University of Toronto

# MIE443: Finding and Interacting with Emotional People in an Unknown Environment

**Group 4 Members**:

Mariem Ahmed

Katrina Cecco

Kevin Han

Anh Kiet Nguyen

Wenhan Jiang

Date of Submission: April 15th, 2021

# 1. Problem Definition

## 1.1. Objectives

The objective of Contest 3 is to explore and find 7 victims in an unknown disaster environment and provide them with important emergency announcements regarding evacuation due to fire. The robot should be able to explore and map the environment in order to find all the victims, identify their current emotional state and appropriately interact with them based on these emotions. The team will design the simulated TurtleBot to appropriately identify and react to the following 7 user emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral. The simulated TurtleBot will interact with the victim using the primary (stimulus reaction) emotions and secondary (deliberative) emotions. Each robot emotion must be distinct, and it cannot be duplicated amongst the two categories of emotions.. The whole process should not take more than 20 minutes.

## 1.2. Requirements and Constraints

- The robot should be able to function in any randomly generated environment.
- The team is provided with the following:
  - A dataset [1] of faces expressing one of the seven emotions (figure 1)
  - The emotions are represented with integers in the code (0 - Anger, 1 - Disgust, 2 - Fear, 3 - Happiness, 4 - Sadness, 5 - Surprise, 6 - Neutral)
  - A sample code to train and use an emotion classification convolutional neural network (CNN)
  - A victim detector that automatically detects simulated victims
  - A frontier exploration algorithm for your team to use to find the victims

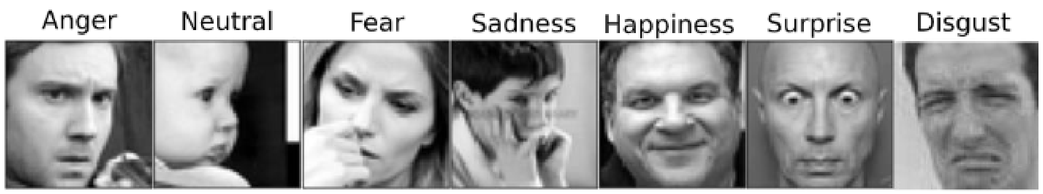


***figure 1: Example Environment Layout with 7 victims [2]***

- The type of disaster should be defined by the team and all the victims should be informed by the robot with evacuation instructions.
- The robot must have unique and distinct emotional responses to the 7 emotions of the victims to aid with conveying the information, which include 3 unique primary, 3 unique secondary, and 1 additional unique emotion based on the facial expression of the victims.

- The robot's emotions must be chosen from the list provided below (Table 1):

**Table 1: List of Robot's Emotions Response [2]**

| | |
|---|---|
| a. Fear | g. Hate |
| b. Positively excited | h. Resentment |
| c. Pride | i. Surprise |
| d. Anger | j. Embarrassment |
| e. Sad | k. Disgust |
| f. Discontent | l. Rage |

- The team must implement original motion and/or sound for the emotional interaction part of the contest.
- The robot must locate and interact with all victims in 20 minutes max.

## 2. Strategy

The robot's ability of emotion classification is accomplished by the trained Convolutional Neural Network (CNN) model. This is a class of deep neural networks that is most commonly applied for analyzing visual images and recognizing the features within the images. A complete CNN network usually includes a convolutional layer, a pooling layer and a few hyperparameters such as batch size, number of epochs, and learning rate, etc, which determine the model's accuracy to a large extent [3]. For Contest 3, the accuracy of the CNN model is improved by adjusting the hyperparameters.

The whole process of recognizing the victims' emotions are as follows:

- The victims' locations are stored in victims.csv files in the form of (`X position, Y position, radius`). The robot begins with positioning the victims using `victimLocator.py`. A victim is considered to be located if it is within a certain distance (`radius`) to the robot. Once a victim has been located during exploration, the `victimLocator.py` publishes a message containing a set of images for one emotion.

- After locating a victim, the robot begins predicting the emotion of the victim: the emotion of the victim is predicted using Convolutional Neural Network. When locating the victim, the emotionsub function is called and a series of images representing the victims are published. The CNN classifies each of the images into an emotion and votes on what emotion the set may be showing. These votes are then counted and the maximum vote-getting emotion is determined to improve the accuracy of the prediction.

- The CNN structure defined in the `emotionTrainingSample.py` function is used for classifying the emotions. The convolutional part of the network is stored in the `self.cnn` variable. After the CNN extracts the features, the fully-connected part is stored by the `self.nn` variable.

Cross-validation is a model validation method for examining how a statistical analysis result can apply to general independent datasets [4]. It is mostly used for models with goals of prediction and determining how accurate a prediction will be in practice. Thus, cross-validation would be the best choice for evaluating how accurately the CNN model for Contest 3 recognizes the victims' emotions based on their faces [4]. Under most circumstances, a dataset of known data will be given for training the parameters of the model, and another unknown dataset is used for evaluating the accuracy of the model [4]. The goal of cross-validation is to examine the model's performance for predicting new data, which was never seen by the model during training [4]. It, therefore, shows the model's performance in recognizing new faces, while preventing issues such as overfitting or selection bias [4].

After determining the emotion of each victim, there will be a multi-modal robot emotional response that is appropriate to the given disaster scenario of fire in a building. The TurtleBot will display images, play reactionary sounds, give evacuation instructions, and perform motions, all of which will be representing suitable primary and/or secondary responses. The various scenarios and emotional responses will be outlined below.

## 3. Robot Design and Implementation

In Contest 3, the hardware of the robot is already provided. This contest focuses on integrating with existing hardware and developing a controller to successfully complete the objectives for this project. This section will address the use of particular hardware and controller architecture for sensing, navigation, and mapping.

### 3.1. Sensory Design

The TurtleBot used in this course is divided into 2 main components: the Kobuki mobile base and the Simulated Microsoft Kinect 360. Each of these two contains multiple sensors. The Kobuki base includes 2x wheel drop sensors, 3x front bumpers, odometry, a gyroscope and 3 Cliff sensors, while the Kinect 360 is composed of an RGB camera and a depth sensor, shown in figure 2. The key sensors are the Kinect RGB camera, Odometry, and Gyroscope, which will be used for the move_base package to explore the environment and locate the 7 victims.

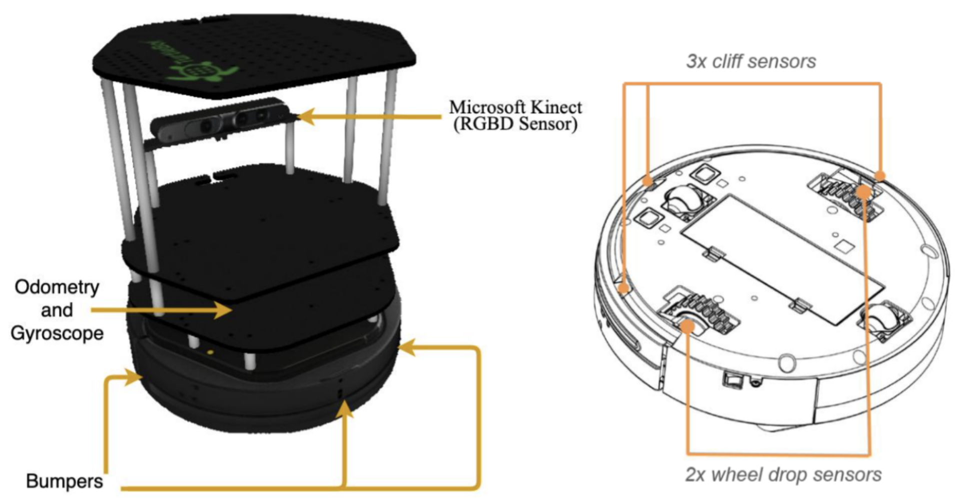


*figure 2: TurtleBot Hardware Components Overview [5]*

### 3.1.1. Kobuki Base

#### 3.1.1.1. Bumper

There are three bumpers on the kobuki base, each located on the center, right and left side of the simulated Turtlebot, as shown in figure 3. The bumper sensor is triggered whenever each one of the three is pressed, or the Turtlebot runs into an obstacle.

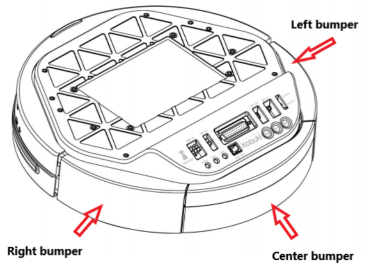


*figure 3: Kobuki base & location of the three bumpers [5]*

#### 3.1.1.2. Odometry

Odometry will be used to approximate the location and orientation of the Turtlebot simulated in relation to a starter position in x, y, and z (phi) position, and to update the Turtlebot position in the map.

#### 3.1.1.3. Cliff Sensor

Three Cliff sensors are implemented at the left, front-center, and right of the robot for searching for cliffs actively and preventing the robot from falling off steps or steep ramps. It also helps the robot to measure the subtle height difference between low-level obstructive objects and low-level non-obstructive objects. The sensor's state turns to 1 when a cliff is detected and stays at 0 otherwise.

#### 3.1.1.4. Gyroscope

The Gyroscope measures the angular velocity and orientation of the robot. It will be used for the move_base package when frontier exploration is occurring.

### 3.1.2. Microsoft Kinect 360 Sensor

The Kinect sensor will be used to locate the scene and capture images of each of the objects. The Kinect Sensor includes an RGB Camera, and a Depth Sensor (which contains the IR laser emitter and IR depth sensor), as shown in figure 4.

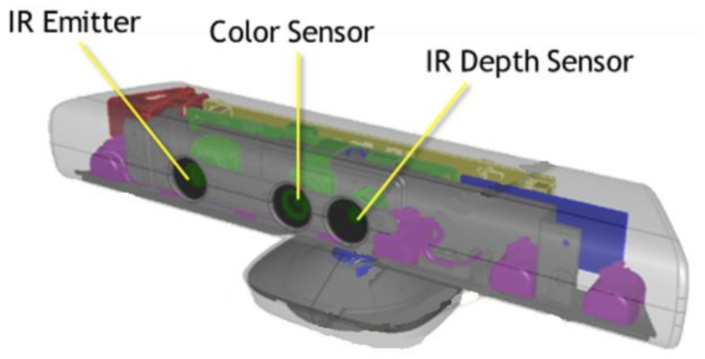


***figure 4: Microsoft Kinect 360 sensors [5]***

#### 3.1.2.1. Depth Sensor

A depth sensor composed of an infrared projector and a monochrome CMOS sensor is implemented in the robot for acquiring a deformed pattern of the environment based on the geometric shape. The infrared projector, also presented as an IR emitter in figure 4, will generate the laser scan, while the IR depth sensor will detect the IR beams reflected back to the sensor. Therefore, the robot is able to compute the x, y, z coordinates of each point in the observed pattern and obtain a 3D point plot of the environment. In contest 3, the Depth Sensor plays an important role in the robot's navigation, since the Frontier exploration algorithm with Gmapping will use the laser scan generated by the depth sensor to map the environment.

## 3.2. Controller Design

### 3.2.1. High-level control

**Frontier Exploration:** In this contest, the team is provided with a frontier exploration algorithm to explore the map and find the seven victims. Frontier exploration is implemented using the *Explore_lite* package provided by ROS. During the exploration process, the ROS *Gmapping* package will generate a map, and the *Explore_lite* package will implement a greedy frontier-based exploration. This means that the robot will move to unexplored areas on the map, and greedily explore its environment until no more frontiers (the boundary between known and unknown regions) could be found, or there are no more unmapped regions.

Since the team's goal is to use frontier exploration to find the seven victims and then interact with them, we will stop the frontier exploration using the `explore.stop()` command after a victim emotion is published (a victim is found) by `victimLocator`, so that the robot will not move away during the interaction. After the robot finishes its interaction, we will turn the frontier exploration back on with `explore.start()` to continue its exploration until all seven victims are found.

During the testing process, our team encountered a problem that the robot unexpectedly stopped while exploring the environment with the provided frontier exploration algorithm. In order to solve this issue, we designed a function called `unstuckFrontier` to make the robot rotate in a small circle around the spot for a while whenever the `explore.cpp` made the robot stop the frontier exploration without any victims located yet. Since this issue only appeared to occur when the robot started in a large, concave room without much information on the scanned map, this mechanism is only there to be triggered in the beginning. Once the frontier exploration started again right after, the robot had more scanned data to work with and gained momentum to continue exploring until the entire map was traversed.

**`victimLocator.py`** and **`emotionClassifier.py`:** *victimLocator.py* publishes the victim coordinates and images of the victims' emotions to *emotionClassifier.py*. When the Turtlebot reaches within the area of a victim, *emotionClassifier.py* will receive several facial expression images of the same emotion and classify them into a determined emotion using the CNN training model. The most commonly determined emotion is the output message from the *emotionClassifier.py,* which will then be called by the `emotionCallback` function in the main contest file for the low-level control.

**`robotResponse:`** We defined this function for showing the emotional responses of the robot. The robot has 7 responses corresponding to the 7 emotional states of the victims (Table 1). In this function, the robot will receive the output results of `emotionCallback`, which is the CNN classified victims' emotions, and return the corresponding robot's responses to these emotions. Each of the robot's responses will contain sound and image of the appropriate emotional response to the victim's emotion, robot's movement to further demonstrate its emotional response, and an evacuation instruction for the victim to escape from the fire disaster, with the tone and message matched with its current emotion. Further detail will be discussed in section *3.2.4. Robot Emotional Response.*

### 3.2.2. Low-level control

**Laserscan:** as previously mentioned, the laser scan is generated by the depth sensor. It will be used in the frontier exploration along with Gmapping to help the robot scan and map the environment it is exploring.

**`move_base:`** is utilized in the frontier exploration as it takes the navigation goals output from the frontier exploration algorithm and moves the robot to that goal location. The combination of `move_base` and Gmapping allows simultaneous mapping and localization within an environment.

**`emotionCallback:`** this function will subscribe to the emotionClassifier.py file, and return the detected emotion of the victim.

**`frontierCallback:`** this function detects when the frontier exploration stops. The output of this function is a boolean, and whenever the boolean is *True*, `unstuckFrontier` can be called to get the Turtlebot out of the stuck position due to errors in the existing frontier algorithm.

**`unstuckFrontier:`** as mentioned previously, this function is called whenever the robot stops working during its frontier exploration. The function will recall the `ros::spinOnce()` and make the robot rotate for a specific time to restart the scanning of the robot.

**`outputResult:`** this function stores the results after each victim is found, where it opens, writes and saves the victim's emotions, robot's emotional responses and the responses' type to an output text file.

The flow chart below demonstrates the full process of the controller design, and how the high-level and low-level control interact with each other.

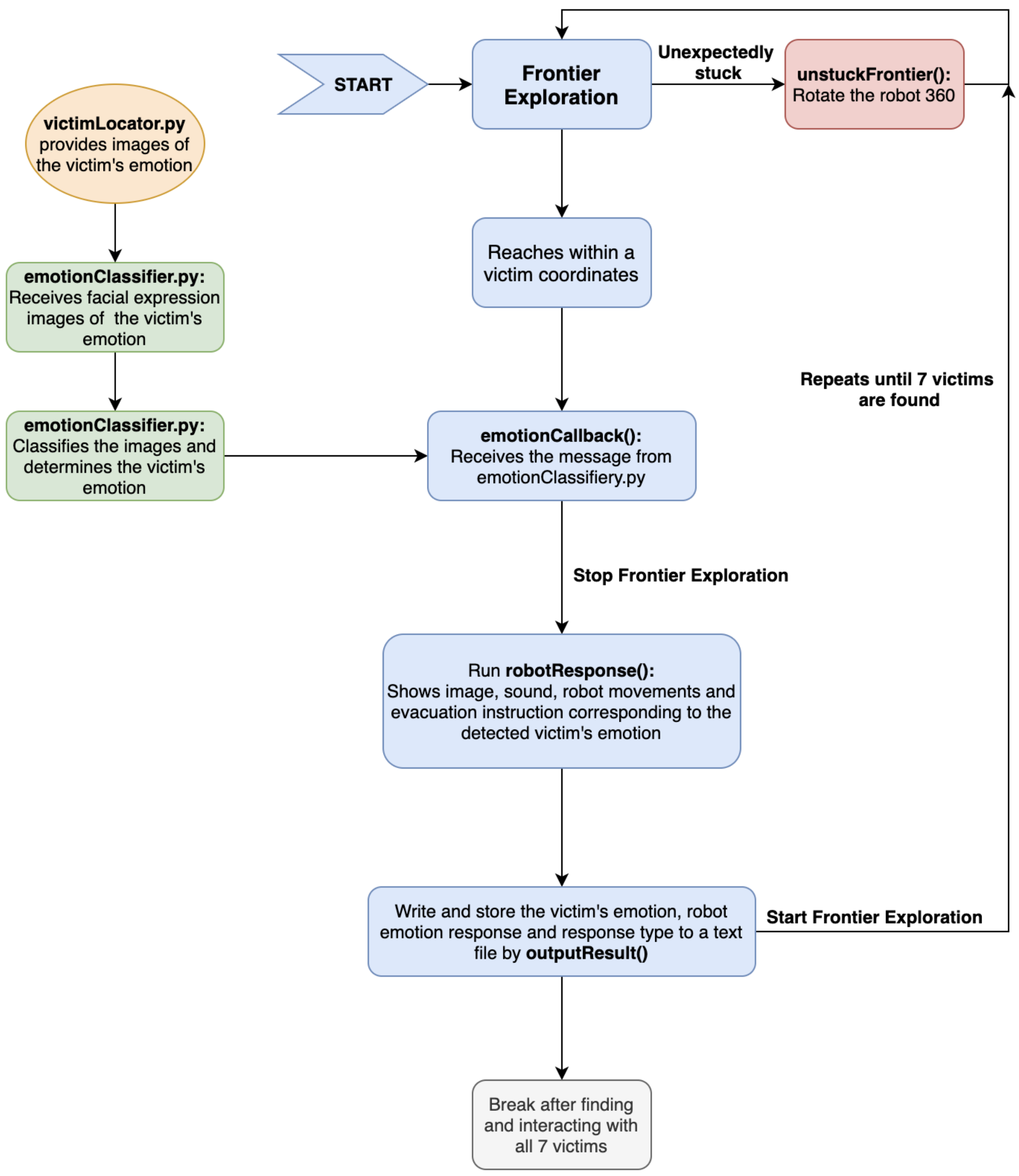

START
Frontier Exploration
Unexpectedly stuck
unstuckFrontier(): Rotate the robot 360
victimLocator.py provides images of the victim's emotion
emotionClassifier.py: Receives facial expression images of the victim's emotion
emotionClassifier.py: Classifies the images and determines the victim's emotion
Reaches within a victim coordinates
Repeats until 7 victims are found
emotionCallback(): Receives the message from emotionClassifiery.py
Stop Frontier Exploration
Run robotResponse(): Shows image, sound, robot movements and evacuation instruction corresponding to the detected victim's emotion
Write and store the victim's emotion, robot emotion response and response type to a text file by outputResult()
Start Frontier Exploration
Break after finding and interacting with all 7 victims

***figure 5: Flowchart for Controller Design***

### 3.2.3. Classifying victims' emotions

Our team tried to train the Baseline Model (CNN) with different parameters including input layer shape, CONV layer, fully connected layer, batch size, n_batch_loss and output layer. A full list of our attempted parameters change is shown in Appendix B. As a result, we got 65% of validation accuracy (figure 6) and 63% of validation accuracy, there is an increase of 2 percent of the validation accuracy.

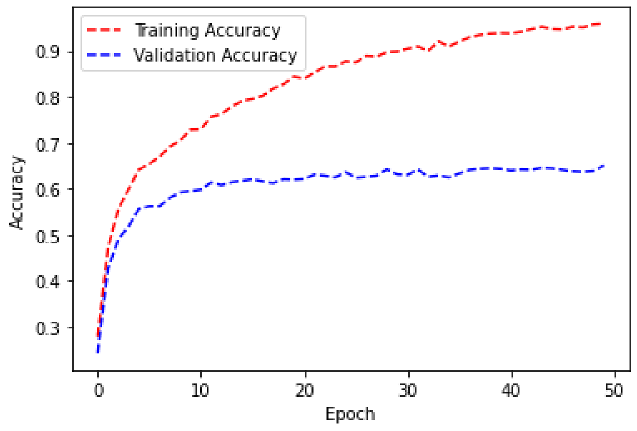


*figure 6: 65% Validation Accuracy*

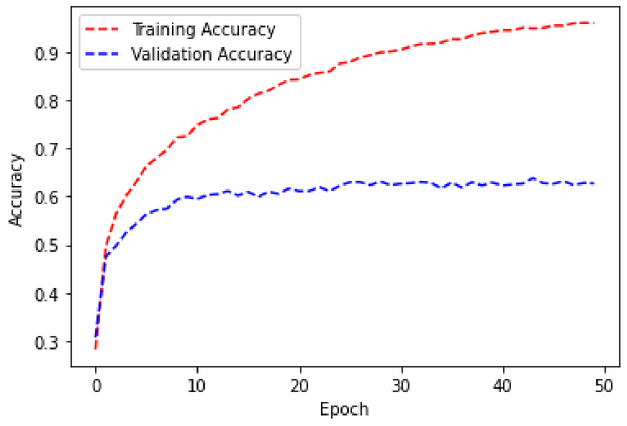


*figure 7: 63% Validation Accuracy*

We also experimented with different dropout rates. High dropouts are used to introduce variety into the dataset by getting rid of random nodes [3]. This is beneficial for our relatively small training set to prevent overfitting. Overall, it was found that the best result occurred when the first layer dropout was set to 0.0 (employing the whole input dataset) and subsequent layers had dropouts of 0.4. This combination resulted in a validation accuracy of 62% as shown in figure 8. This was still not better than the above approach.

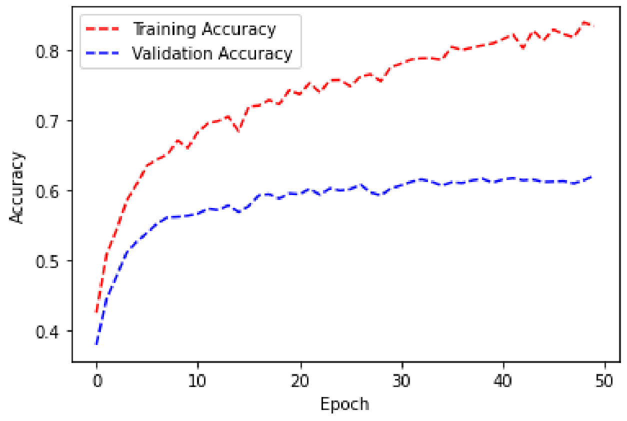


*figure 8: Best validation accuracy from dropout changes*

To try and improve the performance of the CNN, k-fold cross-validation was attempted. An extra argument `num_folds` was added to the training model. The dataset was split into `num_folds` equally-sized subsets or 'folds'. The cross-validation method alternates which fold is selected as the validation set, training on the other `num_folds-1` sets. The results are averaged across all folds to create a model that has ‘learned’ from the whole spread of the original dataset. In the interest of time, `num_folds` was selected to be 5, corresponding to a validation dataset size of roughly 20%.

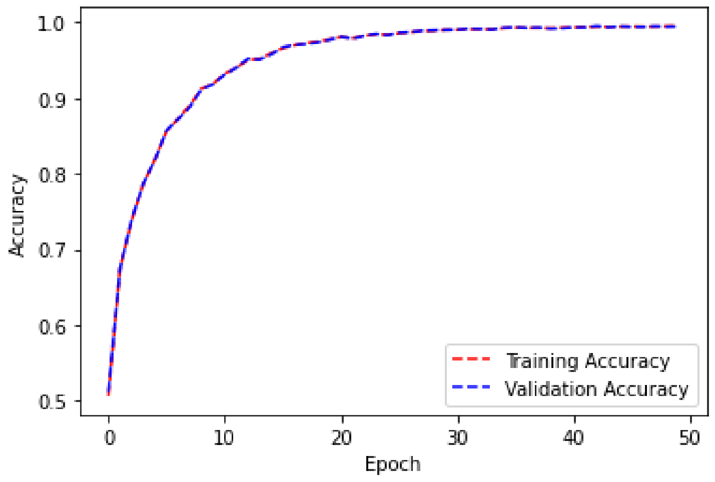


***figure 9: Results of 5-fold cross-validation***

The results of the cross-validation implementation are shown in figure 9. The results are a little strange because the training accuracy was almost identical to the validation accuracy throughout training. This is probably due to an error in the algorithmic design - rather than averaging the results of *k* separate models, the *k* folds were used to train one ‘averaged’ model. As a result, the training and validation sets got mixed in the process.

Fixing this error and improving the cross-validation technique are future recommendations for this contest. Due to the errors with our implementation of cross-validation, we used the tuned base model for our final submission instead.

#### 3.2.4. Robot Emotional Response

Our team chose fire to be the disaster in this situation. We have come up with seven unique emotional responses for the robot to react with the seven victim’s emotions, four of which are secondary and three are primary. Detailed emotional responses are shown in Table 2 below.

**Table 2: Robot’s emotional responses**

| **Victims’ Emotions** | **Robot’s Responses** |
|---|---|

| | |
|---|---|
| Neutral | Surprised (Primary) |
| Anger | Rage (Secondary) |
| Fear | Positively Excited (Secondary) |
| Sadness | Discontent (Secondary) |
| Happiness | Pride (Primary) |
| Surprise | Embarrassment (Secondary) |
| Disgust | Sad (Primary) |

The reason our team chose these corresponding responses are as follow:

- **Neutral victim**: the robot is surprised since the victim is so calm while stuck in the fire disaster. This is a primary emotion since it is a fast reactive mechanism without deliberation.
- **Angry victim**: the robot is taken aback initially by an angry victim who is full of entitlement to the rescuer, and it displays a secondary emotion of rage to make the victim follow the orders for evacuation. This is a secondary emotion to show an angry response and bitter indignation at being treated unfairly.
- **Fear victim**: the robot is positively excited to cheer up the frightened person. This is a secondary emotion since it is done with deliberation.
- **Sadness victim**: the robot initially ponders how to deal with a crying victim as a primary response, but decides to show discontent deliberately as a secondary response to the sad victim. This is to pull the victim out of the overwhelmed emotions and recall the person about the harsh reality of the fire disaster with more important actions to do.
- **Happiness victim**: the robot is proud because the person is happy to see someone coming to rescue. This is a primary emotion with no deliberation.
- **Surprise victim**: when the victim is surprised and screams to see the rescue robot, the robot initially is startled. Then it feels embarrassed for being scared as a secondary emotion since the robot did not expect to make the victim surprised by its appearance.

- **Disgust victim**: the robot shows sadness as a primary emotion since the victim just witnessed a dead body in the disaster and felt disgusted. The robot shows empathy to the victim who experienced tragedy first-hand.

In order to creatively show these emotional responses, our team uses pictures of a dog with seven different facial expressions corresponding to the seven robot's responses. In addition, we also incorporate sounds into the code while playing the image to make the demonstration more vivid.

Besides displaying sounds and images, the team also incorporates robot movements to further demonstrate the response. Table 3 below will discuss these movements in detail.

**Table 3: Robot's Movement Responses**

| **Victims' Emotions** | **Robot's Movements** |
|---|---|
| Neutral | Surprised (Primary): Twirl around in 360 degrees, swiftly direct the calm victim towards the exit |
| Anger | Rage: Taken aback, step back a little (primary), then quickly step up to direct orders and assert authority for the serious situation (secondary) |
| Fear | Positively Excited: Neutral, assessing the situation as the rescuer (primary), trying to cheer victim up and inject positivity and calm him down - shake back and forth, and then spin around 360 (secondary) |
| Sadness | Discontent (secondary): Feel sorry for the victim, assess the situation by going sideways back and forth (primary), step up forward and direct to snap victim out of being overwhelmed with emotion in an emergency situation (secondary) |
| Happiness | Pride (primary): Feel immense pride, and shows this attitude by spinning in 360 back and forth in both directions |
| Surprise | Embarrassment: Fast-shake (primary) to show the robot's startle by the victim's surprised scream, then turn 45 degrees to act embarrassed and shy of being startled (secondary) |

| | |
|---|---|
| Disgust | Sad: Rush of sadness as victim and robot both spot a dead body, Rotating back and forth slowly in empathy (primary) |

The evacuation instructions are to be played as sound files with appropriate language, sounds, and vocal intonations. We also make sure that these evacuation instructions follow and reflect the primary and secondary states of the emotion of the robot as mentioned in Table 3. Detailed discussion will be mentioned below in Table 4.

**Table 4: Robot's Evacuation Instructions**

| **Victims' Emotions** | **Evacuation Instructions** |
|---|---|
| Neutral | Huh?!!?! Wow, you're so calm! Good job! There's a fire in this building, so you need to keep your calm and evacuate through the staircase. You're doing so well! |
| Anger | Hey! You better calm down! This is nobody's fault! Follow my order and get the heck outta here! There's a fire in this building so quick, go out through the evacuation staircase NOW! |
| Fear | Hey there's nothing to be afraid of! I found you, I'm here for you! Just take a deep breath in and out. That's it. The building's on fire so we need you to evacuate quickly through the stairs okay? Don't worry, you got it! |
| Sadness | Hmmmm... Hey! Come on, stop being so down and stop crying like a baby - Let's get out of here. Snap out of it! Don't you want to see your family? Get moving! The building's on fire, so you need to evacuate through the staircase quickly. |
| Happiness | Ha-Hah!! Hey, I'm here for the rescue! Everything will be okay! There's a fire in this building, so you just need to stay level-headed and evacuate quickly! Thanks! |

| | |
|---|---|
| Surprise | Wooahahhoho! Your scream got me startled for a second there, haha... Alright, umm.. that's embarrassing. Yeah so the building's on fire so you need to quickly evacuate through the staircase. |
| Disgust | Oh... I'm so sorry you had to see that... Come on, the building's on fire so you need to evacuate quickly. Make sure to take the fire exit staircase. You'll be okay, alright? |

## 4. Future Recommendation

The team has been trying to improve the accuracy of the model by tuning the hyperparameters. It turns out that improving the accuracy of the model by simply trying different hyperparameters is inefficient and there are no specific rules to follow. Currently, the team's model has been able to achieve a validation accuracy of 65.03%, with 50 epochs, batch loss of 400 and batch size of 700. If more time were given, the team would pay more attention to the structure of the network and the functions used to see if the accuracy could be further improved. If possible, the team would also try models other than Convolutional Neural Network to see if a higher validation accuracy could be obtained.

The cross-validation method could be improved to determine the optimum number of folds. Similarly to how increased performance was achieved in the base model tuning by altering the probability of elements being added to the validation set (i.e. the resulting size of the validation set), it is possible that changing the number of folds (and hence the size of the sets that are being used for validation vs. training) may result in increased performance.

The frontier exploration algorithm can also be improved with more time in order to smoothly navigate the environment without any random stops and back-and-forth movements that were present in the current implementation.

The robot's emotional responses could be made more expressive by adding more modes of output, such as using video files or expanding the range of movements performed by the TurtleBot.

## 5. References

[1] I. J. Goodfellow et al., “Challenges in representation learning: A report on three machine learning contests,” in International conference on neural information processing, 2013, pp. 117–124.

[2] MIE443: Contest 3 (2021). *Finding and Interacting with Emotional People in an Unknown Environment*. Toronto: University of Toronto.

[3] MIE443: Tutorial 5(2021). Introduction to Convolutional Neural Networks. Toronto: University of Toronto

[4] P. A. Devijver and J. Kittler, Pattern Recognition: A Statistical Approach. Hoboken, NJ: Prentice Hall, 1982.

[5] *Kobuki User Guide*. 2017.

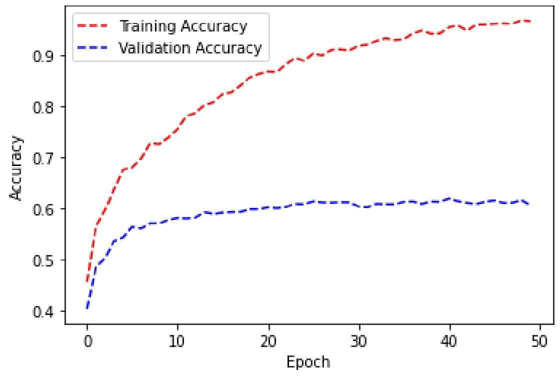
## 6. Contribution Table

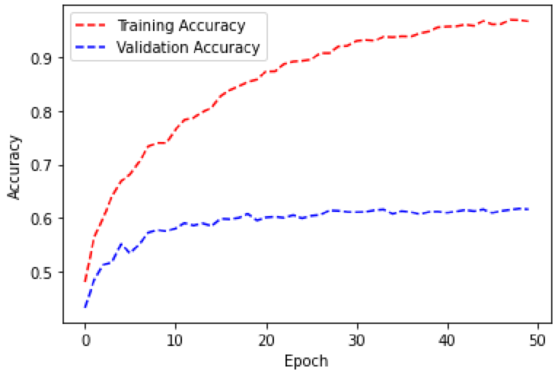
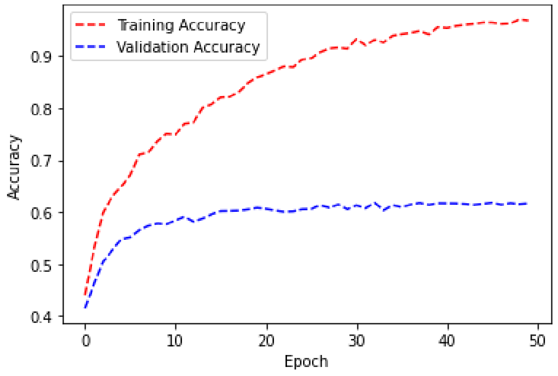
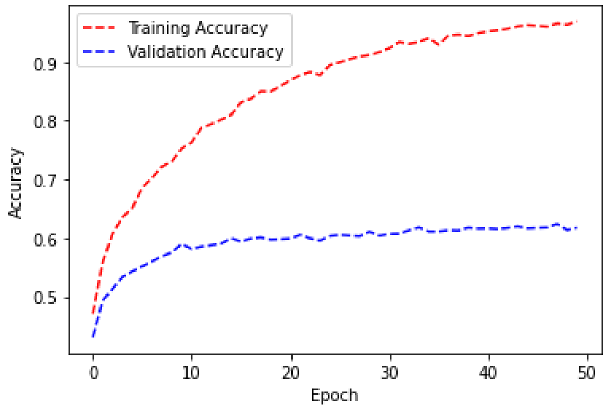
| | | **Katrina** | **Kevin** | **Anh** | **Mariem** | **Wenhan** |
|---|---|---|---|---|---|---|
| **Code** | `CNN baseline + cross-validation` | WC, TC, AD | | | TC, WC | |
| | `emotionCallback / emotionClassifier.py` | | WC, TC, AD | | | |
| | `frontierCallback / explore.cpp` | | WC, TC, AD | | | |
| | `unstuckFrontier` | | WC, TC, AD | | | |
| | `robotResponse` | | WC, TC, AD | WC, TC | WC, TC | |
| | `outputResults` | | WC, TC, AD | | | |
| | `main` | | WC, TC, AD | | | |
| **Report** | Problem Definition | | | | MR, RD | RD |
| | Strategy | | MR | | MR | RD, MR |
| | Sensory Design | | | MR | RD | RD, MR |

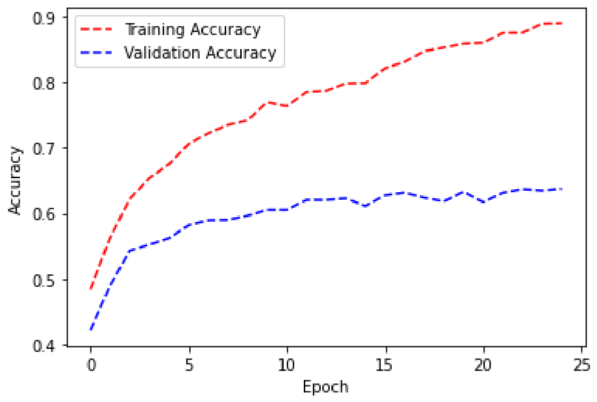
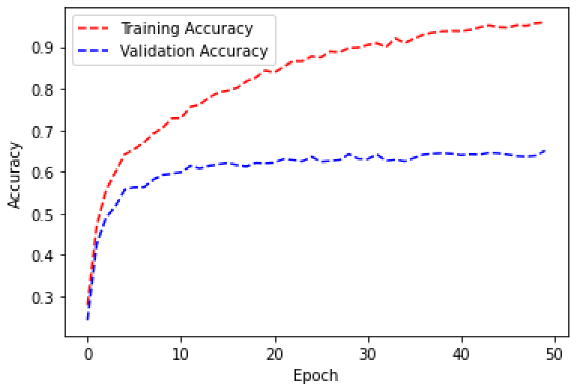
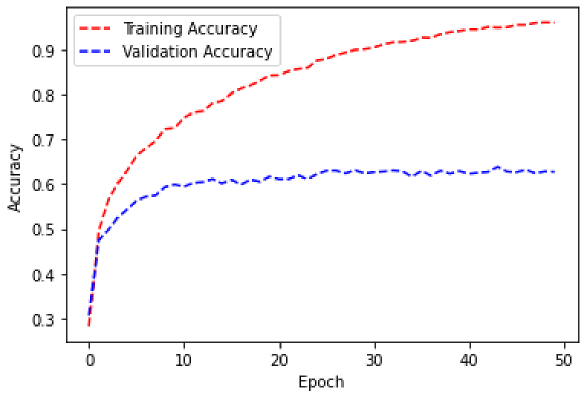
|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
|  | High-Level Controller Design |  | MR | RD, MR |  |  |
|  | Low-Level Controller Design |  | MR | RD, MR |  |  |
|  | Classifying Victims' Emotion | RD | MR |  | RD |  |
|  | Robot Emotional Response |  | MR | RD | RD |  |
|  | Future Recommendation |  | MR |  |  | RD |
|  | Overall Document | ET, FP | ET, FP | FP, ET | FP, ET | ET, FP |
| **Ideas for Emotional Response** | Images |  |  |  | IMG, ETI |  |
|  | Sounds |  | ETV, ETS, RECV |  | ETV, ETS, RECV |  |

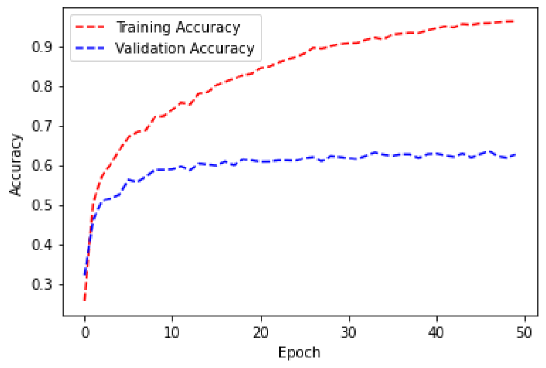
**RS** – research **RD** – wrote first draft **MR** – major revision **ET** – edited for grammar and spelling **FP** – final read **WC** - Written Code **TC** - Testing Code **AD** - Algorithmic Design **ETV** – edited for Voice **ETS** – edited for Sounds **RECV** – Recorded Voices and Instructions **IMG** – image choices **ETI** – edited for Images

## 7. Appendix A - Different Parameter

| epochs | N_batch_loss | Batch Size | code | Results |
|---|---|---|---|---|
| 50 | 50 | 128 | No change | **val ACC: 0.6068**  |
| 50 | 100 | 130 | No change | **val ACC: 0.6167** |

| | | | | |
|---|---|---|---|---|
| | | | |  |
| 50 | 50 | 150 | No change | **val ACC: 0.6170**  |
| 50 | 50 | 128 | `self.nn = nn.Sequential(`<br>`    nn.Linear(1152, 510),`<br>`    nn.ReLU(),`<br>`    nn.Dropout(0.25),`<br>`    nn.Linear(510, 256),`<br>`    nn.ReLU(),`<br>`    nn.Dropout(0.25),`<br>`    nn.Linear(256, 7)`<br>`)` | **val ACC: 0.6177**  |
| 25 | 350 | 200 | `    self.nn = nn.Sequential(`<br>`        nn.Linear(1152, 600),`<br>`        nn.ReLU(),`<br>`        nn.Dropout(0.25),`<br>`        nn.Linear(600, 300),`<br>`        nn.ReLU(),`<br>`        nn.Dropout(0.25),`<br>`        nn.Linear(300, 7)`<br>`    )`<br>`probs = torch.ones(train_imgs.shape[0]) * 0.1` | **val ACC: 0.6372** |

| | | | | |
|---|---|---|---|---|
| | | | |  |
| **50** | **400** | **700** | `class EmotionClassificationNet(nn.Module):`<br><br>`  def __init__(self):`<br>`    super(EmotionClassificationNet, self).__init__()`<br>`    self.cnn = nn.Sequential(`<br>`      nn.Conv2d(1, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br>`       nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br><br>`      nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br><br>`      nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br><br>`      nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.Dropout(0.25),`<br>`    )`<br><br>`    self.nn = nn.Sequential(`<br>`      nn.Linear(1152, 200),`<br>`      nn.ReLU(),`<br>`      nn.Dropout(0.25),`<br>`      nn.Linear(200, 500),`<br>`      nn.ReLU(),`<br>`      nn.Dropout(0.25),`<br>`      nn.Linear(500, 7)`<br><br>`    )`<br><br>`  def forward(self, x, test_mode=False):`<br>`    batch_size = x.shape[0]`<br>`    feats = self.cnn(x)`<br>`    out = self.nn(feats.view(batch_size, -1))`<br>`    #`<br>`    # If we are testing then return prediction index.`<br>`    if test_mode:`<br>`      _, out = torch.max(out, 1)`<br>`    return out`<br><br>`def getDataset(args):`<br>`  import pathlib`<br>`  #if pathlib.Path('./train_split.pth').exists():`<br>`  if pathlib.Path('/content/train_split.pth').exists():`<br>`    #train_imgs, train_labels = torch.load('train_split.pth')`<br>`    train_imgs, train_labels = torch.load('/content/train_split.pth')`<br>`    probs = torch.ones(train_imgs.shape[0]) * 0.1`<br>`    val_set_mask = torch.bernoulli(probs).bool()`<br>`    val_imgs = train_imgs[val_set_mask]`<br>`    val_labels = train_labels[val_set_mask]`<br>`    train_imgs = train_imgs[~val_set_mask]`<br>`    train_labels = train_labels[~val_set_mask]`<br>`    return (train_imgs, train_labels), (val_imgs, val_labels)`<br>`  else:`<br>`    print('The provided dataset does not exist!')`<br>`    exit(0)` | **val ACC: 0.6503**<br> |
| **50** | **400** | **700** | `class EmotionClassificationNet(nn.Module):`<br><br>`  def __init__(self):`<br>`    super(EmotionClassificationNet, self).__init__()`<br>`    self.cnn = nn.Sequential(`<br>`      nn.Conv2d(1, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br>`      nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br><br>`      nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br><br>`      nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.MaxPool2d(2, 2),`<br>`      nn.Dropout(0.25),`<br><br>`      nn.Conv2d(128, 128, 3, padding=1),`<br>`      nn.ReLU(),`<br>`      nn.BatchNorm2d(128),`<br>`      nn.Dropout(0.25),`<br>`    )`<br><br>`    self.nn = nn.Sequential(`<br>`      nn.Linear(1152, 200),`<br>`      nn.ReLU(),`<br>`      nn.Dropout(0.25),`<br>`      nn.Linear(200, 500),`<br>`      nn.ReLU(),`<br>`      nn.Dropout(0.25),`<br>`      nn.Linear(500, 7)`<br><br>`    )`<br><br>`  def forward(self, x, test_mode=False):`<br>`    batch_size = x.shape[0]`<br>`    feats = self.cnn(x)`<br>`    out = self.nn(feats.view(batch_size, -1))`<br>`    #`<br>`    # If we are testing then return prediction index.`<br>`    if test_mode:`<br>`      _, out = torch.max(out, 1)`<br>`    return out`<br><br>`def getDataset(args):` | **val ACC: 0.6272**<br> |

| | | | | |
|---|---|---|---|---|
| | | | import pathlib<br>#if pathlib.Path('./train_split.pth').exists():<br>if pathlib.Path('/content/train_split.pth').exists():<br>  #train_imgs, train_labels = torch.load('train_split.pth')<br>  train_imgs, train_labels = torch.load('/content/train_split.pth')<br>  probs = torch.ones(train_imgs.shape[0]) * 0.1<br>  val_set_mask = torch.bernoulli(probs).bool()<br>  val_imgs = train_imgs[val_set_mask]<br>  val_labels = train_labels[val_set_mask]<br>  train_imgs = train_imgs[~val_set_mask]<br>  train_labels = train_labels[~val_set_mask]<br>  return (train_imgs, train_labels), (val_imgs, val_labels)<br>else:<br>  print('The provided dataset does not exist!')<br>  exit(0) | |
| **same** | **same** | **same** | class EmotionClassificationNet(nn.Module):<br> def __init__(self):<br>   super(EmotionClassificationNet, self).__init__()<br>   self.cnn = nn.Sequential(<br>     nn.Conv2d(1, 64, 3, padding=1),<br>     nn.ReLU(),<br>     nn.BatchNorm2d(64),<br>     nn.MaxPool2d(2, 2),<br>     nn.Dropout(0.25),<br><br>     nn.Conv2d(64, 128, 3, padding=1),<br>     nn.ReLU(),<br>     nn.BatchNorm2d(128),<br>     nn.MaxPool2d(2, 2),<br>     nn.Dropout(0.25),<br>     nn.Conv2d(128, 128, 3, padding=1),<br>     nn.ReLU(),<br>     nn.BatchNorm2d(128),<br>     nn.MaxPool2d(2, 2),<br>     nn.Dropout(0.25),<br>     nn.Conv2d(128, 128, 3, padding=1),<br>     nn.ReLU(),<br>     nn.BatchNorm2d(128),<br>     nn.MaxPool2d(2, 2),<br>     nn.Dropout(0.25),<br>     nn.Conv2d(128, 128, 3, padding=1),<br>     nn.ReLU(),<br>     nn.BatchNorm2d(128),<br>     nn.Dropout(0.25),<br>   )<br>   self.nn = nn.Sequential(<br>     nn.Linear(1152, 512),<br>     nn.ReLU(),<br>     nn.Dropout(0.25),<br>     nn.Linear(512, 256),<br>     nn.ReLU(),<br>     nn.Dropout(0.25),<br>     nn.Linear(256, 7)<br><br>   )<br> def forward(self, x, test_mode=False):<br>   batch_size = x.shape[0]<br>   feats = self.cnn(x)<br>   out = self.nn(feats.view(batch_size, -1))<br>   #<br>   # If we are testing then return prediction index.<br>   if test_mode:<br>     _, out = torch.max(out, 1)<br>   return out<br>def getDataset(args):<br> import pathlib<br> #if pathlib.Path('./train_split.pth').exists():<br> if pathlib.Path('/content/drive/MyDrive/FOURTH YEAR MIE /Winter Semester /MIE443 /MIE443 Project /Contest 3/mie443_contest3/src/train_split.pth').exists():<br>   #train_imgs, train_labels = torch.load('train_split.pth')<br>   train_imgs, train_labels = torch.load('/content/drive/MyDrive/FOURTH YEAR MIE /Winter Semester /MIE443 /MIE443 Project /Contest 3/mie443_contest3/src/train_split.pth')<br>   probs = torch.ones(train_imgs.shape[0]) * 0.1<br>   val_set_mask = torch.bernoulli(probs).bool()<br>   val_imgs = train_imgs[val_set_mask]<br>   val_labels = train_labels[val_set_mask]<br>   train_imgs = train_imgs[~val_set_mask]<br>   train_labels = train_labels[~val_set_mask]<br>   return (train_imgs, train_labels), (val_imgs, val_labels)<br> else:<br>   print('The provided dataset does not exist!')<br>   exit(0) | **val ACC: 0.6257**<br> |

## 8. Appendix B - C++ ROS Code

```cpp
#include <ros/ros.h>
#include <ros/package.h>
#include "explore.h"
//
#include <fstream>
#include <string>
#include <opencv2/core.hpp>
#include <cv.h>
#include <cv_bridge/cv_bridge.h>
#include <opencv2/highgui.hpp>
// If you cannot find the sound play library try the following command.
// sudo apt install ros-kinetic-sound-play
#include <sound_play/sound_play.h>
#include <ros/console.h>

#include <iostream>
```

```
#include <chrono>
#include <geometry_msgs/Twist.h>
#include <mie443_contest3/EmotionMsg.h>
#include <std_msgs/Bool.h>
#define RAD2DEG(rad) ((rad) * 180./M_PI)
#define DEG2RAD(deg) ((deg) * M_PI / 180.)

// Victim Emotions: 0 -> Angry, 1 -> Disgust, 2 -> Fear, 3 -> Happy, 4 -> Sad, 5
-> Surprise, 6 -> Neutral
int detectedEmotion = -1; // initialize to negative value
// Detects when exploration is stuck
bool frontierEmpty = false;
// store emotion values for result output
std::vector<int> storeEmotion;



// Sets the victim emotion, subscribed to emotionClassifier.py
void emotionCallback(const mie443_contest3::EmotionMsg::ConstPtr& msg){
    detectedEmotion = msg->emotion;
}


// Detects when frontier exploration is stuck, subscribed to explore.cpp
void frontierCallback(const std_msgs::Bool::ConstPtr& msg){
    frontierEmpty = msg->data;
}


// Procedure to hopefully get frontier started again if it stops working
void unstuckFrontier(ros::Publisher vel_pub, geometry_msgs::Twist vel){
    ros::spinOnce();
    std::chrono::time_point<std::chrono::system_clock> scanStart;
    scanStart = std::chrono::system_clock::now();
    uint64_t scanDuration = 0;

    std::cout << "Scanning again" << std::endl;
    while (scanDuration < 6){
        ros::spinOnce();
        vel.angular.z = DEG2RAD(90);
        vel.linear.x = 0.5;
        vel_pub.publish(vel);
        scanDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- scanStart).count();
    }
    vel.angular.z = 0.0;
    vel.linear.x = 0.0;
    vel_pub.publish(vel);
}


// Show robot emotion in multi-modal response
// Three modes of response: 1. Image, 2. Sound, 3. Turtlebot Movement
void robotResponse(int emo_num, ros::Publisher vel_pub, geometry_msgs::Twist vel,
std::string path_to_sounds, std::string path_to_images, sound_play::SoundClient
*sc){
    // Victim: 0, Angry -> Robot: Rage (Secondary)
    // Victim: 1, Disgust -> Robot: Sad (Primary)
    // Victim: 2, Fear -> Robot: Positively Excited (Secondary)
    // Victim: 3, Happy -> Robot: Pride (Primary)
    // Victim: 4, Sad -> Robot: Discontent (Secondary)
    // Victim: 5, Surprise -> Robot: Embarrassment (Secondary)
    // Victim: 6, Neutral -> Robot: Surprised (Primary)
    ros::spinOnce();
    ros::Duration(1).sleep();

    std::chrono::time_point<std::chrono::system_clock> responseStart;
    responseStart = std::chrono::system_clock::now();
    uint64_t responseDuration = 0;

    // Victim: 0, Anger -> Robot: Rage (Secondary)
    if (emo_num == 0){
        std::cout << "Found an angry victim who's full of entitlement! Must talk
some sense into him!" << std::endl;

        // Primary stimulus - taken aback by the aggressive victim
        while (responseDuration <= 1){
            //std::cout << "check 1" << std::endl;
            ros::spinOnce();
            vel.angular.z = 0.0;
            vel.linear.x = -0.2;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }

        // Secondary Response - Step up to direct orders and assert authority for
the seriousness of the situation
        //sc.playWave(path_to_sounds + "Anger.wav");
        sc->playWave(path_to_sounds + "Anger.wav");
        cv::namedWindow("Victim Emotion: Anger & Robot Emotion: Rage",
CV_WINDOW_NORMAL);
        cv::setWindowProperty("Victim Emotion: Anger & Robot Emotion: Rage",
CV_WND_PROP_AUTOSIZE, CV_WINDOW_NORMAL);
        cv::Mat emo_image = cv::imread(path_to_images + "Anger.jpg",
CV_LOAD_IMAGE_COLOR);
        cv::resizeWindow("Victim Emotion: Anger & Robot Emotion: Rage", 500,500);
        cv::imshow("Victim Emotion: Anger & Robot Emotion: Rage", emo_image);

        while (responseDuration <= 2){
            cv::waitKey(1);
            //std::cout << "check 2" << std::endl;
            ros::spinOnce();
            vel.angular.z = 0.0;
            vel.linear.x = 0.2;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        // wait until audio is finished
        while (responseDuration <= 13){
            cv::waitKey(1);
            //std::cout << "check 3" << std::endl;
            ros::spinOnce();
            vel.angular.z = 0.0;
            vel.linear.x = 0.0;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        cv::destroyWindow("Victim Emotion: Anger & Robot Emotion: Rage");
    }

    // Victim: 1, Disgust -> Robot: Sad (Primary)
    else if (emo_num == 1){
        std::cout << "Found a disgusted victim who saw a dead body! Feels so sad
:(" << std::endl;
        // Primary response of sadness to the situation with disgusted victim who
experienced tragedy
        //sc.playWave(path_to_sounds + "Disgust.wav");
        sc->playWave(path_to_sounds + "Disgust.wav");
        cv::namedWindow("Victim Emotion: Disgust & Robot Emotion: Sad",
CV_WINDOW_NORMAL);
        cv::setWindowProperty("Victim Emotion: Disgust & Robot Emotion: Sad",
CV_WND_PROP_AUTOSIZE, CV_WINDOW_NORMAL);
        cv::Mat emo_image = cv::imread(path_to_images + "Disgust.jpg",
CV_LOAD_IMAGE_COLOR);
        cv::resizeWindow("Victim Emotion: Disgust & Robot Emotion: Sad",
500,500);
        cv::imshow("Victim Emotion: Disgust & Robot Emotion: Sad", emo_image);

        int direction = 1;
        bool firstTurn = true;

        while (responseDuration <= 12){
            // timer for back and forth turning
            cv::waitKey(1);
            std::chrono::time_point<std::chrono::system_clock> turnStart;
            turnStart = std::chrono::system_clock::now();
            uint64_t turnDuration = 0;

            if (firstTurn == true){
                while (turnDuration <= 1){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = DEG2RAD(30);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = -1;
                firstTurn = false;
            }
            else if (direction == 1){
                while (turnDuration <= 2){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = DEG2RAD(30);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = -1;
            }
            else {
                while (turnDuration <= 2){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = -DEG2RAD(30);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = 1;
            }
        }
        vel.angular.z = 0.0;
        vel.linear.x = 0.0;
        vel_pub.publish(vel);
        cv::destroyWindow("Victim Emotion: Disgust & Robot Emotion: Sad");
    }

    // Victim: 2, Fear -> Robot: Positively Excited (Secondary)
    else if (emo_num == 2){
        std::cout << "Found a victim in fear! Must get him positive and upbeat to
get out of here quick!" << std::endl;

        // Primary stimulus - neutral, assessing the situation as a rescuer
        while (responseDuration <= 1){
            ros::spinOnce();
            vel.angular.z = 0.0;
            vel.linear.x = 0.0;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }

        // Secondary Response - try to inject positivity and calmness to the
victim to facilitate the rescue
        //sc.playWave(path_to_sounds + "Fear.wav");
        sc->playWave(path_to_sounds + "Fear.wav");
        cv::namedWindow("Victim Emotion: Fear & Robot Emotion: Positively
Excited", CV_WINDOW_NORMAL);
```

```
        cv::setWindowProperty("Victim Emotion: Fear & Robot Emotion: Positively 
Excited", CV_WND_PROP_AUTOSIZE, CV_WINDOW_NORMAL);
        cv::Mat emo_image = cv::imread(path_to_images + "Fear.jpg",
CV_LOAD_IMAGE_COLOR);
        cv::resizeWindow("Victim Emotion: Fear & Robot Emotion: Positively 
Excited", 500,500);
        cv::imshow("Victim Emotion: Fear & Robot Emotion: Positively Excited",
emo_image);

        int direction = 1;
        bool firstTurn = true;

        // shake maneuver for showing overflowing excitement and positivity
        while (responseDuration <= 6){
            cv::waitKey(1);
            // timer for back and forth turning
            std::chrono::time_point<std::chrono::system_clock> turnStart;
            turnStart = std::chrono::system_clock::now();
            uint64_t turnDuration = 0;
            if (firstTurn == true){
                while (turnDuration <= 1){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = DEG2RAD(30);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration = 
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration = 
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                firstTurn = false;
                direction = -1;
            }
            else if (direction == 1){
                while (turnDuration <= 2){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = DEG2RAD(30);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = -1;
            }
            else {
                while (turnDuration <= 2){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = -DEG2RAD(30);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = 1;
            }
        }
        // spin 360 fast to show positive excitement
        while (responseDuration <= 14){
            cv::waitKey(1);
            ros::spinOnce();
            vel.angular.z = DEG2RAD(180);
            vel.linear.x = 0.0;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        vel.angular.z = 0.0;
        vel.linear.x = 0.0;
        vel_pub.publish(vel);
        cv::destroyWindow("Victim Emotion: Fear & Robot Emotion: Positively
Excited");
    }

    // Victim: 3, Happiness -> Robot: Pride (Primary)
    else if (emo_num == 3){
        std::cout << "Found a victim happy to see a rescuer! I feel proud of
myself!" << std::endl;
        // Primary response of pride showing as the victim treats you like a hero
        //sc.playWave(path_to_sounds + "Happiness.wav");
        sc->playWave(path_to_sounds + "Happiness.wav");
        cv::namedWindow("Victim Emotion: Happiness & Robot Emotion: Pride",
CV_WINDOW_NORMAL);
        cv::setWindowProperty("Victim Emotion: Happiness & Robot Emotion: Pride",
CV_WND_PROP_AUTOSIZE, CV_WINDOW_NORMAL);
        cv::Mat emo_image = cv::imread(path_to_images + "Happiness.jpg",
CV_LOAD_IMAGE_COLOR);
        cv::resizeWindow("Victim Emotion: Happiness & Robot Emotion: Pride",
500,500);
        cv::imshow("Victim Emotion: Happiness & Robot Emotion: Pride",
emo_image);

        int direction = 1;

        while (responseDuration <= 12){
            cv::waitKey(1);
            // timer for back and forth 360 spinning
            std::chrono::time_point<std::chrono::system_clock> turnStart;
            turnStart = std::chrono::system_clock::now();
            uint64_t turnDuration = 0;

            if (direction == 1){
                while (turnDuration <= 2){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = DEG2RAD(180);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = -1;
            }
            else {
                while (turnDuration <= 2){
                    cv::waitKey(1);
                    ros::spinOnce();
                    vel.angular.z = -DEG2RAD(180);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = 1;
            }
        }
        vel.angular.z = 0.0;
        vel.linear.x = 0.0;
        vel_pub.publish(vel);
        cv::destroyWindow("Victim Emotion: Happiness & Robot Emotion: Pride");
    }

    // Victim: 4, Sadness -> Robot: Discontent (Secondary)
    else if (emo_num == 4){
        std::cout << "Found a victim sad that he lost his stuff to fire! It's too
bad but we must snap him out of grief to get him out of here!" << std::endl;

        while (responseDuration <= 4){
            // timer for back and forth movement like pondering
            std::chrono::time_point<std::chrono::system_clock> turnStart;
            turnStart = std::chrono::system_clock::now();
            uint64_t turnDuration = 0;

            while (turnDuration <= 1){
                ros::spinOnce();
                vel.angular.z = 0.0;
                vel.linear.x = 0.1;
                vel_pub.publish(vel);
                responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
            }
            while (turnDuration <= 2){
                ros::spinOnce();
                vel.angular.z = DEG2RAD(180);
                vel.linear.x = 0.0;
                vel_pub.publish(vel);
                responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
                turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
            }
        }

        // Secondary Response - Step up to direct orders and assert authority for
the seriousness of the situation
        //sc.playWave(path_to_sounds + "Sadness.wav");
        sc->playWave(path_to_sounds + "Sadness.wav");
        cv::namedWindow("Victim Emotion: Sadness & Robot Emotion: Discontent",
CV_WINDOW_NORMAL);
        cv::setWindowProperty("Victim Emotion: Sadness & Robot Emotion:
Discontent", CV_WND_PROP_AUTOSIZE, CV_WINDOW_NORMAL);
        cv::Mat emo_image = cv::imread(path_to_images + "Sadness.jpg",
CV_LOAD_IMAGE_COLOR);
        cv::resizeWindow("Victim Emotion: Sadness & Robot Emotion: Discontent",
500,500);
        cv::imshow("Victim Emotion: Sadness & Robot Emotion: Discontent",
emo_image);

        // step up and snap the victim out of the emotion for the emergency
escape
        while (responseDuration <= 5){
            cv::waitKey(1);
            ros::spinOnce();
            vel.angular.z = 0.0;
            vel.linear.x = 0.2;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        // wait until audio is finished
        while (responseDuration <= 19){
            cv::waitKey(1);
            ros::spinOnce();
            vel.angular.z = 0.0;
            vel.linear.x = 0.0;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        cv::destroyWindow("Victim Emotion: Sadness & Robot Emotion: Discontent");
    }

    // Victim: 5, Surprise -> Robot: Embarrassment (Secondary)
    else if (emo_num == 5){
        std::cout << "Found a victim who's surprised that a rescuer found him! I 
got startled from his scream, now I slightly feel embarrassed" << std::endl;

        // Primary stimulus - robot also surprised and startled from the victim's 
scream - fast shake
        int direction = 1;
        // shake maneuver for showing startled robot
        while (responseDuration <= 4){
            // timer for back and forth turning
            std::chrono::time_point<std::chrono::system_clock> turnStart;
            turnStart = std::chrono::system_clock::now();
            uint64_t turnDuration = 0;
            if (direction == 1){
                while (turnDuration <= 1){
                    ros::spinOnce();
                    vel.angular.z = DEG2RAD(60);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration = 
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now() 
- responseStart).count();
                    turnDuration = 
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now() 
- turnStart).count();
                }
                direction = -1;
            }
            else {
                while (turnDuration <= 1){
                    ros::spinOnce();
                    vel.angular.z = -DEG2RAD(60);
                    vel.linear.x = 0.0;
                    vel_pub.publish(vel);
                    responseDuration = 
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now() 
- responseStart).count();
```

```
                    turnDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- turnStart).count();
                }
                direction = 1;
            }
        }

        // Secondary Response - turn 45 degrees to show that the robot is
embarrassed to be so startled by the scream
        //sc.playWave(path_to_sounds + "Surprise.wav");
        sc->playWave(path_to_sounds + "Surprise.wav");
        cv::namedWindow("Victim Emotion: Surprise & Robot Emotion:
Embarrassment", CV_WINDOW_NORMAL);
        cv::setWindowProperty("Victim Emotion: Surprise & Robot Emotion:
Embarrassment", CV_WND_PROP_AUTOSIZE, CV_WINDOW_NORMAL);
        cv::Mat emo_image = cv::imread(path_to_images + "Surprise.jpg",
CV_LOAD_IMAGE_COLOR);
        cv::resizeWindow("Victim Emotion: Surprise & Robot Emotion:
Embarrassment", 500,500);
        cv::imshow("Victim Emotion: Surprise & Robot Emotion: Embarrassment",
emo_image);

        while (responseDuration <= 5){
            cv::waitKey(1);
            ros::spinOnce();
            vel.angular.z = DEG2RAD(45);
            vel.linear.x = 0.0;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        // wait until audio is finished
        while (responseDuration <= 17){
            cv::waitKey(1);
            ros::spinOnce();
            vel.angular.z = 0.0;
            vel.linear.x = 0.0;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        cv::destroyWindow("Victim Emotion: Surprise & Robot Emotion:
Embarrassment");
    }

    // Victim: 6, Neutral -> Robot: Surprised (Primary)
    else if (emo_num == 6){
        std::cout << "Found a victim who is neutral and calm in this disastrous
environment! Wow good job!" << std::endl;
        // Primary response of surprise that the victim is staying calm,
encourage him/her for the escape
        //sc.playWave(path_to_sounds + "Neutral.wav");
        sc->playWave(path_to_sounds + "Neutral.wav");
        cv::namedWindow("Victim Emotion: Neutral & Robot Emotion: Surprised",
CV_WINDOW_NORMAL);
        cv::setWindowProperty("Victim Emotion: Neutral & Robot Emotion:
Surprised", CV_WND_PROP_AUTOSIZE, CV_WINDOW_NORMAL);
        cv::Mat emo_image = cv::imread(path_to_images + "Neutral.jpg",
CV_LOAD_IMAGE_COLOR);
        cv::resizeWindow("Victim Emotion: Neutral & Robot Emotion: Surprised",
500,500);
        cv::imshow("Victim Emotion: Neutral & Robot Emotion: Surprised",
emo_image);

        while (responseDuration < 11){
            cv::waitKey(1);
            ros::spinOnce();
            vel.angular.z = DEG2RAD(120);
            vel.linear.x = 0.0;
            vel_pub.publish(vel);
            responseDuration =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now()
- responseStart).count();
        }
        vel.angular.z = 0.0;
        vel.linear.x = 0.0;
        vel_pub.publish(vel);
        cv::destroyWindow("Victim Emotion: Neutral & Robot Emotion: Surprised");
    }
}


//Open, write, and save the stored results after each victim is found
void outputResult(std::vector<int> storeEmotion){
    std::ofstream
output("/home/turtlebot/catkin_ws/src/mie443_contest3/CNN_result.txt");
    std::string victim_emotion, response_type, robot_emotion;
    for (int i = 0; i < storeEmotion.size(); ++i){
        if (storeEmotion[i] == 0){
            victim_emotion = "Angry";
            response_type = "secondary";
            robot_emotion = "Rage";
        }
        else if (storeEmotion[i] == 1){
            victim_emotion = "Disgust";
            response_type = "primary";
            robot_emotion = "Sad";
        }
        else if (storeEmotion[i] == 2){
            victim_emotion = "Fear";
            response_type = "secondary";
            robot_emotion = "Positively Excited";
        }
        else if (storeEmotion[i] == 3){
            victim_emotion = "Happy";
            response_type = "primary";
            robot_emotion = "Pride";
        }
        else if (storeEmotion[i] == 4){
            victim_emotion = "Sad";
            response_type = "secondary";
            robot_emotion = "Discontent";
        }
        else if (storeEmotion[i] == 5){
            victim_emotion = "Surprise";
            response_type = "secondary";
            robot_emotion = "Embarrassment";
        }
        else if (storeEmotion[i] == 6){
            victim_emotion = "Neutral";
            response_type = "primary";
            robot_emotion = "Surprised";
        }
        output << "Discovery Order #";
        output << i+1;
        output << ": ";
        output << "We found a victim with a CNN-determined emotion of: ";
        output << victim_emotion;
        output << ". ";
        output << "So the robot displayed a ";
        output << response_type;
        output << " emotion of: ";
        output << robot_emotion;
        output << ".\n";
    }
    output.close();
}



int main(int argc, char** argv) {
    //
    // Setup ROS.
    ros::init(argc, argv, "contest3");
    ros::NodeHandle n;

    //ros::Subscriber emotion_sub = n.subscribe("/detected_emotion", 1,
&emotionCallback);
    ros::Subscriber emotion_sub = n.subscribe("/emotionOut", 1,
&emotionCallback);
    ros::Subscriber frontier_sub = n.subscribe("/emptyFrontier", 10,
&frontierCallback);
    //
    ros::Publisher vel_pub =
n.advertise<geometry_msgs::Twist>("cmd_vel_mux/input/teleop", 1);
    geometry_msgs::Twist vel;
    // Frontier exploration algorithm.
    explore::Explore explore;
    //
    // Class to handle sounds.
    sound_play::SoundClient sc;
    // Keep track of runtime (limit of 20 minutes total)
    std::chrono::time_point<std::chrono::system_clock> start;
    start = std::chrono::system_clock::now();
    uint64_t secondsElapsed = 0;

    ros::Duration(1).sleep();
    // Set the directory address to access the sound and image files to display
in robot emotion response
    std::string path_to_sounds = ros::package::getPath("mie443_contest3") +
"/sounds/";
    std::string path_to_images = ros::package::getPath("mie443_contest3") +
"/images/";
    //
    std::cout << "Start searching for victims!" << std::endl;

    while(ros::ok() && secondsElapsed <= 1200) {
        // Your code here.
        ros::spinOnce();

        // Start frontier exploration.
        explore.start();
        ros::Duration(0.01).sleep();
        // scan if frontier exploration unexpectedly stops in the beginning
        if (frontierEmpty && storeEmotion.size() == 0){
            std::cout << "I've only found " << storeEmotion.size() << " victims
so far. Must continue" << std::endl;
            explore.stop();
            unstuckFrontier(vel_pub, vel);
            std::cout << "Start moving again!!!!!!" << std::endl;
            frontierEmpty = false;
        }
        else if (frontierEmpty && storeEmotion.size() == 7){
            std::cout << "Finished finding all 7 victims! We can stop exploring 
now!" << std::endl;
            explore.stop();
            break;
        }

        if (detectedEmotion >= 0 && detectedEmotion <= 6){
            // Subscriber receives message from emotionClassifier.py for a new 
victim! Stop and respond!
            explore.stop();
            std::cout << "The CNN detected victim emotion is: " << 
detectedEmotion << std::endl;
            // Do robot response according to the detected emotion - show image, 
sound, and robot movements (3 modes of response)
            robotResponse(detectedEmotion, vel_pub, vel, path_to_sounds, 
path_to_images, &sc);
            // Reset to negative and wait until another victim is found
            storeEmotion.push_back(detectedEmotion);
            outputResult(storeEmotion);
            detectedEmotion = -1;
            std::cout << "Time Elapsed: " << secondsElapsed << " seconds" << 
std::endl;
        }

        // Start frontier exploration.
        //explore.start();

        ros::Duration(0.01).sleep();

        secondsElapsed = 
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now() 
- start).count();
    }
    return 0;
}
```

## 9. Appendix C - Python CNN Training Code

```python
#!/usr/bin/env python3
import torch
import torch.nn as nn
import argparse
from tqdm.auto import tqdm
import matplotlib.pyplot as plt

#
# Parse the input arguments.
def getInputArgs():
    parser = argparse.ArgumentParser('Sample for training an emotion
classification model.')
    parser.add_argument('-f')
    parser.add_argument('--gpu', dest='gpu', default=torch.cuda.is_available(),
type=bool, help='Use gpu for training')
    parser.add_argument('--nepoch', dest='nepoch', default=50, type=int,
help='Number of training epochs')
    parser.add_argument('--mdl', dest='mdl', default=None, type=str, help='Model
to load')
    parser.add_argument('--val', dest='val', action='store_true', help='Get
validation score')
    args = parser.parse_args()
    return args

class EmotionClassificationNet(nn.Module):

   def __init__(self):
       super(EmotionClassificationNet, self).__init__()
       self.cnn = nn.Sequential(
           nn.Conv2d(1, 128, 3, padding=1),
           nn.ReLU(),
           nn.BatchNorm2d(128),
           nn.MaxPool2d(2, 2),
           nn.Dropout(0.25),
            nn.Conv2d(128, 128, 3, padding=1),
           nn.ReLU(),
           nn.BatchNorm2d(128),
           nn.MaxPool2d(2, 2),
           nn.Dropout(0.25),

           nn.Conv2d(128, 128, 3, padding=1),
           nn.ReLU(),
           nn.BatchNorm2d(128),
           nn.MaxPool2d(2, 2),
           nn.Dropout(0.25),

           nn.Conv2d(128, 128, 3, padding=1),
           nn.ReLU(),
           nn.BatchNorm2d(128),
           nn.MaxPool2d(2, 2),
           nn.Dropout(0.25),

           nn.Conv2d(128, 128, 3, padding=1),
           nn.ReLU(),
           nn.BatchNorm2d(128),
           nn.Dropout(0.25),
       )

       self.nn = nn.Sequential(
           nn.Linear(1152, 200),
           nn.ReLU(),
           nn.Dropout(0.25),
           nn.Linear(200, 500),
           nn.ReLU(),
           nn.Dropout(0.25),
           nn.Linear(500, 7)

       )

   def forward(self, x, test_mode=False):
       batch_size = x.shape[0]
       feats = self.cnn(x)
       out = self.nn(feats.view(batch_size, -1))
       #
       # If we are testing then return prediction index.
       if test_mode:
            _, out = torch.max(out, 1)
        return out

def getDataset(args):
   import pathlib
   #if pathlib.Path('./train_split.pth').exists():
   if pathlib.Path('/content/drive/MyDrive/FOURTH YEAR MIE /Winter Semester
/MIE443 /MIE443 Project /Contest
3/mie443_contest3/src/train_split.pth').exists():
       #train_imgs, train_labels = torch.load('train_split.pth')
       train_imgs, train_labels = torch.load('/content/drive/MyDrive/FOURTH YEAR
MIE /Winter Semester /MIE443 /MIE443 Project /Contest
3/mie443_contest3/src/train_split.pth')
       probs = torch.ones(train_imgs.shape[0]) * 0.1
       val_set_mask = torch.bernoulli(probs).bool()
       val_imgs = train_imgs[val_set_mask]
       val_labels = train_labels[val_set_mask]
       train_imgs = train_imgs[~val_set_mask]
       train_labels = train_labels[~val_set_mask]
       return (train_imgs, train_labels), (val_imgs, val_labels)
   else:
       print('The provided dataset does not exist!')
       exit(0)

def getDataloader(args):
    train, val = getDataset(args)
    train_dataset = torch.utils.data.TensorDataset(*train)
    val_dataset = torch.utils.data.TensorDataset(*val)
    #
    # Due to class imbalance introduce a weighted random sampler to select rare
classes more often.
    batch_size = 700
    weights_label = train[1].unique(return_counts=True,
sorted=True)[1].float().reciprocal()
    weights = torch.zeros_like(train[1], dtype=torch.float)
    for idx, label in enumerate(train[1]):
        weights[idx] = weights_label[label]
    sampler = torch.utils.data.sampler.WeightedRandomSampler(weights,
len(weights))
```

```
    #
    # Create the dataloaders for the different datasets.
    train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=batch_size,
                                            num_workers=2, sampler=sampler)
    val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size,
                                            shuffle=False, num_workers=2)
    return train_loader, val_loader

def train_loop(mdl, loss_fn, optim, dl, device):
    print('')
    pbar = tqdm(dynamic_ncols=True, total=int(len(dl)))
    n_batch_loss = 400
    running_loss = 0
    for nex, ex in enumerate(dl):
        ims, labels, = ex
        ims = ims.to(device)
        labels = labels.to(device)
        #
        # Optimization.
        optim.zero_grad()
        outs = mdl(ims)
        loss = loss_fn(outs, labels)
        loss.backward(loss)
        optim.step()
        #
        # Statistics
        running_loss +=  loss.item()
        nex += 1
        if nex % n_batch_loss == 0:
            status = 'L: %.4f '%(loss / n_batch_loss)
            running_loss = 0
            pbar.set_description(status)
        pbar.update(1)
    pbar.close()
    return mdl

def calc_acc(mdl, dl, dl_type, device):
    print('')
    with torch.no_grad():
        pbar = tqdm(dynamic_ncols=True, total=int(len(dl)))
        total = 0
        ncorrect = 0
        for nex, ex in enumerate(dl):
            ims, labels, = ex
            ims = ims.to(device)
            labels = labels.to(device)
            predicted = mdl(ims, True)
            total += labels.size(0)
            ncorrect += (predicted == labels).sum().item()
            status = '%s ACC: %.4f '%(dl_type, float(ncorrect) / total)
            pbar.set_description(status)
            pbar.update(1)
    pbar.close()
    return float(ncorrect) / total

if __name__ == "__main__":
    args = getInputArgs()
    train_dl, val_dl = getDataloader(args)
    mdl = EmotionClassificationNet()
    ce_loss = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(mdl.parameters())
    device = torch.device('cpu')
    #save vectors for train, val accuracy vs. epoch
    train_list= []
    val_list = []
    if args.gpu:
        device = torch.device('cuda:0')
    if args.mdl is not None:
        mdl.load_state_dict(torch.load(args.mdl))
    mdl = mdl.to(device)
    if args.val:
        print('Val ACC loop')
        mdl.train(False)
        val_acc = calc_acc(mdl, val_dl, 'val', device)
        print('VAL ACC: ', val_acc)
    else:
        #
        # Training loop.
        best_val = -float('inf')
        for epoch in range(args.nepoch):
            print('Epoch: ' + str(epoch))
            print('Train loop')
            mdl.train(True)
            mdl = train_loop(mdl, ce_loss, optimizer, train_dl, device)
            print('Train ACC loop')
            mdl.train(False)
            train_acc = calc_acc(mdl, train_dl, 'train', device)
            train_list.append(train_acc)
            print('Val ACC loop')
            val_acc = calc_acc(mdl, val_dl, 'val', device)
            val_list.append(val_acc)
            #
            # Early stopping.
            if val_acc > best_val:
                best_val = val_acc
                #torch.save(mdl.state_dict(), 'mdl_best.pth')
                torch.save(mdl.state_dict(), '/content/drive/MyDrive/FOURTH YEAR
MIE /Winter Semester /MIE443 /MIE443 Project /Contest
3/mie443_contest3/src/mdl_best.pth')

    plt.plot(range(args.nepoch), train_list, 'r--', label="Training Accuracy")
    plt.plot(range(args.nepoch), val_list, 'b--', label="Validation Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()
```