up the movements a bit.
Essentially the main functions main actions are to call the pusher.push() and then moveARmsToSafetyPosition() member functions. This causes baxter to perform a button pushing action with his left arm and then move his arms back to a position that is deemed relatively safe.

```
def main():
    pusher = Pusher()
    rospy.on_shutdown(pusher.cleanShutdown)
    pusher.push()
    time.sleep(1)
    pusher.moveArmsToSafetyPosition()

    rospy.loginfo("Finished.")

if __name__ == "__main__":
    rospy.loginfo("Initializing node... ")
    rospy.init_node("beerPusher")
    main()
```

The Pusher class is where all the action happens. It contains all the important member functions that govern Rosie the Baxter Bot's movements as she endeavors on her button pushing quest. Most importantly it contains pusher.push().
The push() member function starts off by moving Rosie's arms to a safe position. This is so that the arms are in a known and safe position to start the button pushing motions from. This call is followed by a number of calls to Pusher member functions that when combined in the particular combination seen below comprise the essence of button pushing.

```
def push(self):
    rospy.loginfo("We are in Puther.push")
    rate = rospy.Rate(self.rate)
    self.moveArmsToSafetyPosition()
    """ Execute the all important pushing sequence """
    self.__retractLeftArm(rate, "forward")
    self.leftGripper.close() # For better button pressing abillities!
    time.sleep(1)
    self.__partiallyExtendLeftArm(rate, "forward")
    time.sleep(1)
    self.__pushButtonWithLeftGripper(rate, "forward")
```

We will now look at the __pushButtonWithLeftGripper() member function to see how these functions that cause Rosies arms to move are implemented in a general sense. The function starts with a couple of variable definitions, their names are written in screaming snake case to indicate that they are intended to be constants. However we cannot actually make them constants. This is in our opinion one of python's many failings.
The variables are DIRECTION_0 and ARM. These variables are used to help make the program more extensible by making it more general.

After these definitions there is an if statement that tests if our direction is DIRECTION_0 (if the program were extended we would recommend changing this. Perhaps to a for loop that iterates over an array that contains potential directions, that is if there are more than two directions.) If the test pases we know that we are moving our arm in DIRECTION_0 and we create a variable jointPositions. JointPositions contains the angles we want to move each joint in ARM to. After JointPositions is defined we call self.__waitForLimbToMoveToPosition(), this is where all the fun happens. We pass a number of arguments to this member function. Notably we pass ARM and jointPositions, these variables define which arm will move and how.

```
# Moves arm and gripper in appropriate fashion for button pression (or beer bottle spilling.)
def __pushButtonWithLeftGripper(self, rate, direction):
    # Directions we can move in
    DIRECTION_0 = "forward"
    ARM = "left"            # Which Acorn Risc Machine are we using?
    if(direction == DIRECTION_0):
        rospy.loginfo("Attempting to push button using " + ARM + " arm and gripper in " + DIRECTION_0 + " position")
        jointPositions = {self.LEFT_ARM_JOINTS[0]: 0.0, self.LEFT_ARM_JOINTS[1]: -0.525, self.LEFT_ARM_JOINTS[2]: 0.0,
                          self.LEFT_ARM_JOINTS[3]: 1.0, self.LEFT_ARM_JOINTS[4]: 0.0, self.LEFT_ARM_JOINTS[5]: -0.487,
                          self.LEFT_ARM_JOINTS[6]: -0.8}
        self.__waitForLimbToMoveToPosition(rate, ARM, jointPositions, self.LIMB_MOVEMENT_TIMEOUT_TIME, " arm and gripper in " + DIRECTION_0 + " positi
    else:
        rospy.logerr("Error (in __pushButtonWithLeftGripper()): pos = ", direction, ", but the only option/s currently implemented are ", DIRECTION_0)
        exit(self.ERROR_NO_SUCH_DIRECTION)
```

3

__waitForLimbToMoveToPosition() calls set_joint_positions() on an object returned by baxter_interface.limb.Limb("left") or baxter_interface.limb.Limb("right") with jointPositions as its argument. Set_joint_positions() is part of the baxter_interface library. It causes Rosie to try to move the joints of the limb associated with the member functions object to angles equal to those specified in the argument to the function. If the limbs movement is obstructed in any way this function will not exit. It will instead continuously try to move the limb into position. It is for this reason that we have added some code at the end of the __waitForLimbToMoveToPosition() function that essentially calls the member function __getNumberOfArmAndGripperJointsInPosition(). __getNumberOfArmAndGripperJointsInPosition() does what it says on the tin, it returns the number of joints that are in jointPositions. If after a timeout all of the joints are not in position we exit with an error message. __getNumberOfArmAndGripperJointsInPosition() check that joint's are within Pusher.JOINT_PLAY of the desired angles. This is because in the real world (as well as the simulation) Rosie may never manage to move her joints exactly into position, at least not all of them at once.

The button model component of the push button portion of the project took a lot more time than expected as there were many problems encountered when trying to load

[3] *Note that the last argument to __waitForLimbToMoveToPosition() is partially cut off, this had to be done to make the image more readable.*

and edit the models. The button model consists of two main components. The first component is the button itself. With regards to this it must be noted that the editing facilities in Gazebo appear to be rather limited and editing is not made any easier by using gazebo over a remote connection. Instead of trying to make an exact replica of the VXLab green door button we decided that a batter course of action would be to get at the essence of the problem and so we came up with the idea of using a beer can model from the Gazebo model repositories that rosie could push over. The second component was then of course something for the can to rest on while waiting to be obliterated by Rosie. We created a pedestal for the can to sit on.
These models are located in the gazeboModels directory. The files that define the models contain XML. Each model is defined by a model.config and a model.sdf file. The model.config file contains metadata and the model.sdf file contains the specification of the model proper. The pedestal file was generated by Gazebo. Gazebo is directed to load the models upon start up by statements in the vxlab.world XML file. The location of each model to be loaded is specified by a URI and it's position by a set of coordinates.

```
<!-- Load something for our beer button to sit on. -->
<model name="beerPedestal">
  <include>
    <static>true</static>
    <uri>model://beerPedestal</uri>
    <pose> -0.590677 6.026460 0.739853 0 0 0 </pose>
  </include>
</model>
<!-- Load our beer can! -->
<model name="beer">
  <include>
    <static>true</static>
    <uri>model://beer</uri>
    <pose> -0.591135 5.932420 1.030944 0 0 0 </pose>
  </include>
</model>
```

[4]

The pedestal is loaded with its static attributes set to true. This means that it is essentially immutable and cannot be moved during the simulation, that is at least not without direct human intervention. The beer can model is loaded with its static attribute set to false. This means that it is movable during the simulation. However we encountered problems with this so the beer model must be moved to the correct location on top of the pedestal manually using the Gazebo interface after Gazebo has finished loading. Once the beer can is correctly placed on top of the pedestal it can be hit by Rosie.

[4] *Section of the .world file that loads button related models.*

Wave At User

The functionality to wave at the user is implemented in the python script waveLikeMade.py. There is not much here that needs explanation that has not already been explained as this script shares a lot of common code with the beerPusher.py script (here there is room for improvement as these functionalities could be raised out into another script in the future to improve modularity and reduce code repetition.) The script has a class that is analogous to Pusher, namely Waver. Waver has member functions for waving both arms. We are only interested in waving the right hand as it is Rosie's favorite hand so the call to wave Rosie's other hand is commented out. It is notable that the joint positions defined in Waver.__waveRightArm() and Waver.__waveLeftArm() are not exact mirrors of each other as we had trouble getting them to look exactly the same when performing the waving actions.

Initial Approach

The Initial approach was to get accustomed to the new environment we've been given access to. This also required some of the documentation Ian has sent to us and referred to. The process was lengthy as there wasn't any well documented or any documentation about the libraries for ROS available for Baxter that we could find. The feature of a chatbot was more easily implemented but required more tuning as running it in a docker instance was more challenging to install the libraries. The team ran some of the sample scripts with Rosie and watched tutorials online of the simulator to get a better understanding of the tools we were working on. Throughout the development of the project we have encountered several issues which we have shown to our stakeholders. This includes:

- Navigation system not working correctly
- Unpredictable occurrences in the simulator
- Missing components such as cameras

Iterative Sprint Development

Sprint 0

- Setting up the Development Environment:

The first few days we were tasked to get a method to implement our workflow for the simulator, This raised a number of questions and tasks that needed to be handled. Such as:

1. How to set up the project to use github
2. How to transfer files from the server to our own local script

3. Become accustomed to the new simulator being implemented since the previous version was missing the mobility base.
4. Reading up on documentation on general ROS knowledge

During these early stages of being introduced to the project and the tools that we'll be using. A large amount of time was spent on exploring the capabilities of the simulator and if there was a method to create customised scripts that Rosie could use inside the simulator. However for the development of a simple chatbot it was possible to showcase to the client as a form of progress.

Sprint 1

This sprint focused more on the team trying out the functions they discovered in the last sprint. This was done remotely as access to the lab was no longer possible. At this stage the team were tasked with separate components of the project to focus on. This would allow us to work on a particular feature we proposed to our client and supervisor which we aim to deliver to them. However at this stage of the project the navigation aspect of Rosie was not yet functional so the most optimal approach was to read documentation until the environment was ready for the team to start using it. The team also worked on the movement of the arms in the simulator where we could successfully move the prop to the desired location. On the chatbot development side, the team was able to implement a sample audio for the chatbot. This was to lead into the design of the chatbot accepting audio through a file.

Sprint 2

In this sprint, we applied what we learned in sprint 1. The time we spent on reading the documentation is put to use here. Functional scripts are made, and Rosie was able to perform actions, although there were many bugs. Early in the sprint, the team has worked together with Ian Peake to get a new version of the simulator up and running on the server. This new version contains working functionalities for Rosie's navigation. At this stage, we also learned how to use RViz to debug any navigational problems, although in the first half of the sprint, there were numerous problems with RViz that were eventually fixed. Basic functionality such as Rosie moving to the door was doable, but not yet perfect. Rosie would be several meters off course. This navigational problem would later go unsolved, but in later sprints, we developed a workaround to this problem. There was also a problem with loading models into the simulator, which delayed the development of the button pushing mechanism. Most of the time spent in this sprint was ensuring that our requirements could be met when working in the current environment, which was unfortunately not perfect. Most of the work in this sprint needed to be done because of the Pandemic causing us to change our work environment.

Sprint 3

In this sprint, most of the work done is integration and testing. The testing section can be explored more under Technical Overview. To name a the components we've tested are:

- NLTK Chatbot
- Navigation
- Push Button and Wave

The problem with the button pushing mechanism and the model loading of the button, which is currently a beer can as a placeholder, has been fixed for the time being, whether through workarounds or through functional means. We have also decided to add some basic functionalities to Rosie, like move and spin, as well as expanded upon the chatbot, improving Rosie's conversation ability. The basic navigational functions are also beneficial in working around the issues in the simulator. Any deviations from Rosie's destination point can be made up using these basic movements. However, we soon find that Rosie is not able to do rotations properly, due to problems coming from the simulator itself. During integration, there was a problem with integrating the button pushing mechanism. This problem was not seen initially because the code was tested with an earlier version of the simulator that did not contain the navigation function. We did not believe this would cause a problem until the time came when we integrated our parts. By the end of the sprint however, we have managed to integrate our parts.

Sprint 4

In this sprint, we decided to make a demonstration video that will showcase all the work we have done so far. This was suggested by our supervisor and client. We have also used this sprint to refactor and reorganise our code, as well as our repository. Work on this report also started from the beginning of this sprint. In this sprint, the simulator's bugs are often preventing us from creating a proper demonstration video. We thought of many different ways that could workaround many of the simulator's problems since sprint 3, but recording the demonstration video is a different problem altogether. Not much technical work was done here, as we believe last minute changes may cause unexpected damage, hence we decided to focus more on the report and demonstration videos. Our supervisor and client gave us feedback on our videos each week, and we try to incorporate it into our demonstration videos.

Conclusion

The new opportunities this project include students and clients would be able to interact and demonstrate projects built in the simulator. Our project is an approach the team has opted towards since the pandemic has forced us to work in this kind of environment. It is a learning phase for all of us as this was not expected to happen and with this project we will be able to better prepare for next time. Ian has highlighted that this project is what would greatly benefit future students with their projects when a similar scenario occurs. This also enables earlier interaction with Rosie and the vxlab while going through the mandatory induction. The outlook for a different approach is to focus more on delivering navigation as this aspect was the one with the most issues. But the team did an overall excellent job on each of their respective components as the simulation was limited in capability. With the new simulator environment currently under development by Ian. We encourage future teams to use our developed scripts to get inspiration from.

Demonstration

An example of our project can be found on youtube through this link below:
https://www.youtube.com/watch?v=aYiqchH3MK4&t=3s