MIE 443 Mechatronics Systems

Contest 3 Report

Team 7

Name	Student Number
Amanat Dhaliwal	1003425377
Jianfei Pan	1003948115
Qiwei Zhao	1003579950
Yuhang Song	1002946510

1.0 Problem Definition and Objectives of Contest 3

The goal of contest 3 is to let the simulated Turtlebot explore and identify seven victims located randomly in an unknown disastrous environment and provide them with emergency evacuation announcements. As a result, the Turtebot should find all the victims by continuously exploring and mapping the unknown environment. After reaching the victims, the Turtlebot should identify their emotions and interact with them according to their emotional states. The frontier exploration and victim identification algorithms are provided. Thus, the team needs to design algorithms for the Turtlebot to identify and respond to the seven user emotions and perform unique interactions using primary and secondary emotions. Each robot emotion has to be unique and thus must not be duplicated among the primary and secondary emotions.

1.1 Project Requirements

The requirements for contest 3 are discussed below.

- The contest environment is formed by many rooms of random layout.
- The seven victims are distributed randomly in the environment and are stationary.
- The victims are represented as blue visual squares in the simulated environment and their existence does not impact the exploration process.
- The team is provided with a dataset of images of seven emotions, a sample convolutional neural network (CNN), a victim detector for identifying victims, and a frontier exploration algorithm for the Turtlebot to navigate and map out the unknown environment and reach the victims.
- Once reaching the vicinity of the victim's location, the victim detector will provide multiple different images of the same emotion to the CNN for classification.
- The team can choose the type of disasters.
- The simulated Turtlebot will start at random locations selected by the instructor.
- The emotions of victims found by the Turtlbot will be automatically written to the 'detectedVictim.txt' file.
- The overall accuracy of the emotion classification algorithm will be tested with unseen test dataset that will not be provided to the team in advance.

1.2 Constraints

There are some constraints that the Turtlebot must meet in order to complete the tasks successfully.

- The algorithms must be robust for the Turtlebot to complete missions in an unknown environment.
- The face images in the contest will be different from the package and thus the CNN algorithm must be robust to a new test dataset.
- The Turtlebot must have unique emotional responses to seven possible emotions of the victims to help convey information.
- The Turtlebot must inform all victims with the evacuation instructions for the chosen disaster.

- During interactions, the team must choose 3 unique primary and 3 unique secondary emotions and 1 additional unique primary or secondary emotion.
- The robot emotions must be selected from the given list of emotions.
- The team must implement original motion and sound for the emotion interaction with the victims.
- The Turtlebot must meet and interact with all victims within 20 minutes.
- The team must not use any additional software packages in crafting emotion classification methods.
- The team must not use the 'detectedVictim.txt' file for image recognition during the contest.
- The team must not modify the automatically generated log files.

2.0 Robot Design Strategy and Implementation

The team is simulating a fire disaster emergency situation, where there are seven victims located randomly in the simulated environment. In order for the Turtlebot to traverse the unknown environment and inform and provide all victims with evacuation instructions, the team's design strategies are categorized into four aspects: mapping and localization, frontier exploration, emotion classification, robot interaction.

2.1 Mapping and Localization

In order to reach the victims, the Turtlebot needs to traverse the environment without bumping into obstacles or getting stuck. However, the first and foremost is the ability to map the environment and localize itself in the global coordinates in the unknown disaster environment. The move_base and GMapping algorithms will be used to achieve simultaneous mapping and localization. The team's strategy is to use the move_base package provided in ROS that can implement the action goal in the global coordinate and thus navigate the robot to the desired location via commanding the mobile base. Additionally, the team's strategy for mapping and localization is to use the GMapping package that utilizes the laserscan and odometry to achieve simultaneous localization and mapping. Thus, the Turtlebot can acquire abundant knowledge about the environment and then be commanded to the desired locations in the quest of continuous exploration for victims.

2.2 Frontier Exploration

In addition to mapping and localization, the Turtlebot needs to continuously explore the unknown environment to facilitate mapping and search for victims. Therefore, the team's strategy is to use the frontier exploration algorithm to achieve the point to point exploration. With the frontier exploration algorithm, the Turtlebot can use the current mapped frontier and send instructions to move_base to continuously explore the unmapped areas. However, since the algorithm sends asynchronous commands, it needs to be stopped and restarted between interactions with the victims. The team also

plans to add a failsafe random obstacle avoidance algorithm in case the frontier exploration algorithm unexpectedly stops functioning.

2.3 Emotion Classification

One important goal of the contest is to identify the emotions of the victims in order to provide appropriate responses. In order to recognize the emotions accurately, the team plans to use a well-designed convolutional neural network (CNN) with fully connected layers at the end as the emotion classifier. The team plans to train our CNN on the provided dataset and seven distinct emotions by tuning various hyperparameters. In order to ensure good generalizability and reduce overfitting, the team plans to use data augmentation, dropout, weight decay, k-fold, and batch normalization techniques during training. The team will store the pre-trained CNN model for image classification. When the robot reaches the victim and receives victim images, the team plans to use an ensemble method to classify a batch of images of the same emotion at once and select the most often occurring result, which increases the reliability of our emotion classification algorithm.

2.4 Robot Interaction

After identifying the victims' emotions, we need to provide multi-modal responses to the victims. We are required to choose unique primary and secondary emotions from the provided options. The team has selected robot emotions corresponding to different victim emotions and summarized them in Table 1. The team plans to include messages, movements, audio, and visual responses for each emotion during the robot interaction.

Victim Emotions	**Robot Emotions**	**Emotion Type**
Anger	**Fear**	Primary Emotion
Neutral	**Discontent**	Secondary Emotion
Fear	**Positively Excited**	Secondary Emotion
Sadness	**Embarrassment**	Secondary Emotion
Happiness	**Disgusted**	Secondary Emotion
Surprise	**Surprise**	Primary Emotion
Disgust	**Resentment**	Primary Emotion

Table 1. Summary of the Robot Interaction Emotions

The team's strategies for our robot's emotional responses to different victim emotions are that the Turtlebot should provide audio, literal, graphic and motion response of the

correct emotion corresponding to the victim's emotion. Our reasons for designing each pair of emotion interactions are summarized below.

2.4.1 Anger - Fear

For the victim that displays anger, in response the robot will display fear. In this case, the victim is assumed to be unaware of the emergency and the anger is assumed to be directed towards the robot for disturbing them. In response, the robot will display fear as a natural reaction, since the angry victim can potentially harm the robot. Hence, it is a primary emotion.

2.4.2 Neutral - Discontent

The victim that remains neutral is assumed that they are unaware of the emergency situation and are going about their daily work as normal. Hence, the robot feels discontent because it does not wish to be the bearer of bad news, and ruin the victims day. Which means that in this scenario discontent is a secondary emotion because the displayed emotion involves thinking and planning.

2.4.3 Fear - Positively Excited

In the case that the victim is in fear due to the fact that the building is on fire, the robot will display positive excitement to calm them down and let them know that they will be fine. Being positive can invoke a feeling of hope for the victim, remove their fear and calm them down. Hence, in this case the robot is displaying a secondary emotion because the robot deliberately shows this emotion.

2.4.4 Sadness - Embarrassment

In this case it is assumed that the victim is unaware of the emergency situation and is sad before the robot enters the room. In response, the robot will display embarrassment as its emotion. This is because the robot feels embarrassed about the situation where he has to give bad news to someone who is already very sad. Hence, in this scenario embarrassment would be a secondary emotion, because the robot feels inappropriate to worsen the victim's emotion.

2.4.5 Happiness - Disgusted

In the case where the victim displays happiness as its emotion, it is assumed that they are unaware of the emergency situation. So in response, the robot will display disgust as its emotion, since it feels disgusted about ruining the victim's happiness with the unfortunate emergency situation. The robot feels some responsibility and guilt for doing this, even though it hates spoiling others' happiness. Hence, disgust can be classified as a secondary emotion in this case

2.4.6 Surprise - Surprise

In the case where the victim is showing a surprised emotion, it is assumed that they are very surprised to see the robot and almost jump out of their chair. In response, the robot will also display surprise as its emotion. This is because the robot was also startled by their quick reaction and was surprised itself. Hence, in this situation it can be characterized as a primary emotion because it is a purely reactive emotion.

2.4.7 Disgust - Resentment

Lastly, in the case where the victim displays disgust as its emotion, it is assumed that the reaction is in response to the robot and the victim is unaware of the situation. So, in response the robot will feel resentment towards the victim for being disgusted by the robot. So in this case, resentment will be classified as a primary emotion of the robot, because the victim feels disgusted seeing the robot for no apparent reason.

3.1 Sensory Design

3.1.1 Bumpers Design

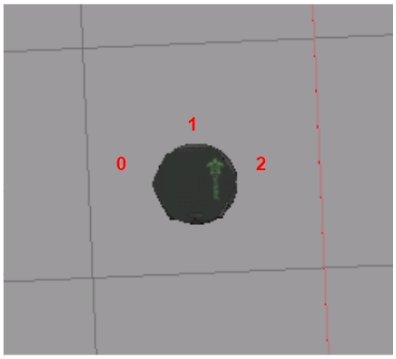


Figure 1. Position of Bumpers

There are three bumper sensors located on the left, front center, and right of the simulated turtlebot that are labeled as 0,1 and 2 respectively in Figure 1. When any of the bumpers are depressed, the bumper's state is 1. The bumper's state is 0 otherwise.

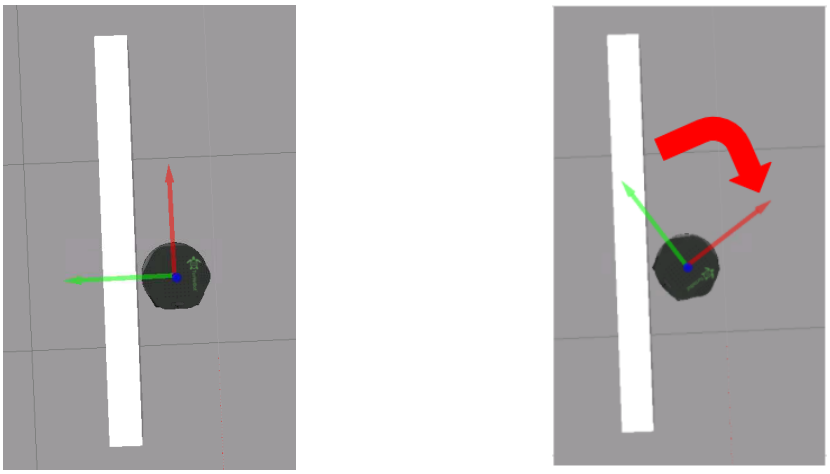


Figure 2. Turning Reaction of Bumper

The use of the bumpers is to detect collision with obstacles so that the Turtlebot can turn away from the obstacle. Therefore, the front bumper might be pressed in some unexpected circumstances. The bumpers on the right and left sides of the Turtlebot are to detect the obstacle at the blind spot of the laser sensor. Figure 2 shows the strategy that the team implements. When the left bumper is pressed, the Turtlebot will receive the feedback from the bumpers and then turn right away from the obstacle. In this way, it minimizes the error in the mapping process and helps Turtlebot escape from obstacles. Bumper is implemented in the obstacle avoidance algorithm and ensures the Turtlebot continues exploring the environment accurately after bumping into obstacles.

3.1.2 Odometry Design

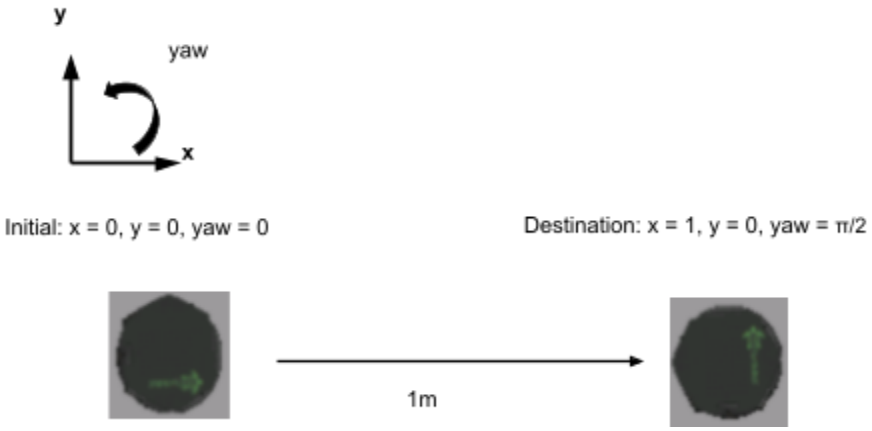


Figure 3. Change of Coordinate and Orientation of The Turtlebot

The use of odometry is to pinpoint the location of the turtlebot in the maze. The initial position of the turtlebot is x=0, y = 0, and yaw =0. The x and y position is measured in meters, and the yaw indicates the orientation of the Turtlebot in radians. Figure 3 shows an example of the Turtlebot's movement. The Turtlebot moves forward and turns 90 degrees to the left. The wheel encoder is an input to odometry as it counts the number of rotations of the motors and thus calculates the distance that the Turtlebot has driven

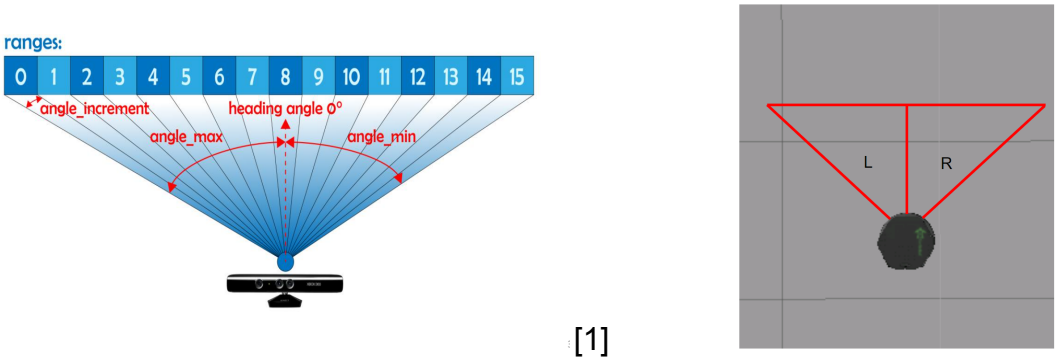
or adjusted. Moreover, the gyroscope is also an input to the odometry because it measures the orientation and angular velocity of the Turtlebot. Therefore, overall, the odometry detects the movement of the Turtlebot and measures the values in meters and radians respectively.

The implementation of odometry is to constantly report the location of the Turtlebot after each step. The sensory data from the odometry is used to assist mapping and localization, obstacle avoidance, frontier exploration, robot interactions with victims. The GMapping package will be used in the contest and it requires the pose data from the odometry together with the laser data from the depth sensor to achieve simultaneous localization and mapping.

Moreover, when the Turtlebot detects the obstacles, it needs the odometry to track the orientation and thus calculate the direction and amount of turning. In contest 3, frontier exploration sometimes stops functioning. Then, odometry is also used to determine if the robot gets stuck. If the location of the robot does not update for a long time, then the obstacle avoidance algorithm will be activated to drive the turtlebot for certain distances and then reactivate the frontier exploration. The frontier exploration algorithm uses the odometer to calculate the target yaw and destination. Moreover, the frontier exploration algorithm outputs to the move_base package needs the location and pose data to achieve the local planner for global navigation tasks. The base-controller also needs odometry to achieve and maintain the desired yaw and angular velocity.

Lastly, during robot interactions, the team plans to include physical motions as robot responses. Therefore, the yaw and location data from the odometry can help us achieve motion responses such as spinning and traversing.

3.1.3 Microsoft Kinect 360 Depth Sensor Design



[1]

Figure 4. The Laser Array and Strategy of Using Laser Scan

The most important function that Turtlebot must have is vision. Microsoft Kinect 360 is the “eye” of the Turtlebot, as it allows the Turtlebot to detect the distance between the Turtlebot and the obstacles at the front. Specifically, Microsoft Kinect 360 provides a depth sensor and an RGB camera. The depth sensor is used in this contest as it provides us with information about the surrounding obstacles. The depth sensor is a

laser scanner that senses a range array of around 600 indexes from left to right. Each index has a measured laser distance. The range of the viewing angle is from -12 degrees to 12 degrees. Increasing the viewing angle makes the Turtlebot have a larger range of detection, but it will decrease the accuracy of the laser scan at the two ends of the index. To precisely measure the position of the obstacles, the Turtlebot will use the smallest laser distance from the indexes. Once the Turtlebot is closing to an obstacle, it will trigger the obstacle avoidance function.

The implementation of the depth sensor (laser scanner) in this contest is to achieve localization and mapping, obstacle avoidance, and frontier exploration. The GMapping package requires the laser data from the depth sensor in addition to data from the odometry to perform the laser-based simultaneous localization and mapping. Moreover, the depth sensor provides distance to the obstacles so that the turtlebot can estimate its location to the obstacles, generate local and global costmaps, and thus avoid obstacles safely by reactive responses.

The depth sensor is also used to control distance to obstacles and the speed, because the team adjusts the speeds as the robot reaches closer to obstacles while maintaining a safe distance. The other use of the laser scan is to divide the range array into left and right sections, as shown in Figure 4. If there is an obstacle close to the left side of the Turtlebot, the Turtlebot will turn slightly to the right and move forward. In that way, the Turtlebot is capable of going through a sinuous and narrow corridor, when the obstacle avoidance algorithm is activated. Lastly, the depth sensor provides the laser scanner input to the frontier exploration algorithm to clear space and mark obstacles to generate the cost map that keeps track of the exploration progress [2]. The frontier exploration method uses the GMapping results and the depth sensor data to get the map and costmap to output point to point movement goal for expanding the frontier of the mapped region. Overall, the laser scanner of the depth sensor is critical for the robot's navigation and exploration, as it provides the distance measurements and facilitates the GMapping and frontier exploration algorithms.

3.2 Controller Design

3.2.1 High Level Controller

Sense-Plan-Act Hybrid Deliberative Controller

In this contest, the team uses the sense-plan-act hybrid deliberative controller as our high level controller, in which both reactive and deliberative controllers are used and the Turtlebot follows a sense-plan-act control architecture under most circumstances. Since the environment is static and does not change over time, a mix of reactive control and deliberative planning was utilized. Reactive control is used in obstacle avoidance and mapping and so on, whereas deliberative control is applied in frontier exploration and robot interactions and so on. The sensing, planning and acting are implemented in parallel and synergy with each other during mapping and localization, frontier

exploration, emotion identification and robot interaction. These three components are summarized at a high level as follows.

1. Sense

The obstacle avoidance algorithm is activated when the robot stays stagnant for a long time. It is an example of reactive control that uses readings from the odometry, bumper, and laserscan to avoid obstacles and efficiently traverse in the environment in a similar manner to contest 1. Moreover, once the Turtlebot reaches the vicinity of the victims based on its localization confidence, the victim detector will provide the victim's emotion to it for identification. The robot senses the surrounding for deliberative planning as well. The Turtlebot's odometry takes in the readings from gyroscope and wheel encoder to calculate the robot's pose orientation and position. The Turtlebot retrieves the depth information of the surrounding environment via the laserscan. Additionally, with the pose position form the odometry and the depth information from the laserscan, the GMapping algorithm can achieve the simultaneous localization and mapping [3]. Then, using the same amount of sensors, the frontier exploration algorithm can be achieved for continuous exploration.

2. Plan

The deliberative planning architecture involves frontier exploration and robot interactions to victims' emotions. In the frontier exploration algorithm, the robot is able to use the current map of the environment to identify the boundary of the currently mapped area, so that it can plan the next destination to an unknown area. The plan of navigation goals will be output to the move_base to 'act' the instruction. The move_base node will then link the global and local planner to reach the destination labeled in the global coordinate system [4]. Additionally, since the frontier exploration algorithm can interrupt the interactions with victims. Another deliberative control is to enable the robot to stop exploration during robot interactions with victims and resume the process afterwards. Moreover, the Turtlebot's emotional response for interacting with the victims is also a planning process. The team has provided emotions corresponding to each distinct victim's emotion in our design strategy section. Then, the robot needs to plan and determine its emotion to display to the victim after identifying the victim's emotion via our CNN classifier. After the robot determines its emotion, the next step is to demonstrate the multi-modal responses to the victim.

3. Act

After sensing and planning, the Turtlebot will receive and execute the commands and thus achieve the goals. In the obstacle avoidance, the robot needs to respond to different obstacles ahead of it and perform velocity and yaw adjustments. Moreover, during exploration, the robot will receive the navigation goals from the frontier

exploration algorithm and then pass it on to the move_base. Then, the move_base will then output desired angular and linear velocity to the mobile base controller that will start the motion and attempt to reach the destination. As a result, the mobile base will act on commands to navigate and explore new areas safely. The actions in the obstacle avoidance are mainly reactive, whereas the motions for exploration are largely a result of deliberative planning. Moreover, after the robot finds the victim and determines its emotion for interaction, the Turtlebot will react with unique multi-modal responses for that emotion. These responses consist of a combination of robot movements, sounds, printed sentences in the terminal, and visual images, which need to be displayed or performed by the robot. In summary, reactive action is only based on sensory data, whereas the deliberative action is based on planning and other deliberative algorithms.

3.2.2 Low Level Controller and Algorithms

1. Move_base and GMapping Algorithm

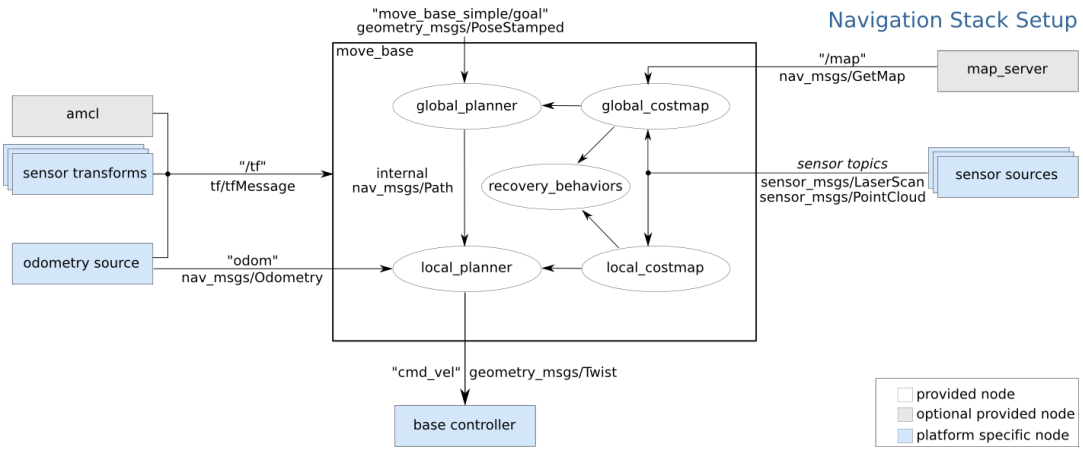


Figure 5. Application of Odometry and LaserScan in Navigation [4]

The combination of the Move_base and Gmapping is to ensure the rover can continuously move to the designated position and track its position in the environment. In order to do this, GMapping uses the information generated from laser sensor and odometry history to calculate the possible location of the rover, and then it will update the scanned environment and the rover's location in the map [1][5]. The frontier exploration will take in the map and the location information of the rover in the environment and then output the movement goals. Then, we can navigate the rover to the target location by passing the goals to the move_base node. Figure 5 shows the architecture of the move_base node, which indicates that it also needs the information from the odometry and the laser scan in order to send the angular and linear velocity goals to the base controller that moves the rover [4]. As a result, the rover can move to target location while mapping and localizing itself in the environment simultaneously [6].

inputs : Information from the laser scan and odometry
outputs : Series of command to localize and move the rover in the environment

2. Exploration Algorithm

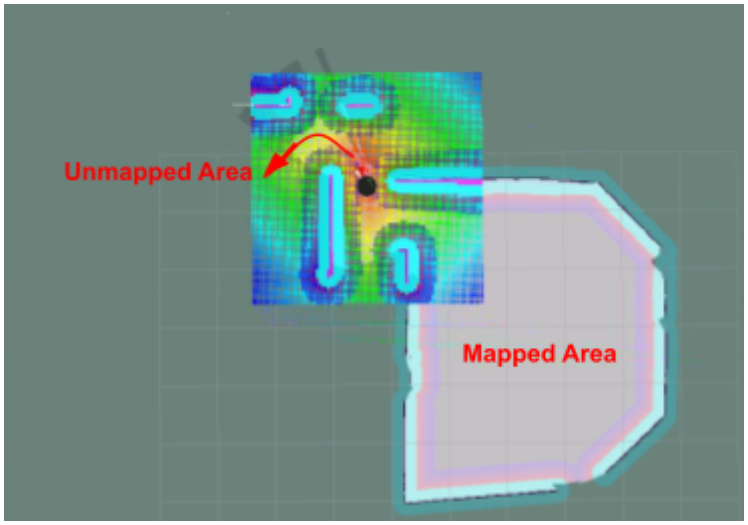


Figure 6. Costmaps of the Mapped and Unmapped Areas

Since the map and victim locations are unknown to the robot in advance, the robot needs to efficiently and continuously explore the environment to localize and find the victims. Thus, a frontier exploration algorithm is demanded. The frontiers are the boundaries between the mapped and unmapped region, as shown in Figure 6. The frontier exploration algorithm acts as an extension to the 2D costmap and move_base navigation stack [2]. The frontier exploration algorithm takes the current map and costmap generated from the GMapping and sends the new navigation goals for the Turtlebot to greedily explore the unmapped area. The navigation goals consist of the desired yaw and location information to move the rover towards the target location through the move_base package. Figure 7 shows the movement of the rover moving to the new frontiers after completing the exploration of the current room. This cycle repeats as the Turtlebot continues exploring the unknown environment until the 20 minutes limit. However, since the frontier exploration module sends commands asynchronously, in order not to interrupt the robot interaction, the team will stop the frontier exploration algorithm and resume it once the interaction is finished.

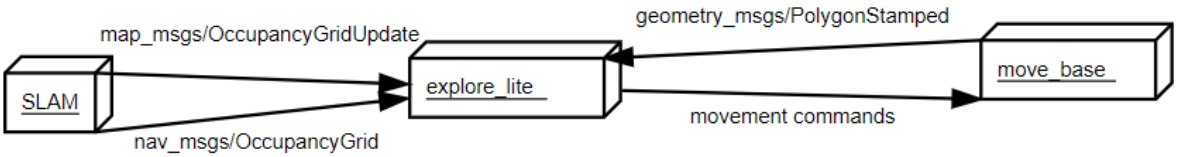


Figure 7. The inputs and outputs structure of the explore_lite library. [7]

Input: Explore_lite takes the inputs from the Simultaneous Localization and Mapping to knows the map around the rover
Output: Explore_lite sets up the target location in the unmapped area and sends the movement commands to that location to the move_base.

3. VictimLocator Algorithm

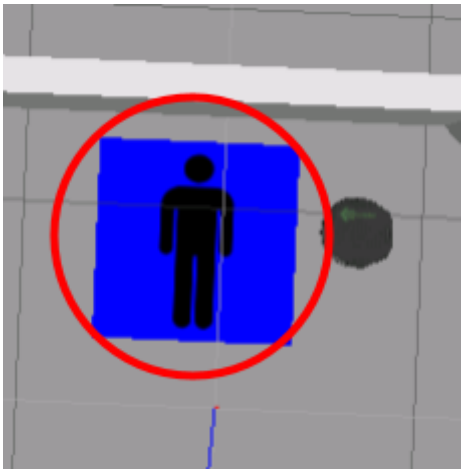


Figure 8. Illustration of a simulated victim

The VictimLocator algorithm enables victim localization and sends facial expression images of the found victim's emotion to the emotion classifier algorithm, while the rover is within the range of detection radius as shown in the red circle in Figure 8. This algorithm firstly loads the victim's locations from the victim.csv file. Then it constantly compares the current global pose of the rover and the victims' location in the environment. Once the algorithm detects the rover's position is within the detect radius of a victim, it publishes a message of the corresponding emotion image to the emotion classifier to detect the face expression of the victim as shown in Figure 9.

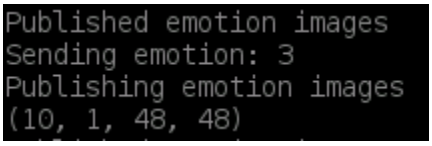


Figure 9. Output from the VictimLocator Algorithm

Input: The rover and victims' position in the environment
Output: The images message contains the victim's facial expression.

4. Stuck-Checking and Obstacle Avoidance

After implementing the exploration algorithm several times, the team noticed that the rover occasionally gets stuck and rotates indefinitely in some particular situations. As a result, the robot stagnated and failed to find all the victims within 20 minutes. It is observed that by conducting some random motions during the stuck period will reactivate the exploration code. Therefore, the first step is to check if the robot stops moving. The team implemented a "stuck-check" algorithm that constantly checks the motion of the robot. Specifically, the algorithm takes advantage of the sensory feedback from the laserscan and the odometer. The measured yaw and laser distance are stored in two separate lists. When the size of the list reaches 50, the algorithm will calculate the average difference between each measurement. If the average is small which means the difference between the previous 50 measurements is small, then the robot likely will stop not moving. Comparing both the yaw and laser distance measurement ensures the accuracy of the stuck-checking algorithm.

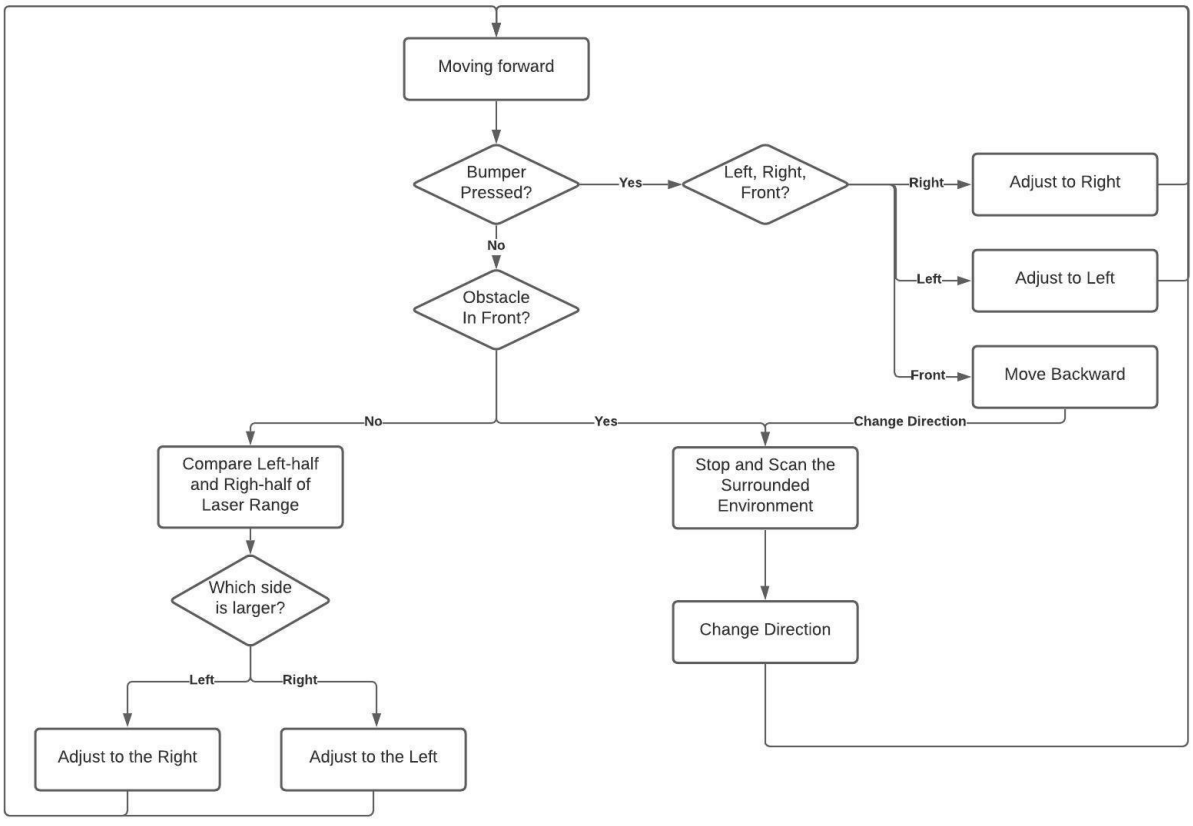


Figure 10. Flowchart of the Obstacle Avoidance

When the stuck condition is confirmed, the provided frontier exploration algorithm will be temporarily stopped and the robot will enter a random exploration mode adapted from our team's contest1 code shown in Figure 10. Since this algorithm is fairly complicated and explained in detail in contest 1, only an overview of major controllers in this algorithm will be provided in this section. The obstacle avoidance mode utilizes the depth data from the laser scanner to determine its proximity to obstacles and then react to avoid it. It also uses the bumper to detect potential collisions. Once collisions have been identified, the robot will turn and move away from the obstacles. Moreover, the rover uses the odometry to continuously track its pose and movement to ensure it can explore the new open area in the environment. This continuous exploration is achieved via noting the previously traversed locations so that the Turtlebot will not turn back unless it's necessary. The random obstacle avoidance will be activated for 7 seconds and then the robot will resume the frontier exploration algorithm.

Input: Measured yaws and laser distances
Output: None or entering the random exploration mode

5. CNN Emotion Classifier Architecture

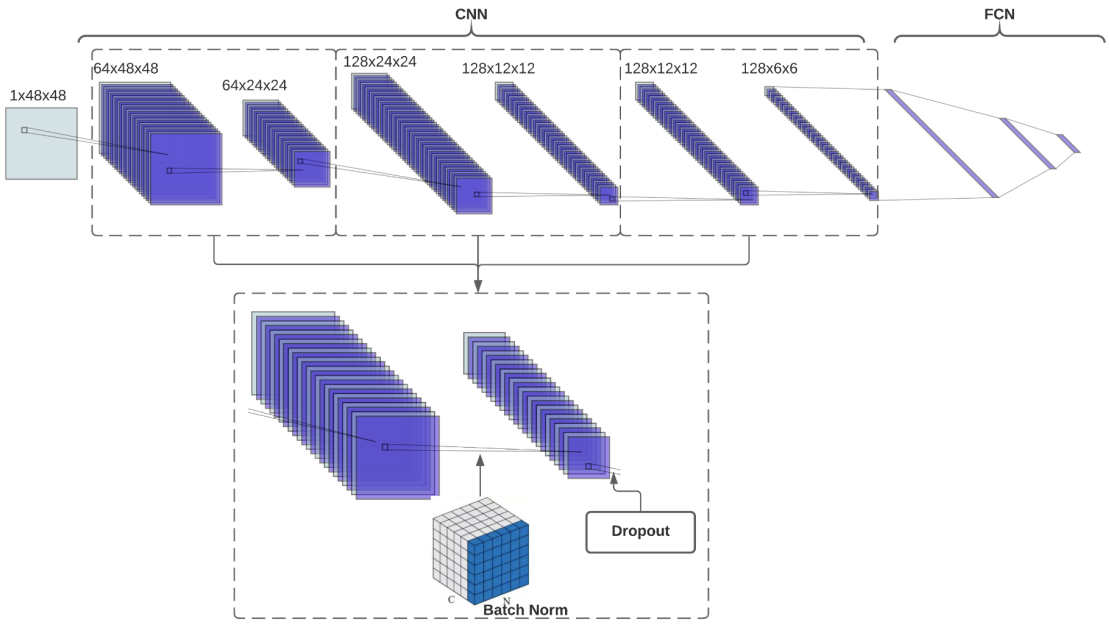


Figure 11. Illustration of the team's CNN and ANN Architecture

A convolution neural network with fully connected artificial neural networks is used to classify the emotions displayed by the victims. Figure 11 shows the overall structure of the neural network for emotion classification, where the input images are fed into a CNN

feature extractor and then a classifier. CNN is more powerful than ANN and RNN, with its ability to produce high accuracy in image recognition problems [8].The current neural network has 5 convolutional layers and 3 fully connected layers. After convolutional layers, the tensors are resized into a 1D vector, which then passes through 3 more fully connected layers. Finally, the last layer of the network outputs 7 different nodes, with each value representing an emotion, including sad, angry, disgust, fear, happy, neutral, and surprise. These values can be converted to probability by applying a softmax activation function at the end. The highest probability indicates predicted emotion from the input image.

A 3x3 kernel is applied at all convolutional layers to extract key feature information from each image to form the feature maps. In our network the team utilised max pooling, which takes the max value within a 2x2 matrix and creates a single pixel after each CNN layer to reduce the size of feature maps while concentrating on important features. The pooling layers make the image detection much less reliant on specific feature positions and reduce the computational complexity as the network gets deeper, making the classifier more efficient. The overall architecture also also uses a feature known as dropout at each hidden layer in the network. Dropout eliminates a certain percentage of nodes within the overall network in the hopes of eliminating overfitting and making the image detection much more robust to overfitting. When training a model with a certain dataset, the overall network adapts according to the dataset making it very accurate when testing with the same dataset. But it also makes it less flexible in identifying images that are not part of that dataset. Hence, dropout is used to reduce the effects of overfitting the model to just the training dataset.

Input: Tensor images of the victim's emotion
Output: The identified type of emotion

6. Training Techniques and Methods for Reducing the Overfitting Problem

The team has tuned multiple hyperparameters and used various training techniques in improving the validation accuracy and reducing the overfitting problem. For example, the team used dropout, weight decay, batch normalization and data augmentation in reducing the overfitting problem.

Firstly, dropout is the process of randomly removing a portion of nodes during training. We added the dropout layers after max pooling layer in the CNNs and after ReLu activation function in the fully connected layers. Dropouts can help prevent weights from overly dependent on each other and thus reduce the overfitting problem. nn.Dropout(0.25) means that we will randomly remove 25% neurons after the layer during training.

Secondly, the team used weight decay to penalize large weights and reduce overfitting. The weight decay of 0.00001 is input to the adam optimizer so that the prediction does not heavily rely on the content of one neuron, because now minimizing loss also incurs minimizing the weighted sum of weights.

Thirdly, the team adds batch normalization layers after each CNN layer to normalize the input layer and hidden layer. Doing so can increase the training efficiency by allowing each layer to be less affected by drastically changing input and to learn by itself more independently of other layers. Batching normalization ensures all features are scaled similarly and thus can help handle overfitting problems.

Lastly, since the team is given with a limited number of data, the team used many data augmentation techniques such horizontal flipping, random rotation, random cropping to obtain slightly but still valid data. In this way, we can expand our training dataset and thus reduce overfitting problems by training on more diverse dataset. We also used the k-fold cross validation, which refers to the way the dataset is split in order to verify the accuracy of the machine learning model [9]. In our case, the dataset was split into 4 folds. During training, each fold alternates to be the validation dataset with the rest three folds as the training dataset. This method allows a more accurate estimate of the model's performance on the validation dataset. Thus, the model's generalizability can be more accurately evaluated and the overfitting more effectively managed.

7. Emotion Classifier and the Ensemble Method

Index number	0	1	2	3	4	5	6
Emotion	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise

Table 2. Illustration of Victims Emotions with corresponding Indexes

Once the rover reaches the victim's detection radius, it would publish the emotion image from victimLocator.py to emotionClassifier.py. The emotionClassifier.py implements the CNN network mentioned previously to classify the emotion as the following: angry, sadness, disgust, fear, happiness, neutral, and surprise. The index number of each emotion is ranged from 0 to 6 respectively, as shown in Table 2. Nevertheless, Through multiple testing on our neural network, the validation accuracy is around 70%. It implies the rover cannot 100 % identify the victim's emotion correctly. Therefore to ensure the rover can detect the emotion accurately and display the correct response. We implement the ensemble in the code, which allows the same image to be processed ten times in the CNN and get ten different outputs. Then from those ten outputs, we select

the most frequently occurring emotion as the final victim's emotion. In that way, the accuracy of the emotion classification is significantly improved by utilizing the ensemble. The example of the emotion classifier is shown in Figure 12. The most occurring emotion is number 4, which corresponds to the emotion, neural, based on Table 2.

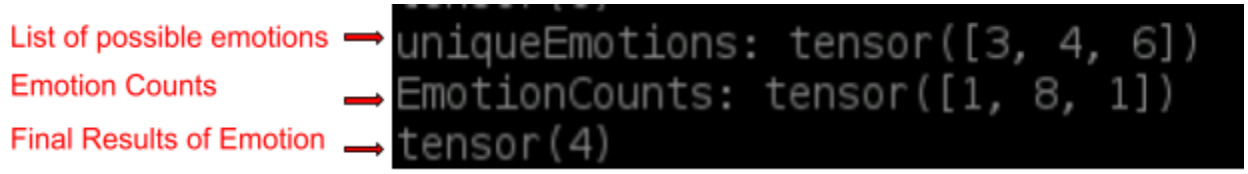


FIgure 12. The Outputs from the Emotion Classifier Algorithm

Input: Image of the victim's emotion
Output: Correctly classify the victim's emotion into 7 categories - angry, sadness, disgust, fearness, happiness, neutral, and surprise.

8. Emotion Responses and Interaction Algorithm

Once the detected_emotion message is updated, the robot will exert a corresponding response. To simulate an actual human emotional response, the team has implemented various types of responses including audio, image, motion responses and printed messages in the terminal. Details of each response are listed in Table 3 below.

Robot Emotions	Audio Response	Visual Response	Motion Response	Message Response
Fear	Don't be angry. I am scared of you.	A face of fear drawn by the team member.	Move backward quickly, then slowly move back to original position to imply the robot being afraid of the victim	Emergency! The building is on fire. You have 20 minutes to evacuate before the
Discontent	I am not happy with the situation.	A face of discontent drawn by the team member.	N/A	
Positively Excited	Hey don't be afraid. I am here to rescue you. Come	A positively excited face drawn by the team	Spin 360 degrees to cheer the victim up	

	with me.	member.		building is engulfed in fire. Please stay low and walk in a calm manner to the nearest fire exit and evacuate the building immediately.
Embarrassment	Uh… Sorry to interrupt. Could you come with me?	A face of embarrassment drawn by the team member.	N/A	
Disgusted	Ew… I am sorry to ruin your mood but I think we should go now, because there is a fire disaster.	A face of disgust drawn by the team member.	Simulate “Shake head” Motion by altering the angular velocity’s direction in each second	
Surprise	Hi, you surprise me. I have a surprise for you too. There is a disaster going on. Let’s leave. Hurry up.	A face of surprise drawn by the team member.	Move backward quickly to imply being surprised by the victim	
Resentment	Hi, why are you disgusted in me? I resent you. But for now I am here to save your life. Let’s go.	A face of resentment drawn by the team member.	Simulate a “Trembling ” Motion by constantly moving back and forth	

Table 3. Multimodal Responses for All Emotions

Input: Detected emotion from the emotion classifier
Output: Corresponding emotion response

4.0 Future Recommendation

The current program enables the turtlebot to successfully complete the mission within 20 minutes. But if the team had more time to improve the existing programme then we would focus on improving the efficiency of the emotion classification and robot

interaction algorithms. The future recommendations for each section are discussed as follows.

- Our current neural network architecture uses a classical CNN model for emotion classification. To improve the accuracy of training and validation, the team had tried various time-consuming and computationally expensive methods such as modifying the number of CNN layers, size of the kernel. In the future, the team can implement the technique of transfer learning which utilizes pre-trained neural networks such as VGG-16 and ResNet that have been extensively trained to solve image classification problems. The pre-trained neural network in the early layers can learn basic visual patterns of the images. The extracted features can then be fed into the team's own classifier model for final predictions. Using the pre-trained neural network will be expected to achieve a much higher training and validation accuracy than our current model, although the transfer learning is currently not allowed in this contest.
- To enrich the dataset, the team has implemented several augmentations to the original images including rotation, horizontal flip and random cropping. However, the fundamental augmentation technique will not be helpful for data distributions that are determined by higher-level features such as human emotions[10]. Therefore, in the future the team will use more augmentation techniques which are specialized to improve the variance of the human facial expressions. For example, X. Zhu et al. utilizes the Generative Adversarial Network (GAN) based data augmentation technique which can generate new human faces from the existing dataset. The data can then be used to complement the imbalance dataset which ultimately improves the accuracy of the model.
- Our current CNN model only predicts the emotion of the victim. In the future, the team can implement an additional intensity estimation on the predicted emotion. The robot can provide a different level of response according to the intensity of the emotion.
- The team has implemented various types of responses to the robot. However, current motion responses are very limited in the distance travelled because the team wants to minimize the disturbance to the exploration algorithm. In the future, the team can try to increase the distance travelled in the emotion response while reducing its disturbance on the exploration algorithm. For example, in the "angry" emotion state, the turtlebot will move faster than normal and in a larger range to show its agitation. Moreover, as stated previously, the same response can also vary according to the intensity of the emotion.

5.0 Contribution Table

The contribution table is summarized below.

Section	Student Names			
	Eugene Song	Amanat Dhaliwal	Richard Zhao	Jianfei Pan
The problem definition/objective of the contest	ET	ET	ET	RD,CM,ET
Strategy	ET	RS,MR, RD	CM,RD,MR,RS	CM,RD,MR,RS
Detailed Robot Design and Implementation	RD,MR,TS,RS	RD,RS	RD,MR,TS,RS	CM,RD,MR,RS
Future Recommendation	RD,MR,TS,RS	ET	ET	MI,ET
Coding	RD,MR,TS,RS	RS,TS	CM,RD,MR,RS	
CNN Classifier Training		RS,TS,MI	RS,TS,MI	RD,MR,TS,RS
All	FP	FP	FP	FP

Fill in abbreviations for roles for each of the required content elements. You do not have to fill in every cell. The “all” row refers to the complete report and should indicate who was responsible for the final compilation and final read through of the completed document.

RS – research
RD – wrote first draft
MR – major revision
ET – edited for grammar and spelling
MI – minor revision
TS – test and debug the codes
FP – final read through of complete document for flow and consistency
CM – responsible for compiling the elements into the complete document
OR – other

References

[1] “weblogin idpz.” [Online]. Available: https://q.utoronto.ca/courses/198595/files/folder/Tutorials?preview=11980879. [Accessed: 16-Apr-2021]
[2] “frontier_exploration - ROS Wiki.” [Online]. Available: http://wiki.ros.org/frontier_exploration. [Accessed: 16-Apr-2021]
[3] “gmapping - ROS Wiki.” [Online]. Available: http://wiki.ros.org/gmapping. [Accessed: 16-Apr-2021]
[4] “move_base - ROS Wiki.” [Online]. Available: http://wiki.ros.org/move_base. [Accessed: 16-Apr-2021]
[5] “gmapping - ROS Wiki.” [Online]. Available: http://wiki.ros.org/gmapping#Nodes. [Accessed: 16-Apr-2021]
[6] “move_base - ROS Wiki.” [Online]. Available: http://wiki.ros.org/move_base#Action_Subscribed_Topics. [Accessed: 16-Apr-2021]
[7] “explore_lite - ROS Wiki.” [Online]. Available: http://wiki.ros.org/explore_lite. [Accessed: 16-Apr-2021]
[8] “Difference between ANN, CNN and RNN - GeeksforGeeks,” 28-Jun-2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-ann-cnn-and-rnn/. [Accessed: 16-Apr-2021]
[9] “pytorch-K-fold cross-validation process description and implementation - Programmer Sought.” [Online]. Available: https://www.programmersought.com/article/72484530160/. [Accessed: 16-Apr-2021]
[10] “[No title].” [Online]. Available: https://arxiv.org/pdf/1711.00648.pdf. [Accessed: 16-Apr-2021]

Appendix

Full Code:

Contest3.cpp

```
// New Stuck-solving code
#include <ros/ros.h>
#include <ros/package.h>
#include <geometry_msgs/Twist.h>
#include <kobuki_msgs/BumperEvent.h>
#include <sensor_msgs/LaserScan.h>


#include "explore.h"

#include <stdio.h>
#include <cmath>
#include <vector>
#include <chrono>
#include <algorithm>

#include<nav_msgs/Odometry.h>
#include<tf/transform_datatypes.h>
//
// 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 <std_msgs/Int32.h>
/////// add new library/////
#include <cv.h>

#include <opencv2/core/core.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/xfeatures2d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
```

```
#include <opencv2/highgui.hpp>
#include <iostream>

#include <cv.h>
#include <cv_bridge/cv_bridge.h>

using namespace cv;
////////////////////////////////
#define N_BUMPER (3)
#define RAD2DEG(rad) ((rad) *180./M_PI)
#define DEG2RAD(deg) ((deg) *M_PI /180.)
#define ANGULAR_SPEED (M_PI/3)
uint8_t
bumper[3]={kobuki_msgs::BumperEvent::RELEASED,kobuki_msgs::BumperEvent::RE
LEASED,kobuki_msgs::BumperEvent::RELEASED};

float minLaserDist=std::numeric_limits<float>::infinity();
float minDistL = 100.0;
float minDistR = 100.0;
float angleadjust = 0.0;
float lavg;
float ravg;
float turning_prob;
float sum_eugene;
float lasterror;
float lastavgerror;
float minDistabs = 0;
float prevtime;

int32_t image_number = 7;
uint8_t victimcount = 0;
std::vector<float> PosXList;
std::vector<float> PosYList;
std::vector<float> Laserlist;
std::vector<bool> TurnLeftList;
////// richard add////
std::vector<int> victimList;

int prev_image_number;
```

```
enum emotion{Angry,Disgust,Fear,Happy,Sad,Surprise,Neutral,none};



//////////////////////////////
int32_t nLasers=0,desiredNLasers=0,desiredAngle=12;

bool isleftbumperpressed = false;
bool right_bumper_pressed = false;
bool front_bumper_pressed = false;
bool left_bumper_pressed = false;
bool turn360requested = false;
bool turndesired = false;
bool turn = false;
bool maxreach = false;
bool turndesiredold = false;
bool pause3 = false;
bool Newvictim = false;
bool tryblacklist = false;

int counting_backward_step = 0;
int countstps = 0;
const int stpsize = 3;

float angular =0.0;
float linear =0.0;
float distinit = 0.0, distprev = 0.0;
float posX=0.0, posY=0.0, yaw =0.0, yawinit = 0, yawdiff = 0, yawprev = 0;
float maxdist = 0.0;
float LRerror = 0.0;
float changeangle = 0.0;
float currtime = 0;
float LRavgerror = 0.0;
int maxElementIndex = 0;
float maxElement = 0;

std::random_device rd;
std::mt19937 mt(rd());
std::bernoulli_distribution dist(0.55); // this is the chance of getting
true, between 0 and 1;
```

```
std::vector< float > distmax;
std::vector< float > distmaxYaw;
std::vector<float> preferred_dir;
std::vector<float> preferred_dist;
std::vector<float> Yawlist;
bool turnleft = false;

int     Checksimilarity(std::vector<float>     &PosXList,std::vector<float>
&PosYList, float &posX, float&posY){
   std::vector<int> simlist;
   for(uint32_t i = 0;i<(PosXList.size());++i) {
        if (fabs(posX-PosXList[i]) < 0.2 && fabs(posY-PosYList[i]) < 0.2){
            simlist.push_back(i);
        }
   }
   if(simlist.size()>0){
       return simlist[simlist.size()-1];
   }
   return PosXList.size();
}

bool      checkstuck(std::vector<float>      &Laserlist,std::vector<float>
&Yawlist){
   float dist_errsum = 0;
   float dist_erravg = 0;
   float yaw_errsum = 0;
   float yaw_erravg = 0;

   for(uint32_t i = 1;i<(Laserlist.size());++i) {
       if(Laserlist[i]> 50){
           return true;
       }
       dist_errsum += fabs(Laserlist[i]-Laserlist[i-1]);
       yaw_errsum += fabs(Yawlist[i]-Yawlist[i-1]);
   }
   dist_erravg = dist_errsum/(Laserlist.size()-1);
   yaw_erravg = yaw_errsum/(Yawlist.size()-1);
   //ROS_INFO("percentage: %f,%f", dist_erravg, yaw_erravg);
   if ((dist_erravg<0.001||dist_erravg > 100) && yaw_erravg <0.001){
      return true ;
```

```
    }
    return false;
}

void bumperCallback(const kobuki_msgs::BumperEvent::ConstPtr& msg)
{

    bumper[msg->bumper] =msg->state;

}

int avoidgoingbackward(std::vector<float> &preferred_dir, float &yawinit){
    ROS_INFO("Checking if going backward");
    for (uint32_t i=0; i<preferred_dir.size(); i++){
          if(fabs(fabs(preferred_dir[i]-yawinit)-M_PI)<0.80){   // check if
its the backward direction
            return i;
        }
    }
    return preferred_dir.size();
}

void laserCallback(const sensor_msgs::LaserScan::ConstPtr& msg)
{
        //fill with your code
    minLaserDist=std::numeric_limits<float>::infinity();
    minDistR = 100;
    minDistL = 100;
    lavg = 0.0;
    ravg = 0.0;
    nLasers=(msg->angle_max-msg->angle_min) /msg->angle_increment;
    desiredNLasers=DEG2RAD(desiredAngle)/msg->angle_increment;
    sum_eugene = 0.001;
    int i = 0;
    //ROS_INFO("Size of laser scan array: %i and anglemin: %f and anglemax:
%f", nLasers, msg->angle_max,msg->angle_min);


if(desiredAngle*M_PI/180<msg->angle_max&&-desiredAngle*M_PI/180>msg->angle
_min) {
```

```
                                                                for(uint32_t
laser_idx=nLasers/2-desiredNLasers;laser_idx<nLasers/2+desiredNLasers;++la
ser_idx){
            minLaserDist=std::min(minLaserDist,msg->ranges[laser_idx]);
        }
    }
    else {
        for(uint32_t laser_idx=0;laser_idx<nLasers;++laser_idx) {
            minLaserDist=std::min(minLaserDist,msg->ranges[laser_idx]);
        }
    }
    for(uint32_t laser_idx=20;laser_idx<nLasers/2;++laser_idx){
            minDistL=std::min(minDistL,msg->ranges[laser_idx]);
            if (!std::isnan(msg->ranges[laser_idx])){
            sum_eugene += msg->ranges[laser_idx];
            i++;}
    }
    if(i>0){
    lavg = sum_eugene/(i);}
    sum_eugene = 0.0, i = 0;
    for(uint32_t laser_idx=nLasers/2;laser_idx<nLasers-20;++laser_idx) {
            minDistR=std::min(minDistR,msg->ranges[laser_idx]);
            if (!std::isnan(msg->ranges[laser_idx])){
            sum_eugene += msg->ranges[laser_idx];
            i++;}
    }
    if(i>0){
    ravg = sum_eugene/(i);}
    sum_eugene = 0.0, i = 0;
    LRerror = minDistL - minDistR;
    LRavgerror = lavg - ravg;
    minDistabs = msg -> ranges[nLasers/2];

    Laserlist.push_back(minLaserDist);

}

void odomCallback(const nav_msgs::Odometry::ConstPtr&msg)
{
    //fill code
```

```
   posX=msg->pose.pose.position.x;
   posY=msg->pose.pose.position.y;
   yaw=tf::getYaw(msg->pose.pose.orientation);
   tf::getYaw(msg->pose.pose.orientation);
   Yawlist.push_back(yaw);
    //ROS_INFO("Position: (%f,%f) Orientation:%frad or%fdegrees.", posX,
posY,yaw,RAD2DEG(yaw));
}

float distcal(float px1, float py1, float px2, float py2){
   return sqrt(pow(fabs(px2-px1),2) + pow(fabs(py2-py1),2));
}


void EmotionCallback(const std_msgs::Int32 intmsg){

       if(image_number != intmsg.data){
           ROS_INFO("Find a Victim");
           Newvictim = true;

       }
       image_number = intmsg.data;
   }

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 bumper_sub = n.subscribe("mobile_base/events/bumper",
10, &bumperCallback);
   ros::Subscriber laser_sub = n.subscribe("scan", 10, &laserCallback);
   ros::Subscriber odom=n.subscribe("odom", 1, &odomCallback);
                                ros::Publisher          vel_pub          =
n.advertise<geometry_msgs::Twist>("cmd_vel_mux/input/teleop", 1);

   ros::Rate loop_rate(10);
```

```
   geometry_msgs::Twist vel;


   float angular = 0.0;
   float linear = 0.0;
   int bumper_check = 0;

   //
   // Frontier exploration algorithm.
   explore::Explore explore;
   //
   // Class to handle sounds.
   sound_play::SoundClient sc;
   //
   // The code below shows how to play a sound.
   std::string path_to_sounds = ros::package::getPath("mie443_contest3") +
"/sounds/";
   ros::Duration(0.1).sleep();
   sc.playWave(path_to_sounds + "sound.wav");
   //
   // The code below shows how to start and stop frontier exploration.
   //own code for exploration at specific problem case

   ////////////////richard to set up the image ///////

    std::string path_2_imgs = ros::package::getPath("mie443_contest3") +
"/images/";

   Mat angry = imread(path_2_imgs + "fear.png", IMREAD_COLOR);
   Mat disgust = imread(path_2_imgs + "resentment.png", IMREAD_COLOR);
        Mat fear = imread(path_2_imgs + "positively_excited.png",
IMREAD_COLOR);
   Mat happy = imread(path_2_imgs + "disgusted.png", IMREAD_COLOR);
   Mat neutral = imread(path_2_imgs + "discontent.png", IMREAD_COLOR);
   Mat sad = imread(path_2_imgs + "embarrasment.png", IMREAD_COLOR);
   Mat surprise = imread(path_2_imgs + "surprise.png", IMREAD_COLOR);


///Spin 360 initially
```

```
   std::chrono::time_point<std::chrono::system_clock> start;
   start = std::chrono::system_clock::now();
   uint64_t secondsElapsed2 = 0;
   while(ros::ok() && secondsElapsed2<10){
       ros::spinOnce();
       vel.angular.z = M_PI/3;
       vel.linear.x =  0;
       vel_pub.publish(vel);
                                                  secondsElapsed2       =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock
::now()-start).count();
       loop_rate.sleep();
   }

   explore.start();
   start = std::chrono::system_clock::now();
   secondsElapsed2 =0;
   ROS_INFO("Searching the victim....");
   while(ros::ok() && secondsElapsed2 < 1200) {
       // Your code here.
       ros::spinOnce();

        if (Newvictim){

           bool any_bumper_pressed=false;
           for(uint32_t b_idx=0; b_idx<N_BUMPER; ++b_idx)
           {
                                       any_bumper_pressed|=(bumper[b_idx]
==kobuki_msgs::BumperEvent::PRESSED);
           }
           float settime = 6;
           explore.stop();
           std::chrono::time_point<std::chrono::system_clock> start;
           start = std::chrono::system_clock::now();
           uint64_t secondsElapsed = 0;
           linear = 0.3;
           angular = M_PI/3;
           while(ros::ok() && secondsElapsed < settime){
               ros::spinOnce();
```

```
        switch (image_number){  //

            case Angry://scared and step back
             if (secondsElapsed ==2){
                ROS_WARN("Victim Emotion Identified to be Angry");
                cv::imshow("Display window", angry);
                waitKey(5);
                    ros::Duration(0.1).sleep();
                sc.playWave(path_to_sounds + "fear_angry.wav");}

                settime = 7;
                angular = 0;
                if(!any_bumper_pressed){
                if(secondsElapsed<1){
                linear = -0.3;}else{
                    if (secondsElapsed%2!=0){
                        linear = 0;
                    }else{
                        linear = 0.1;
                    }
                }}
                break;

            case Disgust: // shaking
                ROS_WARN("Victim Emotion Identified to be Disgust");
                if (secondsElapsed ==2){
                cv::imshow("Display window", angry);
                waitKey(5);
                    ros::Duration(0.1).sleep();
                sc.playWave(path_to_sounds + "fear_angry.wav");

                }
                settime = 6;
                angular = 0;
                linear *= -0.95;
                break;


            case Fear://shake head
                ROS_WARN("Victim Emotion Identified to be Fear");
```

```
                if (secondsElapsed ==2){
                cv::imshow("Display window", fear);
                waitKey(5);
                    ros::Duration(0.1).sleep();
                                        sc.playWave(path_to_sounds   +
"positivelyexcited_fear.wav");
                }

                linear = 0;
                settime  = 4;
                if(secondsElapsed <1 || secondsElapsed>2.9){
                    angular = -M_PI/6;
                }else {
                    angular = M_PI/6;
                }
                break;


            case Happy: // Shake head show disgust

                ROS_WARN("Victim Emotion Identified to be Happy");

                if (secondsElapsed ==2){
                cv::imshow("Display window", happy);
                waitKey(5);
                    ros::Duration(0.1).sleep();
                sc.playWave(path_to_sounds + "disgusted_happy.wav");
                }

                linear = 0;
                settime  = 4;
                if(secondsElapsed <1 || secondsElapsed>2.9){
                    angular = -M_PI/6;
                }else {
                    angular = M_PI/6;
                }
                break;
```

```
            case Sad: // Turn a circle to cheer victim up

                ROS_WARN("Victim Emotion Identified to be Sad");

                if (secondsElapsed ==2){
                cv::imshow("Display window", sad);
                waitKey(5);
                    ros::Duration(0.1).sleep();
                sc.playWave(path_to_sounds + "embarrassment_sad.wav");
                }

                settime = 6;
                linear = 0;
                break;


            case Surprise: // get suprised

                ROS_WARN("Victim Emotion Identified to be Surprise");

                if (secondsElapsed ==2){
                cv::imshow("Display window", surprise);
                waitKey(5);
                    ros::Duration(0.1).sleep();
                sc.playWave(path_to_sounds + "surprise_surprise.wav");
                }
                if(secondsElapsed<2){
                linear *= -1;
                angular *= -1;} else{
                linear = 0;
                angular = 0;
               }
                break;


            case Neutral:

                ROS_WARN("Victim Emotion Identified to be Neutral");

                if (secondsElapsed ==2){
```

```
                    cv::imshow("Display window", neutral);
                    waitKey(5);
                        ros::Duration(0.1).sleep();
                    sc.playWave(path_to_sounds + "discontent_neutral.wav");

                    }
                    angular = 0;
                    linear = 0;
                    break;


                case none:

                    ROS_WARN("Victim Emotion Identified to be None");


            }
                vel.angular.z = angular;
                vel.linear.x =  linear;
                vel_pub.publish(vel);
                                                        secondsElapsed    =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock
::now()-start).count();
                loop_rate.sleep();
            }
            //Richard's Reactions End

               ROS_FATAL("Emergency! The Building is on fire. You have 20
minutes to evacuate before the building is engulfed in fire. Please stay
low and walk in a calm manner to the nearest fire exit and evacuate the
building immediately");

            Newvictim = false;
            victimcount++;
            if (victimcount <7){
                ROS_INFO("Searching the Next Victim.......");
            } else{
                return -1;
            }
            goto endofop2;
```

```
       }

       if (Laserlist.size() > 100){



        if(checkstuck(Laserlist,Yawlist)){
           explore.clearblacklist();
           ROS_WARN("Frontier Exploration stucked, exerting back-up random
motions");
           explore.stop();

           std::chrono::time_point<std::chrono::system_clock> start_stuck;
           start_stuck = std::chrono::system_clock::now();
           uint64_t secondsElapsed = 0;

           float angular = 0.0;
           float linear = 0.0;
           int bumper_check = 0;
                                    ////////////////////////////code   from
contest1//////////////////////
           while(ros::ok() && secondsElapsed <= 15 && !Newvictim) {
               ros::spinOnce();
               bool any_bumper_pressed=false;
               for(uint32_t b_idx=0; b_idx<N_BUMPER; ++b_idx)
               {
                                          any_bumper_pressed|=(bumper[b_idx]
==kobuki_msgs::BumperEvent::PRESSED);
               }

               if(any_bumper_pressed){

                                           front_bumper_pressed|=(bumper[1]
==kobuki_msgs::BumperEvent::PRESSED);

                                            left_bumper_pressed|=(bumper[0]
==kobuki_msgs::BumperEvent::PRESSED);

                                           right_bumper_pressed|=(bumper[2]
==kobuki_msgs::BumperEvent::PRESSED);
```

```
                }
                                //  ROS_INFO("front:%i,  left:%i,right:%i",
front_bumper_pressed, left_bumper_pressed,right_bumper_pressed);
                if(PosXList.size()>40){
                    PosXList.clear();
                    PosYList.clear();
                    TurnLeftList.clear();
                }


                if (!turndesired && !turndesiredold){
                    turnleft = dist(mt);
                }

                if (turndesired && !any_bumper_pressed){
                   // ROS_INFO("Finding Optimal Direction...");
                      if (minLaserDist - distprev > 0.05 || minLaserDist <1
||minLaserDist>6.5){
                        distprev = minLaserDist;
                        linear = 0;
                        if (turnleft){
                            angular = M_PI/3;
                        }else if(!turnleft){
                            angular = - M_PI/3;
                        }
                     // ROS_INFO("Adjusting:%f, prev:%f , angle: current:%f,
init:%f", minLaserDist,distprev,yaw,yawinit);
                    }
                    else{
                        linear = 0;
                        angular = 0;
                        turndesired = false;
                        pause3 = true;
                       // ROS_INFO("Adjustment finished:%f", minLaserDist);
                    }

                    goto endofop;}


                if(!turn){
```

```
                    yawinit = yaw;
                }


                if (turndesiredold && !any_bumper_pressed){
                    //ROS_INFO("Finding Open Space");

                    if(turnleft){
                    while(yaw < yawprev && (fabs(yaw-yawinit) > 0.01)){
                        yaw += 2*M_PI;
                    }}else{

                    while(yaw > yawprev && (fabs(yaw-yawinit) > 0.01)){
                        yaw -= 2*M_PI;
                    }}

                    yawdiff = fabs(yaw - yawinit);
                    // ROS_INFO("Adjusting:%f, prev:%f , angle: current:%f,
init:%f", minLaserDist,distprev,yaw,yawinit);
                    if ((2*M_PI - yawdiff) > 0.1){
                       if (fabs(M_PI - yawdiff)<M_PI/6 ||  minLaserDist <1
||minLaserDist>6.5){
                        distprev = minLaserDist;
                        linear = 0;
                        if (turnleft){
                            angular = M_PI/3;
                        }else if(!turnleft){
                            angular = - M_PI/3;
                        }

                        yawprev = yaw;
                        currtime = secondsElapsed;
                    }
                    else{
                        linear = 0;
                        angular = 0;
                        turndesiredold = false;
                        pause3 = true;
                        yawprev = yaw;
                      //  ROS_INFO("Adjustment finished:%f", minLaserDist);
```

```
                    }}else{
                        linear = 0;
                        angular = 0;
                        yawprev = yaw;
                        turndesiredold = false;
                        distinit = minLaserDist;
                        distprev =distinit;
                        turndesired = true;
                    }
                    goto endofop;
                }


                if (front_bumper_pressed){ //bumped go backward
               // ROS_INFO("front_bumper");
                    linear = -0.3;
                    angular = M_PI/3;
                    counting_backward_step++;
                   //       ROS_INFO("anguar:%f linear:%f  MinLaserDist:%f
counting_backward:%i   BACK   BACK   ",   angular,   linear,minLaserDist,
counting_backward_step);


                    if (counting_backward_step==14){   // going backward
steps
                        counting_backward_step=0;
                        front_bumper_pressed = false;
                        turndesiredold = true;
                        distinit = minLaserDist;
                        distprev =distinit;

                        int check = 0;
                        if(PosXList.size()>0){
                                                                    check   =
Checksimilarity(PosXList,PosYList,posX,posY);
                            if(check < PosXList.size()){
                                turnleft = (turnleft == TurnLeftList[check]
? !turnleft : turnleft);
                               // ROS_INFO("Same Place");
                            }
```

```
                    }
                    PosXList.push_back(posX);
                    PosYList.push_back(posY);
                    TurnLeftList.push_back(turnleft);
                    goto endofop;
                }
            }



            else if (left_bumper_pressed){ // bumped turn right
          //  ROS_INFO("left_bumper");
                linear = -0.3;
                angular = -M_PI/3;
                counting_backward_step++;
             //          ROS_INFO("anguar:%f linear:%f  MinLaserDist:%f
counting_backward:%i        TURNNING        RIGHT        ",        angular,
linear,minLaserDist,counting_backward_step);
            //

                    if (counting_backward_step==10){   // going backward
steps
                    counting_backward_step=0;
                    left_bumper_pressed = false;

                }
            }

            else if (right_bumper_pressed){ //bumped turn left
                linear = -0.3;
                angular = M_PI/3;
                counting_backward_step++;
               //       ROS_INFO("anguar:%f linear:%f  MinLaserDist:%f
counting_backward:%i                TURNNING     left     ",     angular,
linear,minLaserDist,counting_backward_step);


                    if (counting_backward_step==10){   // going backward
steps
                    counting_backward_step=0;
```

```
                    right_bumper_pressed = false;

                }
            }

            else if(!any_bumper_pressed){  // go forward
              //  ROS_INFO("go forward");

                if (minLaserDist>0.55 && minLaserDist<6.5){

                    if (minDistabs  < 0.7){
                        linear  = 0.2;
                        angular =  -0.3*LRerror;

                    }else{
                    linear = 0.3;
                    angular = -0.255*LRerror;}

                    if(angular >M_PI/6 || angular< -M_PI/6){
                        angular = 0;
                    }

                    if(linear >0.2 ){
                        linear = 0.3;
                    }


                }
                   else if ((minLaserDist<0.55 || minLaserDist>6.5) &&
secondsElapsed >1){
                    turndesiredold = true;
                    //Try

                    int check = 0;
                    if(PosXList.size()>0){
                                                                check   =
Checksimilarity(PosXList,PosYList,posX,posY);
                        if(check < PosXList.size()){
                            turnleft = (turnleft == TurnLeftList[check]
? !turnleft : turnleft);
```

```
                        //    ROS_INFO("Same Place");
                        }
                    }
                    PosXList.push_back(posX);
                    PosYList.push_back(posY);
                    TurnLeftList.push_back(turnleft);
                     //

                    distinit = minLaserDist;
                    distprev =distinit;
                    lasterror = 0;
                    yawinit = yaw;
                    yawprev = yaw;

                }
            }




            endofop:
            vel.angular.z = angular;
            vel.linear.x = linear;
            vel_pub.publish(vel);
            if (angular !=0){
                turn =true;
            }else{
                turn = false;
            }
            // The last thing to do is to update the timer.
                                                    secondsElapsed    =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock
::now()-start_stuck).count();
            loop_rate.sleep();
        }

      ROS_INFO("Searching for Victims....");
      }

//////////////////////////end of contest1/////////////////////////
```

```
        Laserlist.clear();
        Yawlist.clear();
        }
        endofop2:
        explore.start();
        ros::Duration(0.1).sleep();
                                                    secondsElapsed2       =
std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock
::now()-start).count();
        loop_rate.sleep();
    }
    return 0;
}
```

Explore.cpp

```
/*********************************************************************
*
* Software License Agreement (BSD License)
*
*  Copyright (c) 2008, Robert Bosch LLC.
*  Copyright (c) 2015-2016, Jiri Horner.
*  All rights reserved.
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*   * Redistributions in binary form must reproduce the above
*     copyright notice, this list of conditions and the following
*     disclaimer in the documentation and/or other materials provided
*     with the distribution.
*   * Neither the name of the Jiri Horner nor the names of its
*     contributors may be used to endorse or promote products derived
*     from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
```

```
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
*  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
*  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
*  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
*  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
*  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
*  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
*  POSSIBILITY OF SUCH DAMAGE.
*
*********************************************************************/

#include <explore.h>

#include <thread>

inline static bool operator==(const geometry_msgs::Point& one,
                              const geometry_msgs::Point& two)
{
 double dx = one.x - two.x;
 double dy = one.y - two.y;
 double dist = sqrt(dx * dx + dy * dy);
 return dist < 0.01;
}

namespace explore
{
Explore::Explore()
 : private_nh_("~")
 , tf_listener_(ros::Duration(10.0))
 , costmap_client_(private_nh_, relative_nh_, &tf_listener_)
 , move_base_client_("move_base")
 , prev_distance_(0)
 , last_markers_count_(0)
{
 double timeout;
 double min_frontier_size;
 private_nh_.param("planner_frequency", planner_frequency_, 1.0);
```

```
 private_nh_.param("progress_timeout", timeout, 30.0);
 progress_timeout_ = ros::Duration(timeout);
 private_nh_.param("visualize", visualize_, false);
 private_nh_.param("potential_scale", potential_scale_, 1e-3);
 private_nh_.param("orientation_scale", orientation_scale_, 0.0);
 private_nh_.param("gain_scale", gain_scale_, 1.0);
 private_nh_.param("min_frontier_size", min_frontier_size, 0.5);

                                  search_                                    =
frontier_exploration::FrontierSearch(costmap_client_.getCostmap(),
                                                        potential_scale_,
gain_scale_,
                                                  min_frontier_size);

 if (visualize_) {
   marker_array_publisher_ =
       private_nh_.advertise<visualization_msgs::MarkerArray>("frontiers",
10);
 }

 ROS_INFO("Waiting to connect to move_base server");
 move_base_client_.waitForServer();
 ROS_INFO("Connected to move_base server");

 exploring_timer_ =
     relative_nh_.createTimer(ros::Duration(1. / planner_frequency_),
                              [this](const ros::TimerEvent&) { makePlan();
});
}

Explore::~Explore()
{
 stop();
}

void Explore::visualizeFrontiers(
   const std::vector<frontier_exploration::Frontier>& frontiers)
{
 std_msgs::ColorRGBA blue;
 blue.r = 0;
```

```
blue.g = 0;
blue.b = 1.0;
blue.a = 1.0;
std_msgs::ColorRGBA red;
red.r = 1.0;
red.g = 0;
red.b = 0;
red.a = 1.0;
std_msgs::ColorRGBA green;
green.r = 0;
green.g = 1.0;
green.b = 0;
green.a = 1.0;

ROS_DEBUG("visualising %lu frontiers", frontiers.size());
visualization_msgs::MarkerArray markers_msg;
std::vector<visualization_msgs::Marker>& markers = markers_msg.markers;
visualization_msgs::Marker m;

m.header.frame_id = costmap_client_.getGlobalFrameID();
m.header.stamp = ros::Time::now();
m.ns = "frontiers";
m.scale.x = 1.0;
m.scale.y = 1.0;
m.scale.z = 1.0;
m.color.r = 0;
m.color.g = 0;
m.color.b = 255;
m.color.a = 255;
// lives forever
m.lifetime = ros::Duration(0);
m.frame_locked = true;

// weighted frontiers are always sorted
double min_cost = frontiers.empty() ? 0. : frontiers.front().cost;

m.action = visualization_msgs::Marker::ADD;
size_t id = 0;
for (auto& frontier : frontiers) {
  m.type = visualization_msgs::Marker::POINTS;
```

```
    m.id = int(id);
    m.pose.position = {};
    m.scale.x = 0.1;
    m.scale.y = 0.1;
    m.scale.z = 0.1;
    m.points = frontier.points;
    if (goalOnBlacklist(frontier.centroid)) {
      m.color = red;
    } else {
      m.color = blue;
    }
    markers.push_back(m);
    ++id;
    m.type = visualization_msgs::Marker::SPHERE;
    m.id = int(id);
    m.pose.position = frontier.initial;
     // scale frontier according to its cost (costier frontiers will be
smaller)
    double scale = std::min(std::abs(min_cost * 0.4 / frontier.cost), 0.5);
    m.scale.x = scale;
    m.scale.y = scale;
    m.scale.z = scale;
    m.points = {};
    m.color = green;
    markers.push_back(m);
    ++id;
  }
  size_t current_markers_count = markers.size();

  // delete previous markers, which are now unused
  m.action = visualization_msgs::Marker::DELETE;
  for (; id < last_markers_count_; ++id) {
    m.id = int(id);
    markers.push_back(m);
  }

  last_markers_count_ = current_markers_count;
  marker_array_publisher_.publish(markers_msg);
}
```

```
void Explore::makePlan()
{
 // find frontiers
 auto pose = costmap_client_.getRobotPose();
 // get frontiers sorted according to cost
 auto frontiers = search_.searchFrom(pose.position);
 ROS_DEBUG("found %lu frontiers", frontiers.size());
 for (size_t i = 0; i < frontiers.size(); ++i) {
   ROS_DEBUG("frontier %zd cost: %f", i, frontiers[i].cost);
 }

 if (frontiers.empty()) {
   stop();
   return;
 }

 // publish frontiers as visualization markers
 if (visualize_) {
   visualizeFrontiers(frontiers);
 }

 // find non blacklisted frontier
 auto frontier =
     std::find_if_not(frontiers.begin(), frontiers.end(),
                      [this](const frontier_exploration::Frontier& f) {
                        return goalOnBlacklist(f.centroid);
                      });
 if (frontier == frontiers.end()) {
   stop();
   return;
 }
 geometry_msgs::Point target_position = frontier->centroid;

 // time out if we are not making any progress
 bool same_goal = prev_goal_ == target_position;
 prev_goal_ = target_position;
 if (!same_goal || prev_distance_ > frontier->min_distance) {
   // we have different goal or we made some progress
   last_progress_ = ros::Time::now();
   prev_distance_ = frontier->min_distance;
```

```
 }
 // black list if we've made no progress for a long time
 if (ros::Time::now() - last_progress_ > progress_timeout_) {
   frontier_blacklist_.push_back(target_position);
   ROS_DEBUG("Adding current goal to black list");
   makePlan();
   return;
 }

 // we don't need to do anything if we still pursuing the same goal
 if (same_goal) {
   return;
 }

 // send goal to move_base if we have something new to pursue
 move_base_msgs::MoveBaseGoal goal;
 goal.target_pose.pose.position = target_position;
 goal.target_pose.pose.orientation.w = 1.;
 goal.target_pose.header.frame_id = costmap_client_.getGlobalFrameID();
 goal.target_pose.header.stamp = ros::Time::now();
 move_base_client_.sendGoal(
     goal, [this, target_position](
               const actionlib::SimpleClientGoalState& status,
               const move_base_msgs::MoveBaseResultConstPtr& result) {
       reachedGoal(status, result, target_position);
     });
}

bool Explore::goalOnBlacklist(const geometry_msgs::Point& goal)
{
 constexpr static size_t tolerace = 5;
 costmap_2d::Costmap2D* costmap2d = costmap_client_.getCostmap();

 // check if a goal is on the blacklist for goals that we're pursuing
 for (auto& frontier_goal : frontier_blacklist_) {
   double x_diff = fabs(goal.x - frontier_goal.x);
   double y_diff = fabs(goal.y - frontier_goal.y);

   if (x_diff < tolerace * costmap2d->getResolution() &&
       y_diff < tolerace * costmap2d->getResolution())
```

```
    return true;
 }
 return false;
}

void Explore::reachedGoal(const actionlib::SimpleClientGoalState& status,
                          const move_base_msgs::MoveBaseResultConstPtr&,
                          const geometry_msgs::Point& frontier_goal)
{
 ROS_DEBUG("Reached goal with status: %s", status.toString().c_str());
 if (status == actionlib::SimpleClientGoalState::ABORTED) {
   frontier_blacklist_.push_back(frontier_goal);
   ROS_DEBUG("Adding current goal to black list");
 }

 // find new goal immediatelly regardless of planning frequency.
 // execute via timer to prevent dead lock in move_base_client (this is
 // callback for sendGoal, which is called in makePlan). the timer must
live
 // until callback is executed.
 oneshot_ = relative_nh_.createTimer(
     ros::Duration(0, 0), [this](const ros::TimerEvent&) { makePlan(); },
     true);
}

void Explore::start()
{
 exploring_timer_.start();
}

void Explore::stop()
{
 move_base_client_.cancelAllGoals();
 exploring_timer_.stop();
 ROS_INFO("Exploration stopped.");
}

void Explore::clearblacklist()
{
 frontier_blacklist_.clear();
```

```
}

}  // namespace explore
```

Explore.h

```
/*********************************************************************
*
* Software License Agreement (BSD License)
*
*  Copyright (c) 2008, Robert Bosch LLC.
*  Copyright (c) 2015-2016, Jiri Horner.
*  All rights reserved.
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*   * Redistributions in binary form must reproduce the above
*     copyright notice, this list of conditions and the following
*     disclaimer in the documentation and/or other materials provided
*     with the distribution.
*   * Neither the name of the Jiri Horner nor the names of its
*     contributors may be used to endorse or promote products derived
*     from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
*  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
*  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
*  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
*  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
*  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
```

```
*  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
*  POSSIBILITY OF SUCH DAMAGE.
*
*********************************************************************/
#ifndef NAV_EXPLORE_H_
#define NAV_EXPLORE_H_

#include <memory>
#include <mutex>
#include <string>
#include <vector>

#include <actionlib/client/simple_action_client.h>
#include <geometry_msgs/PoseStamped.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <ros/ros.h>
#include <visualization_msgs/MarkerArray.h>

#include <costmap_client.h>
#include <frontier_search.h>

namespace explore
{
/**
* @class Explore
* @brief A class adhering to the robot_actions::Action interface that
moves the
* robot base to explore its environment.
*/
class Explore
{
public:
 Explore();
 ~Explore();

 void start();
 void stop();
 void clearblacklist();

private:
```

```
/**
 * @brief  Make a global plan
 */
void makePlan();

/**
 * @brief  Publish a frontiers as markers
 */
void visualizeFrontiers(
    const std::vector<frontier_exploration::Frontier>& frontiers);

void reachedGoal(const actionlib::SimpleClientGoalState& status,
                 const move_base_msgs::MoveBaseResultConstPtr& result,
                 const geometry_msgs::Point& frontier_goal);

bool goalOnBlacklist(const geometry_msgs::Point& goal);

ros::NodeHandle private_nh_;
ros::NodeHandle relative_nh_;
ros::Publisher marker_array_publisher_;
tf::TransformListener tf_listener_;

Costmap2DClient costmap_client_;
actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
    move_base_client_;
frontier_exploration::FrontierSearch search_;
ros::Timer exploring_timer_;
ros::Timer oneshot_;

std::vector<geometry_msgs::Point> frontier_blacklist_;
geometry_msgs::Point prev_goal_;
double prev_distance_;
ros::Time last_progress_;
size_t last_markers_count_;

// parameters
double planner_frequency_;
double potential_scale_, orientation_scale_, gain_scale_;
ros::Duration progress_timeout_;
bool visualize_;
```

```
};
}

#endif
```

emotionClassifer.py

```
#!/usr/bin/env python
from __future__ import print_function
import torch
import cv2
import torchvision
import numpy as np
import rospy
import roslib # Needed to import ModelStates
import argparse
from gazebo_msgs.msg import ModelStates
from geometry_msgs.msg import Pose
from std_msgs.msg import Int32
from emotionTrainingSample import EmotionClassificationNet
from mie443_contest3.msg import EmotionMsg
from mie443_contest3.msg import EmotionFaceMsg
import matplotlib.pyplot as plt

class EmotionDetector(object):

   def __init__(self, args):
       #
       # Python 2.7 syntax.
       super(EmotionDetector, self).__init__()
       #
       # Load your emotion detector.
       self.model = EmotionClassificationNet()

self.model.load_state_dict(torch.load(args.model_file,map_location=torch.d
evice('cpu')))
       self.model.eval()
       #
       # Visualize.
```

```
        self.vis = args.vis
        print('Setting up subscribers.')
        self.emotion_sub = rospy.Subscriber('/emotion_img', EmotionFaceMsg,
self.emotionsub)
            self.emotion_pub = rospy.Publisher('/detected_emotion', Int32,
queue_size=1) # self.emotionsub ,
        self.emotion_file = open('detectedVictim.txt', 'w')

    def showImBatch(self, imgs):
        img_grid = torchvision.utils.make_grid(imgs)
        self.matplotlib_imshow(img_grid)

    def matplotlib_imshow(self, img):
        img2 = img + 0.5     # uncenter
        npimg = img.numpy().transpose([1,2,0])
        cv2.imshow('Input', npimg)
        cv2.waitKey(0)

    def emotionsub(self, msg):
        with torch.no_grad():
            imgs = msg.data
            w = msg.width
            h = msg.height
            b = msg.batch
                    imgs = torch.from_numpy(np.array(imgs)).view(b, 1, h,
w).float()
            if self.vis:
                print('Showing images.')
                self.showImBatch(imgs)
            emotions = self.model(imgs, True)
            #
            # Emotion voting -- take the most often voted for emotion, ties
are broken arbitrarily.
                    uniqueEmotions, counts = emotions.unique(sorted=True,
return_counts=True)
            print('uniqueEmotions:', uniqueEmotions)
            print('EmotionCounts:', counts)
            cnt_max, max_idx = counts.max(0)
            print(uniqueEmotions[max_idx])
            intmsg = Int32()
```

```
            intmsg.data = uniqueEmotions[max_idx].item()
            self.emotion_pub.publish(intmsg)
            self.emotion_file.write(str(uniqueEmotions[max_idx].item()))

    def logEmotionHistory(self):
        self.emotion_file.close()
#
# Parse the input arguments.
def getInputArgs():
      parser = argparse.ArgumentParser('MIE443_contest3 victim emotion
detector.')
                            parser.add_argument('--gpu',         dest='gpu',
default=torch.cuda.is_available(), type=bool, help='Use gpu for training')
                     parser.add_argument('--model',       dest='model_file',
default='mdl_best.pth', type=str, help='NN model to use for emotion
detection.')
             parser.add_argument('--vis',    dest='vis',    default=False,
action='store_true', help='Visualize the received images.')
    args = parser.parse_args()
    return args
#
#
if __name__ == "__main__":
    rospy.init_node('emotionDetector')
    args = getInputArgs()
    victim_locations = EmotionDetector(args)
    rospy.spin()
```

emotionTrainingSample.py

```
import torch
import torch.nn as nn
import argparse
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
import torchvision.transforms as transforms

#import random #this is used to generate random integers for us to more
uniformly mix the augmented datasets together
```

```
#
# 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, 64, 3, padding=1),
           nn.ReLU(),
           nn.BatchNorm2d(64), #64
           nn.MaxPool2d(2, 2),
           nn.Dropout(0.25),

           nn.Conv2d(64, 128, 3, padding=1),
           nn.ReLU(),
           nn.BatchNorm2d(128), #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),
```

```
## added more layers

# added layers end
            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, 512),
            nn.ReLU(),
            nn.Dropout(0.25),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.25),
            nn.Linear(256, 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('C:/Users/panji/MIE443_Contest3/train_split.pth').exists():
        train_imgs, train_labels = torch.load('train_split.pth')
```

```
        #probs = torch.ones(train_imgs.shape[0]) * 0.3 #0.3
        #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]
          #torch.nn.Sequential
        transform_imgs_1 = transforms.Compose([
           transforms.RandomCrop(40,40),transforms.Resize(48)
           ])

        transform_imgs_2 = transforms.Compose([
           transforms.RandomHorizontalFlip()
           ])

        transform_imgs_3 = transforms.Compose([
           transforms.RandomRotation(180)
           ])

        train = (train_imgs, train_labels)

        #xyz  = [transform_imgs(img) for img in train[0]]
        #torch.cat((x, x, x), 0)

                                                        train_new        =
(torch.cat((train[0],transform_imgs_1(train[0]),transform_imgs_2(train[0])
,transform_imgs_3(train[0])),0),
torch.cat((train[1],train[1],train[1],train[1]),0))     #  train[0]  means
image train[1] means label

        print(train_new[0].shape)
        print(train_new[1].shape)

        return train_new
    else:
        print('The provided dataset does not exist!')
        exit(0)

def getDataloader(train, val):
```

```
   #train, val = getDataset(args)
   #xyz  = [transform_imgs(img) for img in train[0]]
   #torch.cat((x, x, x), 0)
                                                  #train_new                =
(torch.cat((train[0],transform_imgs_1(train[0]),transform_imgs_2(train[0])
,transform_imgs_3(train[0])),0),
torch.cat((train[1],train[1],train[1],train[1]),0))     #  train[0]  means
image train[1] means label
   #print(train_new[0].shape)

   train_dataset = torch.utils.data.TensorDataset(*train)

   val_dataset = torch.utils.data.TensorDataset(*val)

   #dataset = torch.utils.data.ConcatDataset([train_dataset, val_dataset])
   #
    # Due to class imbalance introduce a weighted random sampler to select
rare classes more often.
   batch_size = 64 #128
              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

########k ############
def get_k_fold_data(k, i, X, y): ###This process is mainly steps (1)
```

```
          # Return the training and verification data required for the
i-fold cross-validation, open separately, X_train is the training data,
X_valid is the verification data
   assert k > 1
     fold_size = X.shape[0] // k # Number of copies: total number of
data/fold (number of groups)

   X_train, y_train = None, None
   for j in range(k):
               idx = slice(j * fold_size, (j + 1) * fold_size)
#slice(start,end,step) slice function
       ##idx valid for each group
       X_part, y_part = X[idx], y[idx]
       if j == i: ###The i-fold is valid
           X_valid, y_valid = X_part, y_part
       elif X_train is None:
           X_train, y_train = X_part, y_part
       else:
             X_train = torch.cat((X_train, X_part), dim=0) #dim=0Increase
the number of lines, connect vertically
           y_train = torch.cat((y_train, y_part), dim=0)
   #print(X_train.size(),X_valid.size())
   train = (X_train, y_train)
   val = (X_valid, y_valid)
   return train, val

def train_loop(mdl, loss_fn, optim, dl, device):
   print('')
   pbar = tqdm(dynamic_ncols=True, total=int(len(dl)))
   n_batch_loss = 50
   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)
    train_total = getDataset(args)

    print(train_total[0].shape)
    print(train_total[1].shape)
```

```
   train_acc_sum ,valid_acc_sum = [],[]
   # Training loop.
   best_val = -float('inf')

   k = 4
   for i in range(k):

          train_split, val_split = get_k_fold_data(k, i, train_total[0],
train_total[1])
       train_dl,val_dl = getDataloader(train_split, val_split)

       mdl = EmotionClassificationNet()
       ce_loss = nn.CrossEntropyLoss()
                   optimizer   =   torch.optim.Adam(mdl.parameters(),lr   =
0.0001,weight_decay = 0.00001)
       device = torch.device('cpu')

       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:
           #
           for epoch in range(args.nepoch):
               print(i)
               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)
               print('Val ACC loop')
               val_acc = calc_acc(mdl, val_dl, 'val', device)
```

```
                #
                # Early stopping.
                if val_acc > best_val:
                    best_val = val_acc
                                torch.save(mdl.state_dict(), 'mdl_best.pth',
_use_new_zipfile_serialization=False)

          train_acc_sum.append(train_acc)
          valid_acc_sum.append(val_acc)

  print(train_acc_sum)
  print(valid_acc_sum)
```