Robotics: Final Project
A PERSONAL ASSISTANT

Thomas Murphy[1], Alexis Corney[1], Rehaan Cheema[1], Esha Dasgupta[1], and Joseph Maliszewski[1]
[1] Computer Science, University of Birmingham

Interaction with robotic systems has become increasingly common in recent years with the advancement and autonomous technology, allowing more tasks and jobs that can be carried out by such systems. These systems require multiple components working in tandem to carry out those tasks. This project looks at designing and implementing an autonomous system that can carry out tasks similar to that of a Personal Assistant (PA), as well as focusing on assessing the effectiveness of such a system and by extension, the individual components by setting tasks for it to carry out. The results showed that whilst successful in performing its principal function, the robustness against certain edge cases required improvement as well as possibly reducing system complexity.

I. INTRODUCTION

RAPID developments in Artificial Intelligence have given rise to intelligent voice-control systems such as Alexa and Siri, and seen a surge in the popularity of commercial robots. This project aims to develop a fully autonomous personal assistant robot to aid human users with their day-to-day activities.

Leonard, as the robot is called, works by tasks, he has the ability to receive and deliver objects and messages. A task is created from user input using voice commands, users may request that Leonard deliver a message to another person or take an object to a person or location. Once a task is generated, it is scheduled for execution. The solution created consists of the following components:

- **Machine Vision**, whereby the robot can detect and classify objects placed in its field of view, as well as register faces, both known and unknown.
- **Voice Recognition**, using text-to-speech and speech-to-text to communicate with the user. Leonard can interpret voice commands and convert these into tasks, then synthesize speech to relay information back to the user.
- **Scheduling** algorithms determine the optimum order for task execution based upon proximity to task locations, task priority and future tasks. The scheduler updates continuously.
- **AMCL Localisation** enables the robot to estimate its location within its environment.
- **Route Planning** via RRT to generate the most efficient path between the start and end destinations of each task.

II. RELATED WORK

Examples of related works that involve the use of robotics systems to carry out service-based tasks include: a receptionist [6] and a waiter/server [7], implemented by a previous Birmingham robotics team. These uses of robotic systems for aiding human activity demonstrate how a personal assistant system would also be effective and beneficial. However, the difference between these previous implementations and Leonard is that those systems are only carrying out one specific task, whereas, the personal assistant robot will perform a series of tasks. The system will be able to execute two types of task; message delivery and item delivery.

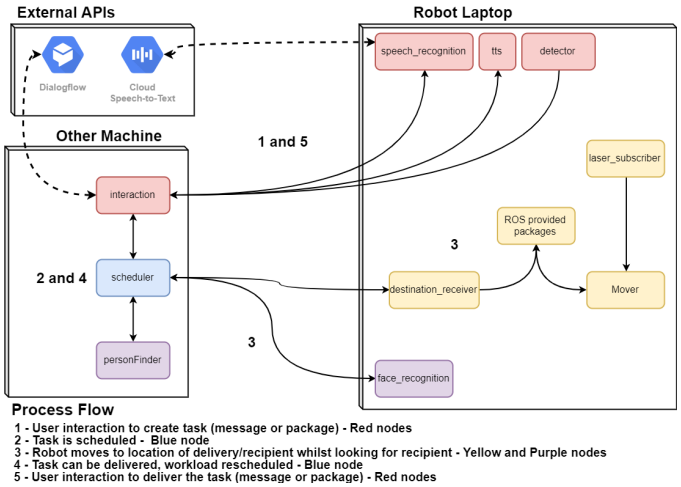

Fig. 1. Process flow over system architecture

III. FRAMEWORK

A. Overview

For the robot to achieve its task of being a Personal Assistant, it requires a series of modules/nodes to handle key functionalities that can influence and be influenced by other modules. These must enable the robot to be given a task, plan its execution based on whatever outstanding task(s) it may have, and execute said tasks. These key functionalities are:

- User Interaction
- Scheduling
- Object Detection
- Face Recognition
- Localisation
- Route Planning/Execution

It was discovered that there is a considerable amount of processing power needed for the system to carry out its task. Therefore, a distributed approach was used which involved running certain nodes on a separate machine, connected to the same network as the laptop that controls the robot. An overview of the process flow for this distributed architecture is shown in Figure 1.

A detailed diagram containing the full system architecture with full interactivity between the nodes, i.e. the topics that each node publishes to and subscribes from, can be found in Figure 20 in Appendix B.

B. User Interaction

1) Introduction

In order for the user to be able to give Leonard a task, either to deliver a message or package, and for Leonard to be able to deliver that task to the recipient, there must be some form of interaction between the user and the system. To do this, a speech-based interaction model was chosen. This is more suitable for the systems requirements than a text based interaction system as the user should be able to speak to the robot as if speaking to a human personal assistant. The microphones within the robot's front-facing camera and the speakers from the laptop were used to facilitate the interaction. The user interaction module influences the scheduling and movement control modules through the topics it publishes to.

2) Algorithms

The interaction between the user and the system can be split into three components.

- Converting the audio input (the user's speech) into text
- Running the text through the conversation logic module to compute a task
- Passing text back so that it can be spoken aloud to the user

Further detail of the program flow of these components is outlined below.

Speech-to-Text

The speech-to-text node runs on the robot laptop, in order to make use of the microphone on the robot's camera.The user can interact with the system by walking up and speaking to it. This functionality was implemented by using the *SpeechRecognition* python package which acts as a wrapper to send audio clips to a recognition API.

The node works by recording a 5 second audio snapshot when instructed to do so (this instruction was published to a topic from the conversation logic module). It then produces the most likely transcription of that audio. The transcription is carried out by the Google Speech Recognition API. Due to time constraints, this was preferable to creating an offline speech to text model from scratch. Before recording the snapshot, the node first listens to the environment to gauge the ambient noise level and set a threshold so that, during the snapshot recording, any noise falling below this ambient threshold is considered noise and discarded. This achieves a more accurate transcription.

Conversation logic

This node is responsible for the flow of control, both for user interaction and other components of the system. Its main functionality is to create the tasks specified by the user and publish them to the scheduler. It also makes sure that any tasks that the scheduler publishes are delivered. In order to understand the user's intention and to be able to extract relevant entities from the user's speech, Google Cloud Dialogflow was used. This is a cloud based service that builds and hosts a learning model for the conversation.

Created within Dialogflow were the set of intentions that a user may have. These included, creating a message or package delivery task as well as other fallback intentions, such as; ending the dialogue or a default response, should the model not understand the sentence. To train the model, a set of example utterances for each intention were given. Also specified, were the kinds of entities that may need to be extracted. These entities form the key task information that the scheduler needs to effectively organise tasks. They include; the recipient of the task, the sender, the location for a package delivery, the message for a message task and the urgency modifier, which can influence the scheduler to prioritise a task should it be specified as urgent.

The conversation logic node can exist in any of the following states:

- **Listening for Command** - This the node's default state, where it waits until it gets given the correct command. When in this state, it will instruct the speech-to-text node to record the audio snapshot and then waits for the text transcription. If the returned transcribed audio is the robot's listen command, which is "Leonard", the node will transition into the *Begin Conversation* state.
- **Begin conversation** - This state is entered into after the user has spoken the command 'Leonard', indicating that the user is going to give the robot a task. The node will send one of a set of various greetings messages to the Text-to-Speech node. It will then wait for the transcribed audio of the users request. This request is passed to the learning model via the Dialogflow API which returns the most likely intention and any entities it was able to extract from the request. The node will then transition to either the *Create Message Task* or *Create Package Task* state based on the intention. If the intention cannot be recognised, or it isn't a recognised intention, the node will pass the model response to Text-to-Speech before returning back to the *Listen for Command* state.
- **Create Message Task** - The node transitions to this state if it identifies that the user's intention is to create a message task for the robot to deliver to someone. The aim of this state is to collect all the required information from the user about the task, so that the scheduler receives all the necessary information. This is done by probing the user for the missing information with the use of contexts. Dialogflow uses a context system so that, when it receives a request, it looks at a context key to see if this request is related to the context of whatever dialogue is happening. In this case, a message task creation dialogue. If the request is related to the context, it will extract any of the required entities that it can find before returning an updated list of entities to the node. The node can then consult the entities list to see what information is missing so to ask the user to provide that missing information. The key entities required for a message task are: the sender and the recipient of the message, the message itself, and whether or not the task is urgent.

Once all the required task information has been collected from the user, the system will relay the request back to the user so that they can confirm that the task is correct. If the user verified the information to be correct, the task is passed onto the scheduler and the conversation logic returns to the *Begin Conversation* state. However, instead of saying a greeting, it will ask the user if there is anything else to send. If, however, the user says the information is incorrect, the node will return to the start of the task creation and ask for the all the necessary information again.

- **Create Package Task** - This state is entered into if the user's intention is recognised as a package task in the *Begin Conversation* state. This states works in the same way as the *Create Message Task* state by collecting the necessary information through contexts and. It, either passes the task to scheduler, or restarts the task creation if the information is incorrect. The main difference between the two states is that, in this state, the user needs to give the robot the item to send. To do this, the node subscribes to the *'/objects_detected'* topic provided by object detection.
  Once the user confirms that the information it has is correct, the node logs how many objects it currently has detected on the platform before asking the user to place their intended object onto the platform. The node will only send the task to the scheduler once it identifies a positive change in the number of objects *(num_objects_after_placement > num_objects_before_placement)*. After the node has published the task for the scheduler to receive, it returns to the *Begin Conversation* state, in the same way as the *Create Message Task* state.

Text-to-Speech

The Text-to-Speech node, like its counterpart, needs to run on the robot laptop so that the system's 'speech' appears to come from the robot, to give the impression of a dialogue between Leonard and the user. The process flow works by subscribing to the *'/speakMsg'* topic, published by the conversation logic node, then sending a notification via *'/doneSpeaking'* to inform the logic node that the system has finished speaking so that the conversation can continue.

3) Experimental Results

In order to gauge the accuracy of the speech recognition node, a series of experiments were carried out to highlight the effect of ambient noise on the transcription accuracy. To do this, a set of example utterances were spoken into the microphone multiple times and the number of times it was able to exactly transcribe the utterance, a transcription with minor errors (spelling mistakes, a word that has similar characters to the target utterance etc.) or a failed transcription, which is either a failure to transcribe the audio or an incorrect transcription, were recorded. The set of test utterances also varied in length and complexity to further investigate the effect of ambient noise. The short utterance was the selected command word "Leonard", the longer, simple utterance was "This is a test message" and the longer more complex utterance was "Can you deliver this parcel to LG04".

Each test utterance was repeated 10 times at 3 different ambient noise levels: no ambient noise (silent room), mild ambient noise (low level conversation) and high ambient noise (busy area, multiple conversations). The results of the speech recognition node's accuracy can be found in Table I

TABLE I
SPEECH RECOGNITION ACCURACY AGAINST AMBIENT NOISE

Utterance		no ambient noise	mild ambient noise	high ambient noise
Leonard	Correct	90%	80%	70%
	Minor error	0%	10%	10%
	Incorrect	10%	10%	20%
This is a test message	Correct	100%	90%	70%
	Minor error	0%	10%	20%
	Incorrect	0%	0%	10%
Can you deliver this parcel to LG04	Correct	80%	70%	70%
	Minor error	20%	30%	30%
	Incorrect	0%	0%	0%

Table I shows that, whilst overall accuracy still remains high over different levels of ambient noise, the higher the ambient noise, the more likely there is to be a slight misinterpretation of the audio for longer utterances. In the case of short, single word utterances, the likelihood for the that audio to be isolated and transcribed amongst the ambient noise slightly decreases, leading to possible inaccuracies or an inability to detect that utterance.

4) Challenges

The principal challenge for the user interaction component of the system was dealing with the variable ambient noise of the environment, as it was constantly changing. Even after the speech-to-text node adjusted to ambient noise, the ambient noise energy threshold could change and no longer match the currently set threshold, thereby affecting the node's ability to transcribe the audio, as previously shown in the experimentation. However, since the ambient noise had less of an effect over longer utterances, a future improvement would be to change the command phrase to be longer and more unique so that when that command phrase is recognised, it is more certain that the user wants to initiate a dialogue with the robot.

Another challenge this component faced was with hardware as the different audio input sources react differently to ambient noise than others due to their sensitivity. For instance, the inbuilt microphone on the laptop was very sensitive to ambient noise and the recording quality was poor which had a detrimental effects on the transcription ability of the node. The microphone that we chose for the final implementation was the microphone array in the front-facing camera as the position of the microphone suited the position in which the user would stand when interacting with the robot. However, its sensitivity to ambient noise was still high which, as the experiments show, made it susceptible to high levels of ambient noise.

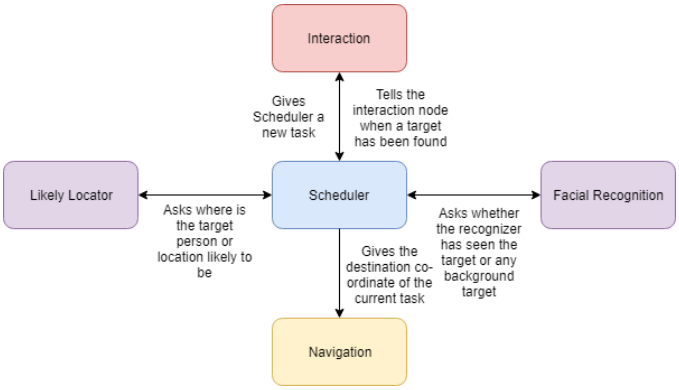

Fig. 2. Information flow through the Scheduler

C. Scheduling

1) Introduction

When Leonard accepts tasks from users, it must be able to decide the order in which to execute the tasks. The scheduler is the central component of the system which takes in the information from the interaction module to translate it to instructions for the navigation nodes by communicating with the facial recognition and locator nodes to gather information. Figure 2 outlines the flow between the five processes.

The ordering of the tasks is decided by a ranking algorithm which assigns weights to each task and updates them as new information is acquired.

2) Algorithms

Interaction accepting a New Task: The scheduler accepts new tasks from *Interaction* by subscribing to the topic \new_task. It then creates a Task Object and subscribes to the topic \amcl _pose to get Leonard's current position.

Scheduler Adjusting to the New Task: As the tasks are chosen to be executed in a weighted order, the weight of the new task must be calculated, and all of the weights for the remaining tasks need to be adjusted because the distance from Leonard to their destinations will have changed since they were first calculated. The scheduler can choose to switch current tasks whenever it gets new information (spotting a target, accepting a new task, etc), so it is important to recalculate all of the weights with the correct information to make sure the decision process chooses the correct task to deliver.

Calculating the Weights: The formula for calculating the weights for each task is simple. There are currently two components; the distance from Leonard to the destination and whether the task is urgent or not.

$$Weight = \alpha/Distance + Urgent \tag{1}$$

where

$$Distance = EuclideanDist(amcl\_pose, DestinationCoordinate) \tag{2}$$

$$Urgent = \begin{cases} \beta, & \text{if urgent flag set in task} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

By varying α and β it is possible to override the scheduler, deciding to execute urgent tasks first if Leonard is close to another task's destination.

The $DestinationCoordinate$ is determined by the Locator, which is described in D. The weights and the tasks are stored in a dictionary.

Setting Background Tasks: To make Leonard more efficient and intuitive when carrying out tasks, it splits tasks into 'Current' and 'Background' jobs. While it is carrying out a current task, if the facial recognition sees a target for one of the background tasks, it will prompt a re weighting with an adjusted distance for the spotted person to see if the scheduler wishes to suspend the task currently being executed, promote the background task to the current task and deliver that task instead. To accommodate for this, whenever the scheduler accepts a new task, it always sets it as a background task before choosing the current task.

Choosing the Current Task: If there is at least one task, the scheduler chooses the task with the highest weight as the next task to carry out. It then publishes the current target to the facial recogniser and the task's destination pose to the navigation node.

When the Facial Recogniser finds a target: When the Facial Recogniser publishes that it has found a target, the Scheduler takes the estimated pose from amcl and readjusts all the tasks with their new distances to their destinations, as well as increasing the weight of the task whose target has been spotted. Once it recalculates the task order (it may or may not change the current task), it checks whether the person spotted is the current target. If they are, it returns, to *Interaction*, all the tasks intended for that target. If the weights are such that the Scheduler decided to continue with a different task, then it publishes the new target, and facial recognition is rebooted. A new target is not published if the scheduler decides to deliver the person's task.

When a Task is delivered: After Interaction finishes its deliver conversation, it publishes which person had their task/tasks completed. The Scheduler removes all of their tasks from the list of outstanding tasks, reschedules, then publishes the new target if there are any tasks left to carry out.

3) Experimental Results

The Scheduler was tested along side the Facial Recognition node, as the latter provides the former with crucial information. The Scheduler was given 3 tasks, and were shown 3 people in order and its responses were recorded. Note: The Current = 'k' refers to the Current Target it was searching for before the adjustment upon seeing the person. The weights, α and β, have been chosen so that the scheduler chooses to deliver a task when it sees any target. Further levels of urgency priority can override this in the future.

Figures 3 and 4 show the scheduler's responses. Figure 3 deals with known targets and varying task urgency. Figure 4 tests the scheduler when targets are unknown, both current and background.

Together, they have an accuracy of 94.4% when delivering messages. As Facial Recognition's accuracy is very high, it allows for the Scheduler to be confident that it has found the correct person to deliver to. The ordering of tasks is

			Tasks Given to it in Order		
		Esha (M nU) (D=23.93)	Esha (M nU) (D=67.73)	Esha (M nU) (D=67.73)	Task 1
		Tom (M nU)(D=55.18)	Tom (M nU)(D=22.48)	Tom (M nU)(D=22.48)	Task 2
		Joe (M nU)(D=132.06)	Joe (M nU)(D=132.06)	Joe (M U)(D=132.06)	Task 3
	Esha	M for Esha (Current=Esha)	M for Esha (Current=Tom)	M for Esha (Current=Joe)	
	Tom	M for Tom (Current =Tom)	M for Tom (Current =Tom)	M for Tom (Current =Joe)	
	Joe	M for Joe (Current=Joe)	M for Joe (Current=Joe)	M for Joe (Current = Joe)	
People the	Tom	M for Tom (Current=Esha)	M for Tom (Current=Tom)	M for Joe (Current = Joe)	
Facial Recogniser	Esha	M for Esha (Current=Esha)	M for Esha (Current=Esha)	M for Esha (Current = Tom)	
saw in order	Joe	M for Joe (Current=Joe)	M for Joe (Current=Joe)	nothing (Current = Tom)	
	Tom	M for Tom (Current = Esha)	M for Tom (Current = Tom)	M for Tom (Current = Joe)	
	Joe	M for Joe (Current = Esha)	M for Joe (Current = Esha)	M for Joe (Current = Joe)	
	Esha	M for Esha (Current =Esha)	M for Esha (Current =Esha)	M for Esha (Current =Esha)	
	Esha	M for Tom (Current =Esha)	M for Esha (Current =Tom)	M for Esha (Current=Joe)	
	Rehaan	nothing (Current =Esha)	nothing (Current =Tom)	nothing (Current =Joe)	
	Joe	M for Joe (Current=Esha)	M for Joe (Current=Tom)	M for Joe (Current = Joe)	
	FR recognises	Esha	Esha	Esha	
		Tom	Tom	Tom	
		Alexis	Alexis	Alexis	
		Joe	Joe	Joe	

Fig. 3. Table showing the results when the scheduler is given 3 tasks and sees 3 people in turn. Green cells indicate correct deliveries, orange cells are mistakes by the facial recogniser.

			Tasks Given to it in Order		
		Rehaan (M U) (D=5.66)	Jesse (M nU) (D=68.28)	Esha (M nU) (D=13.25)	Task 1
		Tom (M nU)(D=129.15)	Jesse (M U)(D=68.28)	Ioana (M nU)(D=47.82)	Task 2
		Joe (M U)(D=132.06)	Alexis (M nU)(D=164.71)	Joe (M U)(D=132.06)	Task 3
	Esha	nothing (Current =Rehaan)	nothing (Current=Jesse)	M for Esha (Current = Joe)	
	Jesse	who are you (Current =Rehaan)	M,M for Jesse (Current=Jesse)	nothing (Current=Joe)	
	Rehaan	who are you, M for Rehaan (Current=Rehaan)	nothing (Current=Alexis)	nothing (Current = Joe)	
People the	Rehaan	M for Rehaan (Current=Rehaan)	nothing (Current=Jesse)	nothing (Current = Joe)	
Facial Recogniser	Joe	M for Joe (Current=Joe)	M,M for Jesse (Current=Jesse)	M for Joe (Current = Joe)	
saw in order	Tom	M for Tom (Current=Tom)	nothing (Current=Alexis)	nothing (Current = Esha)	
	Tom	M for Tom (Current = Rehaan)	nothing (Current=Jesse)	nothing (Current = Joe)	
	Rehaan	M for Rehaan (Current = Rehaan)	nothing (Current=Jesse)	nothing (Current = Joe)	
	Alexis	nothing (Current =Joe)	M for Alexis (Current=Jesse)	nothing (Current = Joe)	
	Ioana	nothing (Current = Rehaan)	nothing (Current=Jesse)	nothing (Current = Joe)	
	Esha	nothing (Current =Rehaan)	nothing (Current=Jesse)	M for Esha (Current = Joe)	
	Rehaan	M for Rehaan (Current = Rehaan)	nothing (Current=Jesse)	nothing (Current = Joe)	
	FR recognises	Esha	Esha	Esha	
		Tom	Tom	Tom	
		Alexis	Alexis	Alexis	
		Joe	Joe	Joe	
			Jesse	Jesse	
			Rehaan	Rehaan	

Fig. 4. Table showing the results when the scheduler is given 3 tasks and sees 3 people in turn. Green cells indicate correct deliveries, orange cells are mistakes by the facial recogniser.

also intuitive, and the scheduler stopping an 'urgent' task to deliver a message to someone close by is understandable. This because the users are students and a truly urgent message, which requires immediate attention, is unlikely to be sent via Leonard.

The accuracy of delivering packages is dependent on the facial recognition as well as navigation to the drop off point, but as the scheduler's role does not change between the two types of tasks, it has only been tested for messages.

4) Challenges

The main challenge with the Scheduler was the communication between the nodes. When it was given current information, the scheduler had to be able to interrupt the running processes and restart while adjusting them. This was achieved with the callback functions when subscribing. The order of the scheduler getting the information was not deterministic, so it had to deal with the uncertainty that it was dealing with old data, but this was negated by increasing the frequency at which the topics it subscribed to were published to. Any old data would be overwritten quickly. There were sometimes strange occurrences when a publisher would publish to a topic but a *rostopic echo* would show that nothing had been published.

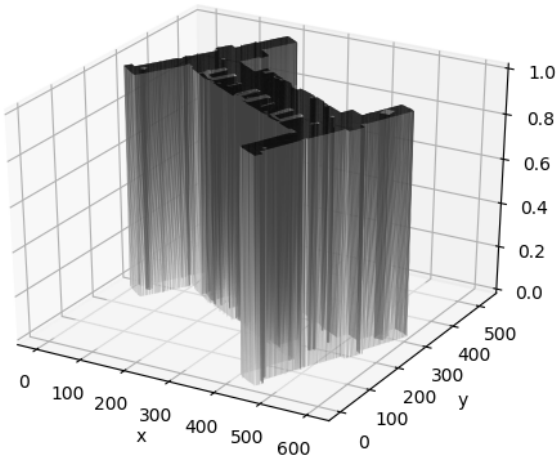

Fig. 5. Graph showing the base weight for locations when trying to find a person with no prior information

D. Locator

1) Introduction

When given a task, for the Scheduler to calculate its Distance weight, it needs to know where to find the target locations or the target people.

2) Algorithms

Destination coordinate for a Target Location: Leonard is allowed to move in the Lower Ground Floor of the University of Birmingham Computer Science Lab. As the locations are static, it is enough for the Locator to have a direct mapping between location names and their coordinates. If the task type is to deliver a package, the locator will look up the target location from the dictionary and return the coordinate.

Destination coordinate for a Target Person: This is a more difficult problem as people are not, by nature, static, and Leonard cannot expect to find his targets in the same location every time. The Locator keeps a distribution for each known person in its database so it can give a probabilistic answer when asked where the target is likely to be. Figure 5 shows the distribution when there is no prior information to where the person could be, and they have not been seen by the facial recogniser before.

When the locator has no prior information, it defaults to a list of common locations where a student is likely to be e.g. the booths, LG04 or the robotics lab.

When the facial recogniser spots a target (current or background), the locator logs the current position and updates the person's distribution. It plots a Gaussian at the coordinate where the person was seen for future reference, and, if they have been seen before, it scales the previous Gaussian plots by 0.9 to the power of how out of date the observation is. For example, if it sees Person A at $(x2, y2)$ at time $t2$, then it will scale the Gaussian for $(x1, y1)$ at time $t1$ by 0.9 and $(x0, y0)$ at time $t0$ by 0.81 to signify currency. However, any value outside of the lower ground floor is clamped to 0 so the locator does not return nonsensical values. The Gaussians are not combined into one because the target is dynamic; a

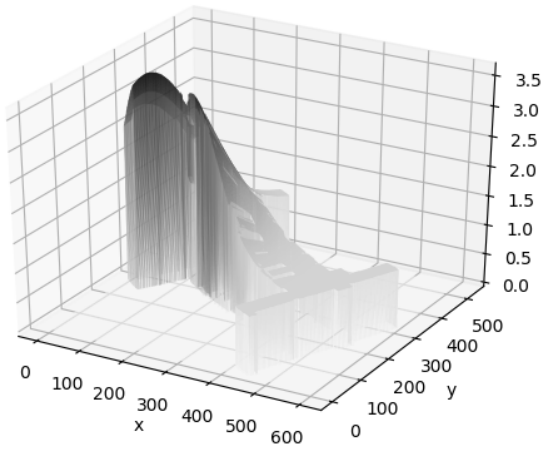

Fig. 6. Graph showing the distribution for a person when the locator has prior information

student's path and likely locations cannot be modelled well by a single Gaussian. An example of the updated distribution can be seen in Figure 6.

The locator returns the coordinate where the weight is the highest as the destination coordinate for the target.

3) Experimental Results

The Locator cannot be tested on its own as it requires face recognition and AMCL to provide it with information. However, to isolate the Locator's results, a few dummy situations were set up.

The Scheduler wants to know where 'Alexis' is with no prior information:

```
('Alexis', 'is most likely to be at [', 136.0, ',', 307.0, ']')
```

Fig. 7. Result for the scheduler asking the locator where Alexis is with no prior information

The locator does not have any information about Alexis so it defaults to a random popular location, as seen in Figure 7. $(136, 307)$ is the coordinate for the Main Robotics Lab.

The Scheduler wants to know where 'Alexis' is with prior information that she was at (450,300) then at (100,500):

```
('Alexis', 'is most likely to be at [', 99, ',', 500, ']')
```

Fig. 8. Result for the scheduler asking the locator where Alexis is with some prior information

The locator has chosen a coordinate near the most recent place Alexis was seen as the place with the highest likelihood to locate her, as seen in Figure 8. Her graph for the decision is visualized in Figure 9.

The Facial Recogniser has recognised Alexis at (198,405):

As seen in Figure 10, the locator updates its distribution correctly when the Facial Recogniser publishes that it has recognised Alexis. The current coordinate is obtained from AMCL, and if asked, the locator will take all of its information

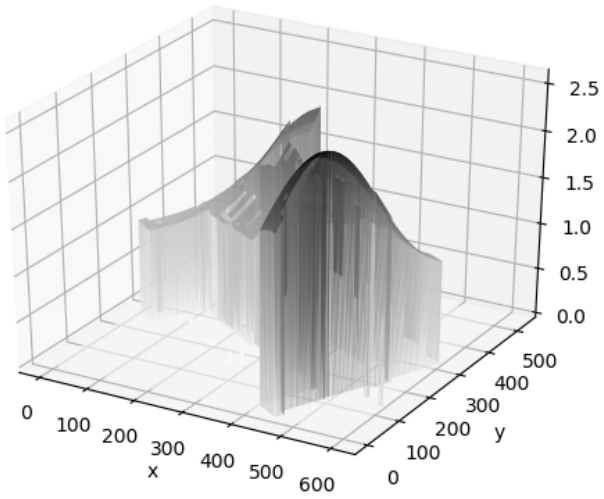

Fig. 9. Alexis' distribution graph visualized

```
('Saw', 'Alexis', 'at', [198, 405])
('Alexis', 'is most likely to be at [', 155, ',', 447, ']')
```

Fig. 10. The Locator is updating its distribution after the Facial Recognition recognizes Alexis

into account to say that Alexis is likely to be at $(155, 447$ next time it is asked. Figure 11 shows the updated distribution.

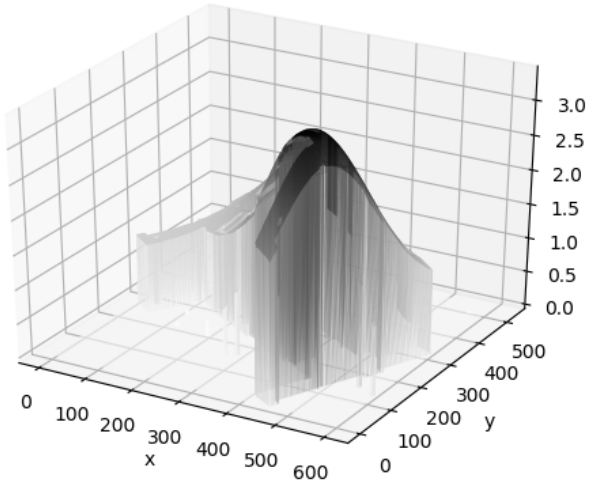

Fig. 11. Alexis' distribution graph visualized

4) Challenges

The main challenge for the Locator was aligning the coordinate system of the image file for the map with the coordinate system AMCL uses so the update would be correct. The image reader which maps the pixel to its colour maintains a different origin to RVIZ, so transforms had to be used to keep the distributions correct.

E. Object Recognition

1) Introduction

Leonard is able to deliver items to its users. The robot must be able to 'see' what objects it is carrying. A platform was constructed underneath a USB camera for items to be placed into.

Using OpenCV[**?**] and imageAI[2] an object detection system was designed, based on the "You Only Look Once (YOLO)" model. This runs continuously during the operation of the robot. It detects all objects in the tray by feeding each frame of the USB camera capture to a classifier. Predictions with sufficient confidence scores are collated into a string which is published to a topic called /objects_detected. This topic is subscribed to by Leonard's interaction module.

2) Algorithms

YOLO Model

Traditional classifiers can be used to detect objects by sliding a small window across the image to return a prediction of which sorts of objects are inside the current window. Potentially thousands of predictions for each image are then filtered out according to the certainty the classifier has for each prediction.

YOLO takes a different approach. It divides the image into a grid of 13x13 cells, each of which is responsible for predicting 5 bounding boxes to describe the rectangle that encloses an object. YOLO also outputs a confidence score detailing how certain it is that a predicted bounding box contains some object, combined with a prediction of the object's class. The range of classes depends on the dataset that the model is trained on.

The YOLO architecture is a Convolutional Neural Network consisting of a number of 3x3 convolutional layers and 2x2 max pooling layers. It uses leaky ReLu as the activation function. Each image is passed once through the network and a 13x13x125 tensor describing bounding boxes for the grid cells is returned. The predictions contain a confidence score. The parameters to the function to produce this score were learned during training by taking the probability of the object multiplied by the Intersection Over Union between the predicted bounding box and the ground truth of training data.

Intersection Over Union is an evaluation metric used to measure the accuracy of object detection on a dataset. It is, simply, the area of the overlap between a predicted bounding box and the true bounding box divided by the area of their union. IOU is an excellent metric for object detection because it is a non-binary classifier. It would be unrealistic to expect predicted bounding boxes to exactly match the ground truth.

Training a Model

ImageAI provides methods to train one's own model. By writing a script to save snapshots from the camera, a labelled dataset was produced by drawing bounding boxes around the objects in the images and labelling them with their object class. This training data was then used to extend a pre-trained YOLO model. Including images specific to the task refined the robot's object detection capabilities by extending the types of objects it is able to detect. It also increased the likelihood that the model would correctly classify items placed on its tray during delivery tasks, as the camera feed would be similar to images in the training data.

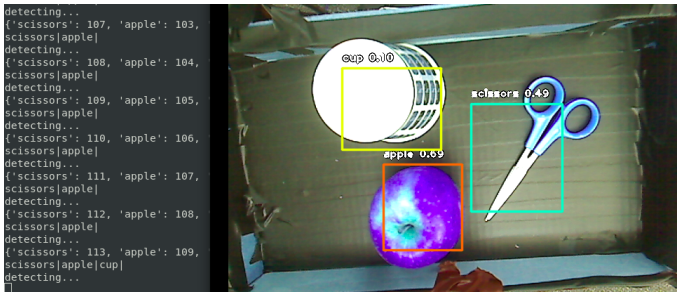

Fig. 12. Example View of Objected Detected

3) Experimental Results

To aid assessment of the accuracy of the classifier, the output of the detector, including bounding boxes, class predictions and confidence scores, was displayed after each frame of the video. After experimentation, it was found that the most satisfactory probability threshold was 70%, whereby, the model detected mostly correct object classes and ignored most incorrect predictions.

However, during experimentation, it was noticed that the detector often predicted the correct object, with much lower probabilities (around 8-15%). Therefore, a probabilistic model was constructed which considered the consistency of predictions. It uses a dictionary to map predicted objects to the number of times they are seen. This dictionary is updated after each frame, the count of objects that are seen again is incremented and the count of objects that are no longer seen is decremented. Then, to compute a final probability score, the probability of the object multiplied by the object's count, multiplied by an arbitrary weighting parameter is added to the the object detector's predicted probability. This computation weights the detected objects by how frequently they are seen. Therefore, after a number of iterations, consistently predicted objects will be included in the list that is published to the /objects_detected topic. Through trial and error experimentation, it was found that the best value for the weighting of the count was 0.2.

It was also beneficial to experiment with pre-processing of the images. By altering the brightness and contrast of the input image and applying mild thresholding, the images could be made sharper and more defined. This allowed the classifier to better detect the objects in the frame.

4) Challenges

The biggest challenge faced with the object detection system was that features of the ImageAI package appeared not to be supported on Linux OS. During construction and testing of the object detection system, a parameter called 'minimum_percentage_probability' was used to restrict the items that the model detects to those with confidence scores above a threshold. Unfortunately, when the system was deployed on

the robot laptop, this parameter did not work.

Consequently, as the detector processed each frame of the camera capture, it returned all potential predictions, regardless of their likelihood. This gave 13 x 13 x 5 = 845 bounding boxes and meant that the processing was extremely slow, and the output predictions unreliable. To combat this, it was necessary to manually filter the predictions based on their probability.

Originally, a ResNet model was tested. This is a Region based Convolutional Neural Network (R-CNN). It extracts 2000 regions from each image which are then fed into a CNN that produces a 4096-dimensional feature vector as output. However, due to the aforementioned issue with the probability threshold and the complexity of the network, the processing of the ResNet probabilities took around 50 seconds per image. This was unacceptable for Leonard's task because it requires the frame-by-frame processing to be as close to real-time as possible. The /objects_detected topic should provide a seamless update of the objects seen by the robot.

There was a trade-off between processing speed and prediction accuracy.

Eventually, a miniature version of the YOLO model, Tiny-YOLO, was used. Tiny-YOLO is based off the Darknet Reference Network and offers an accuracy of 61%. It uses 9 convolutional layers and 6 pooling layers, 3 times fewer layers than the standard YOLO model and does not perform batch normalisation on the data. This means that it runs faster, therefore, decreases the processing time for each camera frame.

F. Facial Recognition

1) Introduction

When the Scheduler is told to find a target person, it is the Facial Recogniser (FR) which is responsible for identifying them and returning confirmation of who it has spotted. It uses the face_recognition python package which has dependencies on OpenCV and the dlib C++ package.

The FR node does this by subscribing to two main topics: current target and background target. Its priority will be to find the current target, but if it spots any of the background targets, it will let the scheduler know to allow it to recalculate its decision to find the current target.

2) Algorithms

The node reads from the camera as a video and analyses it frame by frame to classify any faces using the argmin of the encodings that it has. If faces are spotted in a frame, it draws a bounding box around them. If the facial encoding is known and exists in the dictionary, the person's name will be displayed. An example can be seen in Figure 13

Recognition of Background Targets: The scheduler always schedules every new task as a background task before promoting a job to current. FR subscribes to the background target topic and adds the names of targets into the set *BackgroundList*. If the new background target is the current target, it skips this step, as it is redundant to search for the same target in the foreground and background.

Fig. 13. Example of Facial Recognition with one known person and one unknown

If any of the people FR recognises are in the *BackgroundList*, it publishes that it has found a target and lets the scheduler decide what it will do. If any of the background targets are people for which FR does not have existing facial encodings, it will not attempt to update its database by asking the unknown people what their names are. So FR can chance upon an unknown background target and not realise because its priority is to find the current target.

Recognition of the Current Target: When the Scheduler publishes the current target, FR sets its *Current Target* to the person. It removes the person from *BackgroundList*, if they are a part of it, and adds the current target, if it exists, to the *BackgroundList*. The mechanics of what occurs when FR finds the current target is similar to what occurs when it finds a background target, it publishes to the scheduler who it has found and provides it with more information.

If the Current Target is unknown to FR, then it will ask the people who it recognises but does not have encodings for, what their names are. The process is shown in Figures 14 and 15. When it publishes that it has found a person, FR locks until it is told that it can search again, just to prevent it from recognising people when interaction is delivering a task.

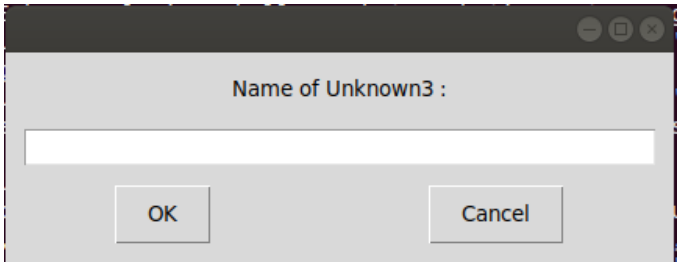

Fig. 14. FR asking for an unknown person's name to check if they are the current target

Removal of a Target: When Interaction has delivered a message, it publishes the person it just finished with and the Scheduler publishes to the Facial Recogniser which person it

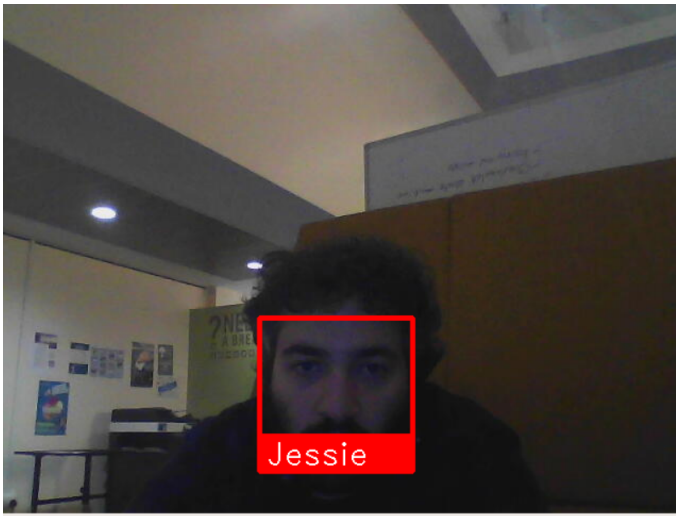

Fig. 15. FR's updated visuals after obtaining a name

can remove. If the person was in *BackgroundList*, then they are just removed without hassle, and if they were the *Current Target*, then *Current Target* is set to None until the scheduler states a current target.

3) Experimental Results

The testing of FR integrated with the Scheduler was tested in *C*, so this section will just test the Facial Recognition in isolation.

		Who FR recognised them as									
		Esha	Tom	Alexis	Joe	Papia	Jesse	Rehaan	Jaydip	Unknown	Accuracy
People for FR to recognise	Esha	21	2	1	0	0	0	0	0	1	0.84
	Tom	0	23	0	0	0	0	0	0	2	0.92
	Alexis	0	1	22	0	0	0	0	0	2	0.88
	Joe	0	2	0	19	0	0	0	0	4	0.76
	Papia	2	0	1	0	21	0	0	0	1	0.84
	Jesse	0	0	0	0	0	25	0	0	0	1
	Rehaan	0	0	0	0	0	0	23	0	2	0.92
	Jaydip	0	0	0	0	0	0	0	23	2	0.92
	Unknown	1	1	0	1	2	0	0	1	19	0.76
										Ave:	0.871111

Fig. 16. FR's results after being asked to identify the person in the frame

As seen in Figure 16, the face-recognition package has an accuracy of 87.1 % when using the argmin of facial encodings to identify the person. This was tested when the person's face was fully in the frame and unobstructed, and are the immediate first classifications. After a few frames, the FR recognises everyone correctly if it can see their entire face.

It has surprising accuracy when a person's face is only partially in the frame, identifying the person correctly 74.8% of the time as they enter the frame. However, it struggles when the face is in the frame fully but is covered by example, a hand. The model requires significantly longer to decide on a stable label, and has an accuracy of 53.5%.

4) Challenges

The primary challenge for facial recognition was that when the frames were captured from the camera and analyzed, it blocked any updates to variables such as *Current Target* and *BackgroundList*. Therefore, every time the scheduler wanted to update any information or the recogniser found a person, it needed to release the capture and shut recognition down before restarting with the new information. This slowed down the communication between the scheduler and the facial recognition node to the point where, if tasks were being given at a rate quicker than one per ten seconds (unlikely, given Interaction's conversation time), the recognition node would be searching for a different current target to what the scheduler thought was the current target. However, this would not cause much of an issue, the scheduler would always have the up to date information and data from the recogniser would pass through the scheduler before going to interaction, so any discrepancies such as finding a person who no longer needed to be found would be caught.

In addition, similar to Object Recognition, Facial Recognition required the use of a camera, one differet to Object Recognition as they have to look in different directions (Object recognition looks at Leonard's platform, while Facial recognition looks ahead). However, during use, the system indices for distinguishing the available cameras would change leading to all captures failing and the nodes crashing. It was possible to fix one camera as the default and that did not change often, but the other changed intermittently.

G. RRT

1) Introduction

Rapidly-exploring Random Trees, RRT, is an incremental sampling based algorithm that efficiently searches a non-convex, high-dimensional space by randomly building a tree between start and goal positions. The reasons for using RRT were:

- RRT can quickly explore the full map rather than being limited to exploring only areas near the starting node
- It is an incremental algorithms meaning that it is a good fit with the real time implementation while still being probabalistically complete
- It has an exponential rate of decay for the probability of failure
- The map could be very cluttered or twisted, but explicit complex constraints would not be necessary as they would be checked per trajectory [4]

Additionally, it is relatively simple to program and can be easily optimised, giving rise to extended algorithms such as RRT*. Limitations, however, include it weak ability to deal with dynamic obstacles (such as people) and narrow passages. However, edge cases such as these were handled by obstacle avoidance and recovery.

2) Algorithms

The RRT algorithm builds up a random tree by sampling random positions and connecting them if there exists a straight edge free of obstacles between them. It starts by initializing a graph with the start node as its only vertex, and then samples a random position in the environment. This random position is generated in the clear space, which is distinguishable by an occupancy grid. It samples the goal with a small probability so the algorithm will eventually terminate. For this chosen point, it finds the nearest vertex in the graph and checks if that speculative edge is collision-free. If it is, the vertex and edge are added to the graph until the goal is reached [5].

Some further properties added to RRT's functionality to better acclimate it for its intended environment was to check whether the start node and the goal nodes, or any intermediate nodes, could connect directly. If it could, RRT finds its full path and the algorithm would terminate. If not, the algorithm would proceed as normal.

Additionally, another step was included to remove some noise in the path generated by RRT due to a combination of the randomness of the point selection, as well as the greedy nearest-neighbor selection by which the tree is built. To do this one of the key steps from RRT* was included, in which the parent of a sampled node is decided by the highest visible parent of the nearest node, rather than just the nearest node itself. This creates much smoother and shorter traversal paths.

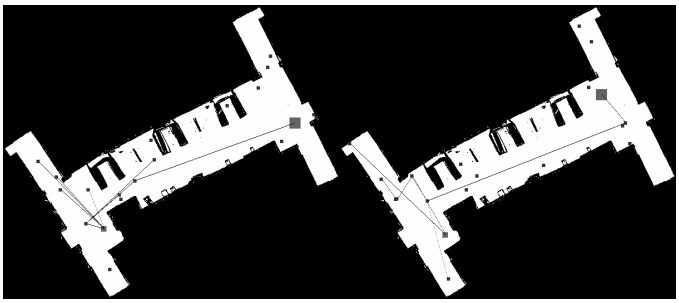

Fig. 17. A comparison of RRT with improvements(left) and without (right)

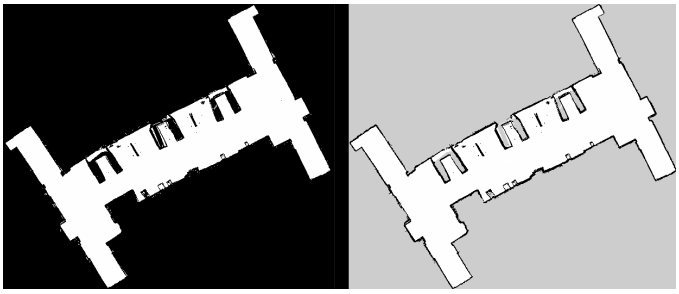

Fig. 18. Creation of the occupancy grid from the SLAM map

Pre-processing, Visualization and Proof of concept:

Utility functions were created to visualise the RRT algorithm before they were run on the physical robot.

A SLAM generated image-map provided an occupancy grid to be used with Stage and RVIZ. For simplicity, the map was flattened from RGB to monochrome using the pillow python library. This image manipulation allowed the grid to be generated, where black values translated to occupied space and white represented unoccupied space. Figure 18 shows the occupancy grid.

The nodes RRT defined as the intermediate points were represented as squares upon the image, with edges to visualise the intended connection. This was done using a GUI, made using pyQT5, Figure 19 shows an example of two nodes being joined by RRT.

Obstacle avoidance and Recovery:

Leonard must deal with both static and dynamic obstacles. To achieve this, it splits laser readings from its sensor into 19 bands and takes the average of each band, while normalising

Fig. 19. A demonstration of the GUI showing the RRT vertices. The large square is the start node, the smaller, top-left, grey square is the goal.

the data to ensure that the readings lie in a sensible range. If a value in front of Leonard indicates an obstacle is near by dropping below a set threshold, the obstacle avoidance will override the RRT path following and force Leonard to rotate in the direction with the highest total readings (indicating the side with the most space available), and move forwards by a small amount. At this point RRT will recalculate a path for Leonard to follow, and control is returned to the path following code.

3) Experimental Results

To test the navigation, Leonard was given a range of start and end vertices and its accuracy was measured when the navigation node was certain it had reached its destination.

TABLE II
A TABLE SHOWING THE RESULTS OF LEONARD NAVIGATING

Attempt	Result	Distance (cm)	Reason
1	Fail	-	Stuck in chairs
2	Fail	-	Hit a person's shoes
3	Success	18	-
4	Success	7	-
5	Success	19	-
6	Fail	-	Hit a wall
7	Success	8	-
8	Success	26	-

As shown in Table II, Leonard navigates to its goal successfully with a probability of 62.5%, with the distance from its goal belonging to a distribution with mean 15.6cm and a std-dev of 7.17 cm.

4) Challenges

A major challenge for navigation was the movement component. There is no way to specify an exact amount for

the robot to turn when still, therefore, it must be told to rotate until it is facing roughly the correct direction based on its estimated location and bearing. This adds uncertainty to movement because it is dependent on the accuracy and update frequency of the localisation component. This is compounded by the fact that the image space has a different coordinate system to RVIZ, so transforms must be applied to rotate and scale one to the other.

Localisation posed another challenge. Initial implementation of localisation used an optimised version of a DBscan based on the localisation algorithm created in assignment 2. Despite optimisation, over time, the reliability of the algorithm diminished, and became subject to inaccuracy and the kidnapped robot problem. The AMCL ROS package also presented some inaccuracies, however was significantly better at handling kidnapping. In the interest of maintaining a robust system, it was decided to utilise AMCL, to better demonstrate the functionality of the other components and benefit the integration of the full system.

IV. Experimental Results for the Integrated System

A. Data

TABLE III
A table showing Number of Tasks Completed Before Failure

Attempt	Tasks Completed	Reason for Failure
1	3	Stuck in chairs
2	5	Hit a person
3	2	Localisation issue
4	4	Battery died
5	4	Stuck in chairs
6	6	Hit the whiteboard
7	7	completed all tasks
Average	4.43	-

During experimentation, Leonard was given a number of tasks, including a mix of object delivery and message delivery tasks. The average number of tasks that the robot was able to complete before a fatal issue occurred was 4.43, based on 7 experiments. The most common cause of failure was when the robot crashed.

B. Analysis

The success of the proposed personal assistant solution is evaluated by how well Leonard is able to perform its tasks and how many tasks it can execute. It performs far better when there are less dynamic obstacles, giving less chance for the localisation as well as the movement to become inaccurate. The chance of completing successive tasks also increases dramatically if the tasks are within a quarter to a third of the map size, indicating that giving a higher weight priority to proximity in the scheduler could improve performance.

C. Method Evaluation

As shown in table II, Leonard is successful in performing its principal objective of being a Personal Assistant by delivering both message and package tasks. However, a certain number of edge cases and environment dynamics caused the system to fail. The major uncertainties in the system comes from localization error and laser error, which also influences the localization. This allowed for Leonard's obstacle avoidance to be sub-par.

A future improvement to increase Leonard's robustness to the dynamic environment would be to include multiple sensors, in conjunction with the laser, such as the camera, so that it can avoid objects that the laser was inconsistent in registering.

D. Challenges

An issue that emerged was in the ease of launching the system. With the large number of node files and packages which needed to be launched, it became time consuming and difficult to execute them in order. Therefore, a launch file was created to launch all ROS packages, custom node.py files, and the configuration of RVIZ.

Another challenge that was faced in the full system was occasional hardware bugs that had a knock-on effect to the rest of the system. For instance, there were consistent cases with loose cable connections which meant the time taken to get the system running varied.

V. Conclusion

In conclusion, all basic requirements of the system were successfully implemented. Facial Recognition was identified, in the project proposal, as a possible extension. This was implemented fully. The Reinforcement Learning extension was adapted to better fit the scenario. During integration, while each component was functional, it was clear that there was some room for refinement. For future improvements, it would be sensible to reduce the size of the MVP, with more focus invested on the basic functionality of the robot, before implementing further extensions. This would have ensured a more user friendly result.

Appendix A
Links

The full source code for the system can be found here - https://github.com/tdmurphy/IntelligentRobotics_Leonard
A link to the video showing the integrated system can be found here - https://drive.google.com/file/d/10M_PF-8TRMyXg7nNEpTldA3V3SywZ1bu/view?usp=sharing

Appendix B
Final System Architecture

Figure illustrating the final system architecture and the interactivity between nodes via the topics that they publish and/or subscribe.

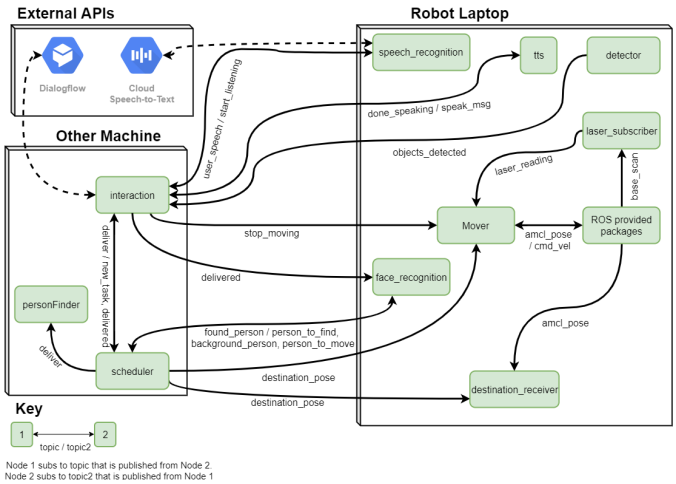

Fig. 20. System architecture diagram with topics

REFERENCES

[1] Bradski, G. *The OpenCV Library* Dr. Dobb's Journal of Software Tools. 2000
[2] Moses Olafenwa and John Olafenwa, *ImageAI*, 2018
[3] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[4] Yoshiaki Kuwata, Gaston A. Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P. How, "Motion Planning for Urban Driving using RRT"
[5] I. Noreen, A. Khan, Z. Habib, "A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms", IJCSNS VOL.16 No.10, October 2016
[6] Holthaus, Patrick Wachsmuth, Sven. (2014). The Receptionist Robot. ACM/IEEE International Conference on Human-Robot Interaction. 10.1145/2559636.2559784.
[7] Team Stuart 2017 final project entry https://www.cs.bham.ac.uk/internal/courses/int-robot/halloffame/2017/Stuart/