MIE443H1S: Contest 3

Finding and Interacting with Emotional People in an Unknown Environment

1.1 Objective:

The goal of this contest is for the simulated TurtleBot to explore and find 7 users/victims in an unknown disaster environment and provide them with important emergency announcements regarding evacuation due to e.g., fire, earthquake, etc. The robot should be able to explore and map the environment in order to find all the victims, identify their current emotional state and appropriately interact with them based on these emotions. Based on our discussions in class, your job is to design the simulated TurtleBot to appropriately identify and react to the following 7 user emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral. Each interaction must be distinct and identifiable. The simulated TurtleBot will interact with the user using the primary/reactive emotions and secondary/deliberative emotions we discussed in lectures. Each robot emotion must be distinct, and it cannot be duplicated amongst the two categories of emotions.

1.2 Requirements:

The contest requirements are:

- The contest environment is a building with multiple rooms, an example is shown in Figure 1. Seven individuals are distributed throughout the environment. They are stationary and do not move. There are no objects in the environment, victim locations are marked by a person on a blue square for visualization. They are purely visual markers, and therefore their presence does not impact the exploration process. A new unknown environment will be generated by the Instructor/TAs for Contest 3 to test your code, you will not be given the contest environment in advance.

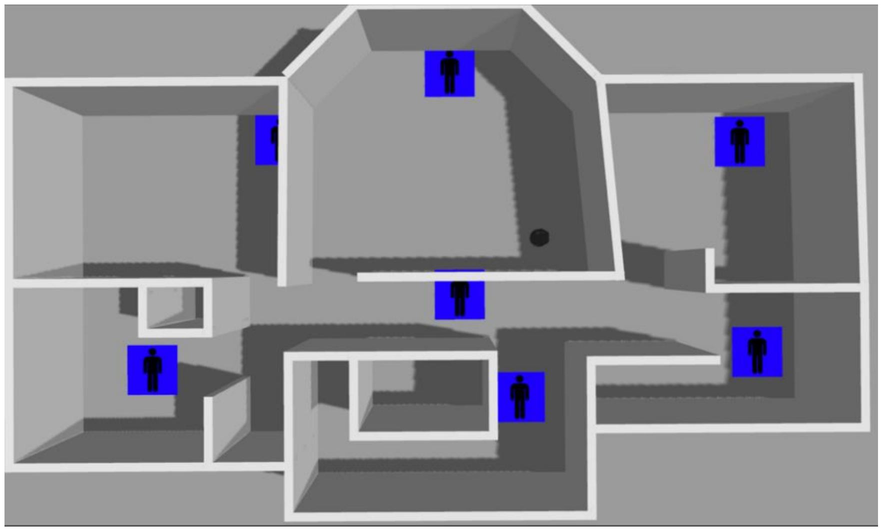


Figure 1: Example Environment Layout with 7 victims

- Your team will be provided with the following: 1) a dataset [1] of faces expressing one of the seven emotions, sample images are shown in Figure 2, 2) sample code to train and use an emotion classification convolutional neural network (CNN), 3) a victim detector that automatically detects simulated victims, and 4) a frontier exploration algorithm for your team to use to find the victims. During frontier exploration, the robot generates a map of the environment by navigating to the boundaries between known and unknown regions.

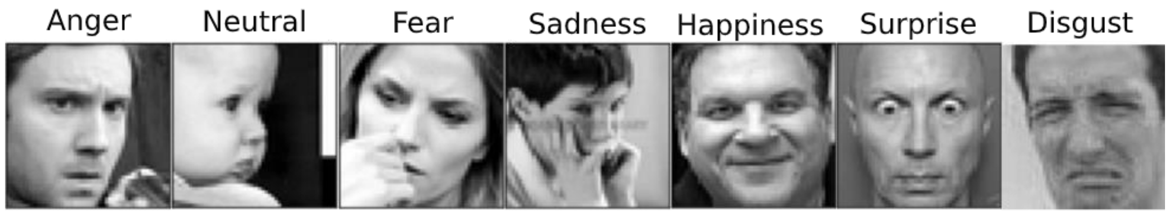


Figure 2: Example Environment Layout with 7 victims

- The victims are stationary in the environment. When your robot navigates within a pre-set distance of a victim's location (defined in victims.csv), the victim detector will provide multiple images of different faces with the same emotion for your CNN to classify.
- Please note that for the contest, the face images will be different than those provided in the contest package. Specifically, your classification method will be tested with a dataset of faces your team does not have access to. This dataset is referred to as a test set.
- For the interaction portion of the contest, the simulated TurtleBot will need to be designed to have unique emotional responses to the 7 possible emotions of the victims to aid with conveying important information. You will choose the type of disaster, earthquake, fire, explosion, etc., and have your robot inform all the victims with evacuation instructions. For example, "*the building is on fire and the building will be engulfed in flames in 20 minutes! Please evacuate*." During the interaction your team must choose **3** unique primary and **3** unique secondary emotions, and 1 additional unique emotion of either type for the robot based on the facial expressions of the victims (7 robot emotions in total).
- The robot's emotions must be chosen from the list provided below:

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

- You are encouraged to use as many input and output modes for the robot. Your team must implement original motion and/or sound for the emotion interaction part of the contest. You have access to the play_sound library which can be used to play stored .wav files.
- During the contest, the simulated TurtleBot will start at a location in the map of the Instructor/TAs choosing. The robot will have a maximum time limit of **20 minutes** to locate and interact with all victims.
- Your team can use a different emotion classification method than the example provided, as long as **no additional** software packages are installed. For instance, you can modify and add

to the CNN structure if you want, but you cannot use other software not provided to you, such as TensorFlow.

- As the robot traverses through the environment the emotions of the victims your robot has found are written automatically to the ‘detectedVictim.txt’ file. This file is only used by the TAs to know the ground truth emotion of each victim during your test run. You are welcome to use it during your practice runs to confirm the emotions. **But, your team cannot use this file for emotion recognition during the contest. If you do, you will automatically get a grade of 0 on the contest.**
- Please do not modify any automatically generated log files.
- The overall accuracy of your emotion classification method will be tested using the test set of images your team does not have access to. Namely, the dataset that contains entirely new images of faces with the same emotion classes as the dataset provided to train your emotion classification method.

1.3 Scoring (15%):

- Your team will be evaluated based on two contributions: 1) finding and classifying all the victims’ emotions, and 2) the robot’s interactive emotional response to these emotions and the evacuation instructions.
- Your team will be given a total of 2 trials. The best trial will be counted towards your final score.
- The scores are as follows. One mark will be given for finding and correctly identifying each victim’s emotional state (7 marks in total). One mark will be given for your robot emotion model to appropriately respond using the 3 primary and 3 secondary emotions, and 1 emotion from either type (7 marks in total). An additional 0.5 points for each emotion will be given based on the creativity and complexity of the emotional behaviour displayed by the robot (3.5 marks). Hence, you are strongly encouraged to display the robot’s emotions using multi-modal outputs.
- You will also receive up to an additional mark based on your team’s classification rate compared to all of the other teams. Namely, this mark is scaled linearly from 60% to the maximum score attained by the best classification rate in the class. A score below 60% will result in a receiving 0 marks here. Refer to the equation below for details:

$$\text{Classification Rate Mark } = min\left(max\left(0,\left(\frac{acc-60}{max_{acc_i \in ALLTEAMS_{ACC}}(acc_i)-60}\right)\right),1\right)$$

Note:

1. “acc” refers to your detection method’s accuracy on the test set.
2. You cannot get negative marks.

1.4 Procedure:

Code Development - The following steps should be followed to complete the work needed for this contest:

- Download the mie443_contest3.zip from Quercus. Then extract the contest 3 package (right click > Extract Here) and then move the mie443_contest3 folder to the catkin_ws/src folder located in your home directory.
- In a terminal, change your current directory to ~/catkin_ws using the cd command. Then run catkin_make to compile the code to make sure everything works properly. ****Please note if you do not run this command in the correct directory a new executable of your program will not be created.****
- Frontier exploration is implemented using the explore_lite package [2] provided by ROS. During exploration a map is generated using the ROS GMapping package [3]. Explore_lite implements a greedy frontier-based exploration method. A frontier refers to the boundary between known and unknown regions on the map generated by GMapping. During exploration the robot is instructed to move to each of these locations to map all of the unmapped regions. This process is very similar to Contest 1. Refer to the contest3.cpp file located in the mie443_contest3/src/ folder for guidance on how to enable and disable frontier exploration. When frontier exploration is disabled you can run your code for your robot to interact with a victim. If you do not disable frontier exploration before interacting with a victim the robot may move away from the victim before completing the interaction.
- It is recommended that robot navigation for this contest be handled by the ROS package called *move_base*. This library takes coordinates and orientations in the global map of the environment and drives the robot to a specified location while handling robot localization and obstacle avoidance. For more information, please refer to the reference material in Section A.1.3 in the Appendix.
- For emotion classification, it is recommended to modify the PyTorch classification CNN provided in the contest 3 files, defined in emotionTrainingSample.py. Please refer to the reference material in Section A.1.4 in the Appendix for more information and the corresponding tutorials with regards to this library. The PyTorch library is an open source machine learning library. It implements automatic differentiation, optimization procedures, common neural network layers, etc., required for training CNNs. In this contest it is used to train the emotion classification CNN using the facial expression training dataset provided. Please keep in mind that the provided sample code is a minimum viable solution, and therefore, it is important to understand how it functions so that your team can develop it further to meet the requirements for this contest.
- Your team can modify the contest3.cpp file to implement the emotional interaction with the victims in the environment. You must use your knowledge gained from previous contests to subscribe and publish to appropriate topics as needed.

Compiling:

- Every time the code is changed you must compile it from terminal in the catkin_ws directory using the following command. If you do not do this, a new executable will not be created when you run it.

```
catkin_make
```

Robot Gazebo Simulation: To simulate the TurtleBot and location of victims in the unknown environment, the following steps should be followed:

- Launching Gazebo, move_base, and GMapping
  - Begin by launching the simulated world through the following commands (you can change the world name if other world files are available in ../worlds folder):

    ```
    roslaunch mie443_contest3 turtlebot_world.launch world:=practice
    ```

  - To run GMapping and move_base in order to have the simulated robot map the unknown environment and perform point-to-point navigation run the following:

    ```
    roslaunch mie443_contest3 gmapping.launch
    ```

- To run the Victim Detector:
  - First install rospy and pytorch in the same working environment (you only need to do this once) run:

    ```
    conda env create --file mie443_contest3.yml
    ```

  - Then activate the environment (this sets up the bash paths to run the appropriate python commands similar to the setup.bash script you ran for your catkin package. Without activating the environment, python will not be able to find PyTorch):

    ```
    conda activate mie443
    ```

  - To run the victim locator code, run the src directory of the mie443 contest 3 folder:

    ```
    python victimLocator.py
    ```

Training the Emotion Classifier:

- To train an Emotion Classifier using the sample training code, the following must be done:
  - First activate the environment:

    ```
    conda activate mie443
    ```

  - To train a CNN and save a model run the following:

    ```
    python emotionTrainingSample.py
    ```

- To run the Emotion Classifier:
  - First activate the environment:

    ```
    conda activate mie443
    ```

  - To run the emotion classification model, run:

    ```
    python emotionClassifier.py
    ```

- To open RVIZ to visualize the current map, the estimated robot pose, and other data published on the ROS network run the following command:

  ```
  roslaunch turtlebot_rviz_launchers view_navigation.launch
  ```

Running the contest code:

- To run the frontier exploration code and your own code to interact with victims run:

```
rosrun mie443_contest3 contest3
```

- If you are also using sounds in your contest code also run:

```
rosrun sound_play soundplay_node.py
```

- If your team chooses to test in a different environment than what was provided, then the provided victim locations **will not work**. To update the victim locations, do the following:
    - Navigate the simulated TurtleBot to the location of the victim in the environment and then run:

    ```
    rostopic echo /gazebo/model_states
    ```

    - Record the *x*, *y* location of the robot's mobile base and set a detection radius into "victims_new.txt" on a new line. The file must have the same structure as victims.txt described below. Save the file in the src directory of the project.

Cancel all processes by pressing Ctrl-C in their respective terminals when you are done.

Code Breakdown – contest3.cpp

- Main block and ROS initialization functions.

```
int main(int argc, char** argv) {
    //
    // Setup ROS.
    ros::init(argc, argv, "contest3");
    ros::NodeHandle n;
    //
    // Frontier exploration algorithm.
    explore::Explore explore;
    //
    // Class to handle sounds.
    sound_play::SoundClient sc;
```

- To display emotions with sounds (i.e., vocal intonation), you can add sound files in the form of .wav files to be played by your TurtleBot. First add your desired sound files in the sounds directory within the mie443_contest3 package. A sample line of code "sc.playWave ("path_to_sound")" is shown below to demonstrate playing .wav files using the sound_play library.

```
    //
    // The code below shows how to play a sound.
    std::string path_to_sounds = ros::package::getPath("mie443_contest3") + "/sounds/";
    sc.playWave(path_to_sounds + "sound.wav");
```

- The code below shows you how to stop and start exploration. Prior to any interaction with a user you should execute the "explore.stop();" function, execute your interaction code, and then execute the "explore.start();" function. This ensures that your robot does not continue exploring the environment during your interaction.

```
    //
    // The code below shows how to start and stop frontier exploration.
    explore.stop();
    explore.start();
```

- This block of code below can be modified to implement your team's robot emotional interaction with the victims. The explore code is asynchronous to this while loop.

```
    while(ros::ok()) {
        // Your code here
        ros::spinOnce();
        ros::Duration(0.01).sleep();
    }
```

Code Breakdown – victimLocator.py

- This file implements victim localization. It publishes an image message when a victim is located in the environment. Victims are considered located if the simulated TurtleBot is within a distance threshold (as defined in victims.txt) of the victim. An example detection radius of 1m is shown in Figure 3. Your team **should not** modify this file.

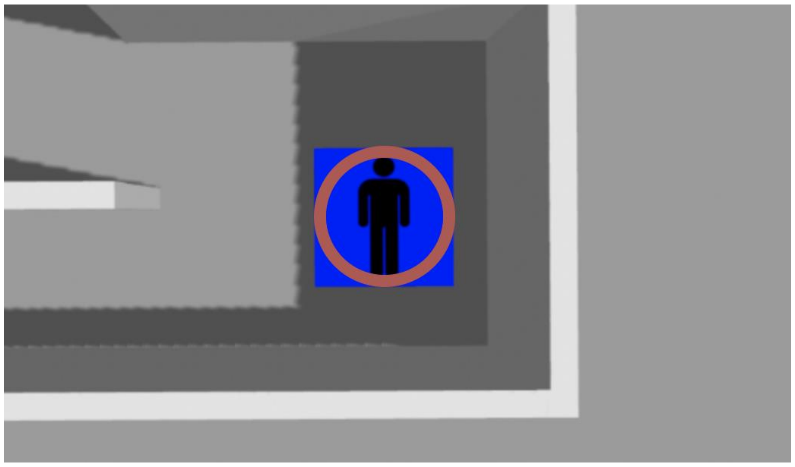
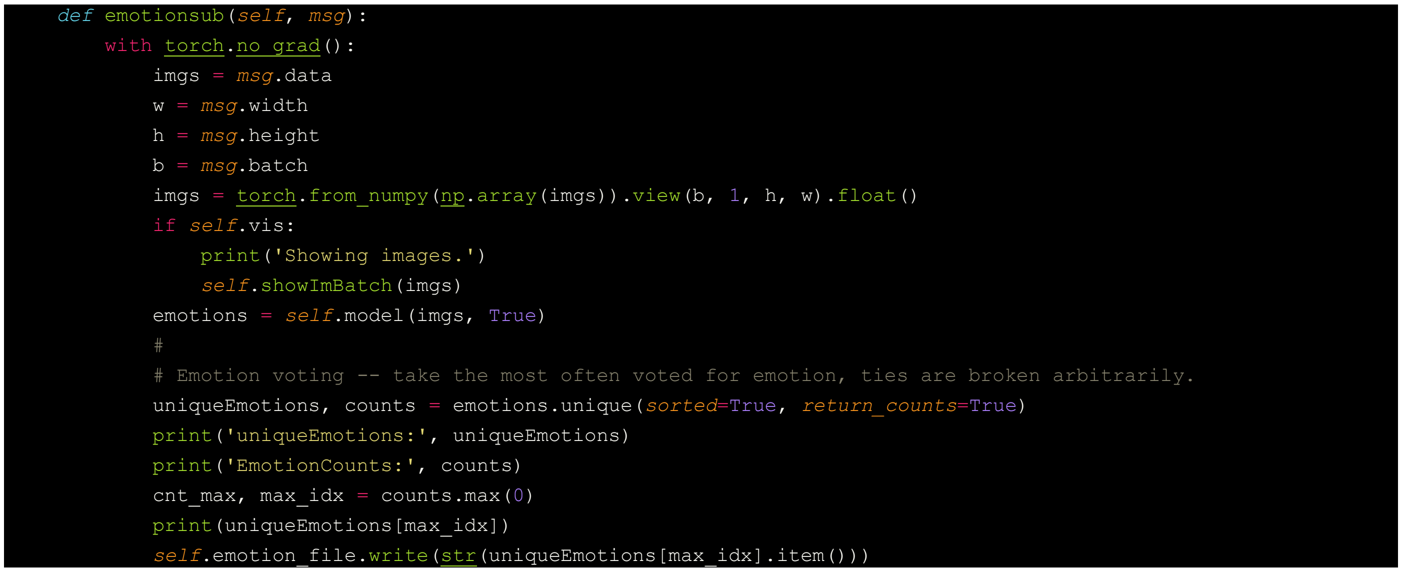


Figure 3: Example detection threshold of 1 meter.

Code Breakdown – emotionClassifier.py

- This file implements the classification of emotions at run-time. That is, it uses the CNN to predict the emotion of images of faces at different user locations.
- The *emotionsub* function is the callback function called when the victim locator. It publishes a set of victim images (recall, multiple images of faces with the same emotion are published by the victim locator). It implements an ensemble estimate of the published emotion via voting. Here ensemble refers to the use of multiple predictions to refine the estimate of a single classification prediction. This code can be modified as needed if your team wants to explore different ensemble strategies or if you would like to modify the structure of your classifier. Please test your code to ensure that it works as intended.



```python
    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])
            self.emotion_file.write(str(uniqueEmotions[max_idx].item()))
```

Code Breakdown – emotionTrainingSample.py

- This file provides a sample for how to train a CNN using PyTorch.
- The EmotionClassificationNet class defines a simple CNN structure that is used for the task of emotion classification of faces. The self.cnn variable stores the convolutional part of the network, and the self.nn variable stores the fully connected part that is applied after the CNN extracts features. Each operation is applied to the input data sequentially as they were declared in the code. In train mode it returns class logits, and in test mode the predicted class index is returned.

```
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),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.Dropout(0.25),
        )
        self.nn = nn.Sequential(
            nn.Linear(1152, 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 test_mode:
            _, out = torch.max(out, 1)
        return out
```

- The *getdataset* function loads the provided dataset of faces from your computer's file system and automatically splits this dataset into a training set and a validation set. These two datasets are mutually exclusive and there is no overlap between them. The training set is used to train the parameters of your CNN model. The validation set is used to estimate your model's accuracy. The function provides a simple method for generating a validation set. It uses a Bernoulli distribution to randomly sample approximately 70% of the data for the training set and the remaining 30% of the data for validation. The code provided here is just an example of how to divide the dataset. Your team should implement a more rigorous cross validation method to train your model to provide a more accurate estimate of your model's performance on the validation dataset.

```python
def getDataset(args):
    import pathlib
    if pathlib.Path('./train_split.pth').exists():
        train_imgs, train_labels = torch.load('train_split.pth')
        probs = torch.ones(train_imgs.shape[0]) * 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]
        return (train_imgs, train_labels), (val_imgs, val_labels)
    else:
        print('The provided dataset does not exist!')
        exit(0)
```

- The *getDataloader* function implements the creation of the dataloaders that randomly sample data from the training and validation datasets and generate batches for training and testing. The inverse frequency of each emotion in the dataset is used to provide a weight to the sample frequency of each image. This is done to compensate for the data imbalance of the training set. In particular, since some of the classes have more images than others, we sample the classes with fewer examples from the dataset more often to offset this imbalance.

```python
def getDataloader(args):
    train, val = getDataset(args)
    train_dataset = torch.utils.data.TensorDataset(*train)
    val_dataset = torch.utils.data.TensorDataset(*val)
    #
    # Due to class imbalance introduce a weighted random sampler to select rare classes more often.
    batch_size = 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
```

- The *train_loop* function implements a batch gradient descent CNN training routine using pytorch. You **should not** modify this function.
- The *calc_acc* (not shown here) function calculates the accuracy of your model given a dataset. You **should not** modify this function.
- The main code sets up the CNN and runs training for a specified number of epochs. You **should not** modify this code.

Code Breakdown – victims.csv

- Csv files are files that contain variables separated by commas. The file type allows data to be stored in a very efficient structure for retrieval in code. Also, because it is csv and not C++ code, it does not need to be complied every time you make a change.
    - The headings in the following csv contain the *x*, *y*, position of a victim and a detection radius for victims in the map. The code reading this csv file automatically populates a list of victim positions and detection radii using each row of the file.

```
pos_x,pos_y,radius
-0.13,0.83,1.
-3.54,-0.52,1.
-6.12,-5.12,1.
1.24,-5.7,1.
6.01,-4.82,1.
5.69,-0.6,1.
0.05,-3.64,1.
```

Code Breakdown – <file_name>.pth

- Files ending with ".pth" are data files saved by pytorch. For example, mdl_best.pth contains the saved CNN parameters, and train_split.pth is the training set.

1.5 Code Submission:

- In the README.md, please include the following:
    - Group Number and all names of students in your group.
    - The list of commands that we need to run in order to execute your group's code as needed in consecutive order.
    - Any computer specific *file path string* that we need to modify for your code to run.
    - Where exactly the output file will be stored.

1.6 Report:

- The report for each contest is worth half the marks and should provide detailed development and implementation information to meet contest objectives (15 marks).
- Marking Scheme:
    - The problem definition/objective of the contest. (1 mark)
        - Requirements and constraints
    - Strategy used to motivate the choice of design and winning/completing the contest within the requirements given. (2 marks)
    - Detailed Robot Design and Implementation including:
        - Sensory Design (4 marks)
            - All Sensors used, motivation for using them, and how are they used to meet the requirements for the contest including functionality.

        - Controller Design, both low-level and high-level control (including architecture and all algorithms developed) (5 marks)
            - Architecture type and design of high-level controller used (adapted from concepts in lectures)
            - Low-level controller
            - All algorithms developed and/or used
            - Changes and additions to the existing code for functionality
    - Future Recommendations (1 mark)
        - What would you do if you had more time?
        - Would you use different methods or approaches based on the insight you now have?
    - Full C++ ROS code and Python CNN training code (in an appendix). Please do not put full code in the text of your report. (2 marks)
    - Contribution of each team member with respect to the tasks of the particular contest (robot design and report). (Towards Participation Marks)
    - The contest package folder containing all your code, README file, etc. that will also be submitted alongside the softcopy (PDF version) of your report. Check and make sure your code runs before submitting (Towards the Contest and Code Marks Above)

1.7 Individual Report (One per group member): Each group member will also submit an individual report answering a set of questions on Contest 3 development to be completed on your own. These questions will count towards your Individual Participation Marks (10%).

1.8 Reference Materials for the Contest:

The following materials in Appendix A will be of use for Contest 3:

- A.1.1
- A.1.3
- A.1.4

References

[1] I. J. Goodfellow *et al.*, "Challenges in representation learning: A report on three machine learning contests," in *International conference on neural information processing*, 2013, pp. 117–124.
[2] "explore_lite - ROS Wiki." http://wiki.ros.org/explore_lite (accessed Dec. 08, 2020).
[3] "gmapping - ROS Wiki." http://wiki.ros.org/gmapping (accessed Dec. 08, 2020).