UNIVERSITY OF TORONTO

DESIGN REPORT

MIE443 - Mechatronics Systems: Design and Integration

Contest #3: Follow Me Robot Companion

Authors:
Sophie Miller - 1005811392
Jenna Del Fatti - 1006018738
Tiger Luo - 1008306502
Stephanie Beals - 1004450872
Kevin Hsu - 1005367728

Professor:
Goldie Nejat

Head Teaching Assistant:
Aaron Tan

April 9th, 2024

1.0 Introduction and Problem Definition

Robots that are able to show emotions in human-robot interactions are said to create an illusion of intelligence and connection, making the interactions more natural [1]. Therefore, robots that show emotion can modify human perception of robots and enhance the way we interact with them. As the use of robots in industrial and social applications is becoming more normalized, the ability of robots to interact with humans in ways that resemble human-to-human interaction becomes increasingly important [2]. Therefore, the way that robots effectively express the right emotions is an area of high research interest; robots can use facial expression, gesture, posture, body language, speech, motion and movement to communicate positive and negative feelings [1].

1.1 Objective of Contest

The objective of Contest 3 is to develop and program an algorithm for the TurtleBot to effectively engage with users as a companion robot. The primary goal is to build an interactive TurtleBot to find and follow a user while they move through an environment and interact with the user through emotional expression. The user will instruct four labeled markers in the environment where the TurtleBot will interact with the user through an emotional response to four environmental stimuli. The emotional response of the robot must include two primary/reactive emotions and two secondary/deliberative emotions selected from the list in **Table 1** below. Reactive emotions are synonymous to alarm mechanisms, where internal stats are triggered and react quickly to produce a global change to motor or sensory control; reactive emotions often override other processes [1]. Deliberative or secondary emotions are defined as adaptations to location specific events and environment characteristics that enable the robot to pursue other activities than the original goal; they are often triggered by "what ifs" and thoughts about what may or may not have happened [1]. Secondary emotions are often reactions to primary emotions and are more complex, often influenced by past experiences, personal beliefs, or thoughts [3].

Table 1. Emotional Response Options

● Fear	● Hate
● Positively Excited	● Resentment
● Infatuated	● Surprise
● Pride	● Embarrassment
● Anger	● Disgust
● Sadness	● Rage
● Discontent	

1.2 Design Requirements & Constraints

The design requirements and constraints for Contest 3 are provided in the MIE 443 Contest 3 Instructions 2024 [4], they are described below in **Table 2.**

Table 2. Design Requirements & Constraints

Design Requirements & Constraints
The TurtleBot must identify and track a user throughout the environment, following the user to 4 labeled markers in the environment.
The TurtleBot has a maximum time limit of 8 minutes to show robot functionality and emotional responses.
The TurtleBot must be designed to have four unique responses to four environmental stimuli. Two stimuli are predefined by the Contest 3 requirements: 1) Losing track of the user it is following 2) Blocked from following user due to static obstacle in its path The remaining two stimuli are up to the discretion of the team and are further described in subsequent sections.
The Turtlebot's reactions to the environmental stimuli must be distinct (no duplicate emotional responses); Turtlebot must have two primary/reactive emotions and two secondary/deliberative emotions (i.e., four emotions in total).
The team must implement original motion and sounds for the emotional responses; full access to the *play_sound* library is granted to play the .wav file. Emotions of the robot must be implicitly conveyed, no explicit statement of robot emotion can be made by the TurtleBot or the team.
Turtlebot emotions must be chosen from the provided list in **Table 1.**
C++ code must be non blocking**

***self imposed by group throughout design process*

2.0 Strategy and Methodology

Similar to the two prior contests, the primary design objective is to maximize the reliability of the implemented algorithm while maintaining a high level of engagement with the user and the environment using emotional responses. As project requirements call for primary and secondary emotional responses, this design objective is achieved by employing the different architectural underpinnings required. For primary reactions, this includes sensory feedback from system odometry, the bumper sensor, and depth sensor to trigger rapid global signal patterns that are sent to the motors and other subsystems to portray the associated emotion. Secondary emotions use a deliberative control architecture that follows a ***sense, plan, act***

order of decision making when the TurtleBot interacts with its environment, sending signals from deliberative mechanisms to fast reactive mechanisms that trigger reactions. The emotional responses are based on behavioral control: when the Turtlebot interacts with the contest stimuli, it will react according to a set of behaviors programmed into the algorithm.

The team has chosen to tell a story by using zoomorphism to design the follow me robot companion algorithm as if it were the user's personal pet. Specifically, the team has modeled the Turtlebot as a cat. Zoomorphism is defined as attributing animal characteristics to non-animals [1]. This choice was made so that all emotional reactions would follow the same theme and to mimic the interactive nature and companionship between cats and owners as there is a reliance for physical and emotional support [5]. Further, creating a storyline in the Turtlebot's interaction with the user more effectively conveys the emotions being displayed.

Table 3 below briefly summarizes the emotional reactions and strategy functionalities that were implemented within the state machine to execute the robot companion algorithm. Each reaction type is identified, the stimuli is listed, the emotional response mechanism (i.e., movement, sound, visual), and justification for the use of each emotion are provided. Their implementation will be discussed in further detail in the high and low level control descriptions in **Section 3.0.** The images of the visual responses are also shown in **Section 3.0.**

Table 3. Summary of Strategies

Emotion 1	**Positively Excited**
Type	Secondary/Deliberative
Stimuli	Shown a Picture of a Can of Tuna
Emotional Expression	***Movement:*** Spins in a circle ***Audio:*** "Yippee" exclamation ***Visual:*** A happy cat lying in front of a can of tuna
Justification of Usage	Positively excited was classified as a secondary emotion that is derived from happiness. This was chosen as a response to being shown a picture of tuna to mimic the learned behaviour of cats in response to food. This goes beyond the primary reaction of joy, as the cat (TurtleBot) would have had to learn overtime the positive correlation between food and excitement; pets are often food motivated and highly responsive when bribed.

Emotion 2	**Surprised**
Type	Reactive
Stimuli	Press/Kick of Left or Right Bumpers
Emotional Expression	***Movement:*** Turn quickly to face the direction of the kick ***Audio:*** Short cat shout ***Visual:*** Surprised Cat
Justification of Usage	Surprise was classified as a primary emotion as it is a common reaction to external stimuli that catches the person/thing off guard; it is difficult to control the reaction and is typically short in duration. Surprise was selected in response to a kick of the left and right bumpers to mimic the behaviour of a cat or pet if they are kicked or touched when they do not expect it. This reaction is typically short and includes a quick jerk or flinch motion due to the unexpected nature of the stimuli.

Emotion 3	**Fear**
Type	Reactive
Stimuli	Losing Track of the User
Emotional Expression	***Movement:*** Look left and right for the user followed by shaking in fear ***Audio:*** Cat screaming in distress ***Visual:*** Cat cowering
Justification of Usage	Fear was classified as a primary emotion as it is a common instinctual response to perceived threat. Fear was chosen as a response to losing track of the user as this stimulus can imitate abandonment and the psychological threat of harm if the owner/user is no longer there to provide the survival needs of the cat (TurtleBot). Pets and owners generally have strong attachments and losing track of the user can instill feelings of fear.

Emotion 4	**Rage**
Type	Secondary/Deliberative
Stimuli	Static Obstacle in Path; Unable to Track User

Emotional Expression	***Movement:*** Moves backwards and forwards twice to try and get around/break obstacle and then spins in rage ***Audio:*** Cat growling and low purring ***Visual***: Image of cat scratching followed by an image of a cat with red eyes
Justification of Usage	Rage was classified as a secondary emotion that is derived from anger, as it requires a consideration of the present situation or past experiences to grow into an uncontrollable and violent fury. Rage was chosen as a response to the obstacle stimuli as a cat may become extremely violent and destructive when they are blocked from their owner (i.e. the user) with whom there is an emotional connection. Considering the storyline of responses implemented in this contest, after experiencing negative emotions such as surprise and fear, the TurtleBot is likely to be frustrated after being "toyed" with. Being blocked from the user but still being able to see them is the final straw, and causes the TurtleBot to rage in response.

2.1 Contest 3 Storyline

The order of emotions outlined in **Table 3** above in **Section 2.0** is the order of execution for the two contest runs. Implementing the emotional reactions in this order was a design choice implemented by the team in order to tell a story. It begins with a positive emotion by showing the TurtleBot a picture of tuna to make it excited and spin around with joy, using food as a stimulant and reward. Next, it is surprised. The TurtleBot being kicked brings up feelings of shock or surprise, which begins a series of negative emotions and general dissatisfaction with its owner (the user). The following stimuli is losing track of the user which introduces a sense of fear, adding to the dissatisfaction and negative emotions of the TurtleBot. Finally, when the TurtleBot is blocked from following the user, feelings of frustration set in. Compiled with the feelings of surprise from being kicked, fear of being separated, and being blocked from the user, the TurtleBot bursts into feelings of rage, and violence.

3.0 Detailed Robot Design and Implementation

This section of the report details the team's solution to the problem through the sensory design and algorithm design implemented.

3.1 Sensory Design

There are multiple sensory units found in the TurtleBot 2 used in this contest, including an iClebo Kobuki mobile base and Microsoft Xbox Kinect 360 sensor. The iClebo Kobuki

features three cliff sensors, 2 wheel-drop sensors, an odometer, a gyroscope, and three bumpers, while the Kinect sensor hosts an RGB camera, depth sensor and microphone array [6]. In this contest, the implemented companion and emotional response algorithms utilize the following sensor groups to gather the required data from the environmental stimuli:

- RGB Camera
- Depth Sensor/Laser Scanner
- Bumpers

3.1.1 RGB Camera

The RGB camera is one of three main components of the Microsoft Kinect Sensor, the layout for which is shown below in **Figure 1.** The RGB Camera is made up of three channels, one for each of the primary colors: red (R), blue (B), green (G). The brightness of each pixel is determined independently, and the image pixel will consist of three RGB components with a range from 0-255 [7]. Therefore the overall pixel color is a combination of the RGB pixels, where each pixel is a 3D vector [8].

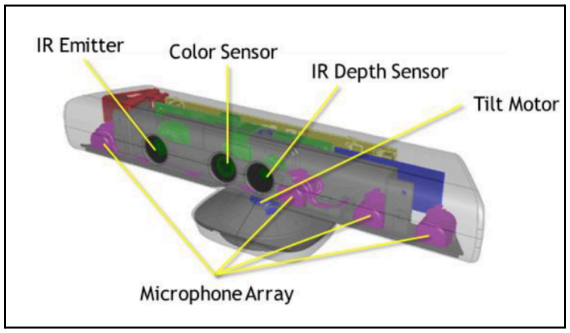

Figure 1. Microsoft Kinect 360 Sensor Components [8].

The RGB camera can collect a video feed which the Turtlebot stores for frame by frame processing [8]. The RGB camera is used in this contest to perform SURF feature detection in order to identify images that are used as stimuli for the emotional responses. Speeded Up Robust Features (SURF) is a high speed detection algorithm used to find distinctive keypoints in an image [9]. The key points of an image act as a control in image scanning and ensures that the same image is being tracked over multiple frames of the video feed (that is converted to images) from the Kinect sensor. The data collected from the RGB Camera is used by the ***matchImage*** function in the ***contest3.cpp*** file and is then stored in a matrix called ***knnMatch***, where it is used to compare a template image to the perceived image in order to trigger an emotional response.

3.1.2 Depth Sensor/Laser Scanner

The depth sensor is the second of three primary components of the Microsoft Kinect Sensor that is shown in **Figure 1** above. This sensor is an infrared projector with a complementary

metal oxide semiconductor (CMOS) image sensor [8]. The projector is an infrared (IR) laser emitter which projects a pattern of infrared beams onto objects and users in the environment. The projector also has an IR depth sensor that detects the IR beams that are reflected and to the depth sensor, which uses structured light that is projected onto an object [8]. The light pattern distorts the object, which can be used to determine depth information, as the coordinates (x,y,z) of each point in the distorted object pattern can be computed [8]. The provided follower package in the ***follower.cpp*** file uses the information collected by the Kinect depth sensor to locate and follow the user closest to the robot to carry out the companion algorithm.

3.1.3 TurtleBot Follower: *followerCB*

The TurtleBot follower code is used in this contest to enable the bot to follow the user when it detects an object in front of it. When an object is present in front of the TurtleBot, the follower code acts as a controller and publishes command velocities to maintain a distance of 1 meter between the bot and the user [10]. The velocity callback in the contest 3 code retrieves the velocities published from the follower code and saves it to a global twist variable so the follower can be called on in the main function [10]. The follower is used in default state 0 which is further explained in **Section 3.2.2.1.**

3.1.4 Bumper Sensors: *bumperCB*

The iClebo Kobuki base contains three bumpers along the front as shown in **Figure 2.** The sensor data from these bumpers return a state of either pressed or not pressed depending on real world conditions.

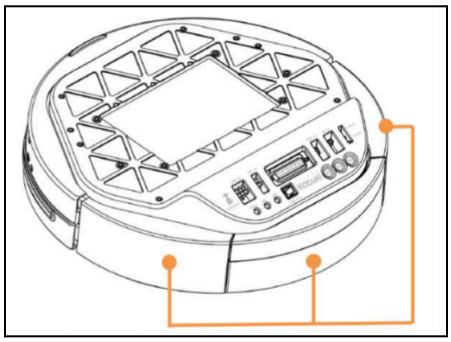

Figure 2. Right, center, and left bumpers on the Kobuki base [8].

In this contest, the sensory data from the bumpers is used to trigger two different emotional states. First, the right and left bumpers are used as the trigger of the TurtleBot being kicked by the user. This initiates the surprised response. Second, the center bumper is used as the trigger to inform the TurtleBot it has been stopped by an obstacle and can no longer follow its user. This then triggers the rage response. Within the bumper callback function, shown in **Appendix A,** the logic of which state to go into is dependent on which bumper is being

pressed as shown in **Figure 3.** Details on these specific states and emotional responses are given in **Section 3.2.**

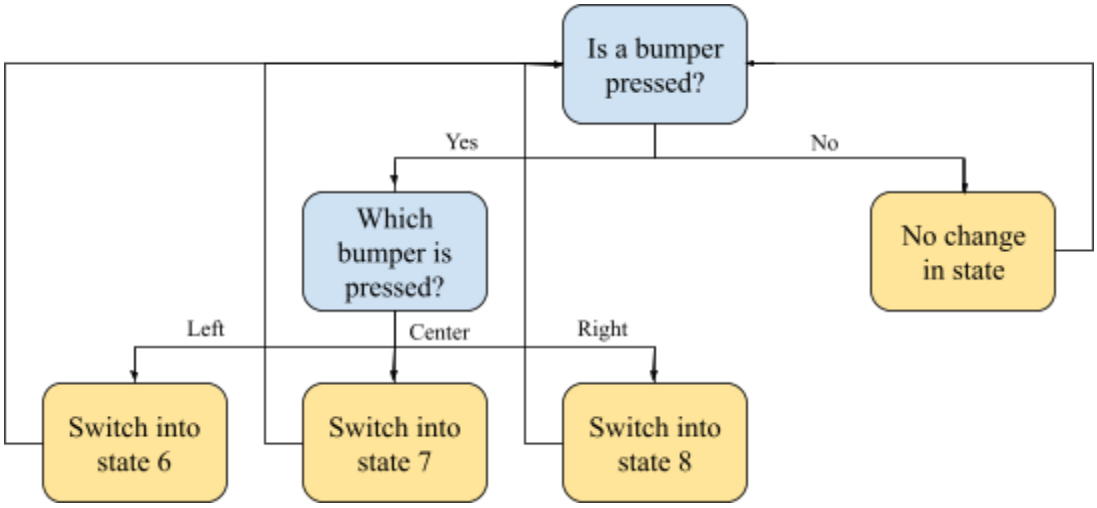

Figure 3. Logic followed in the bumper callback function.

3.2 Algorithm Design

This section outlines the overall algorithm design to implement the emotions storyline of the TurtleBot including low level and high level control. The low level control explains SURF feature detection as well as the individual functions that were developed in the code to control the TurtleBot's movements. The high level control includes an explanation of each emotion and how the functions are used together to convey the emotion as well as the state machine used to implement switches into different functions. The full C++ ROS code which was implemented for this contest is shown in **Appendix A**.

3.2.1 Low Level Control

This section outlines the low level control of the TurtleBot, including movement algorithms and libraries used to provide the baseline functionality to complete the contest.

3.2.1.2 SURF Feature Detection

Computer Vision (CV) is an open source computer vision software library used by the ROS package. There are many applications including image recognition and processing, and feature detection which are crucial functionalities for completing the contest objectives of this challenge as an emotional response is triggered by identifying an image by comparing it to a loaded reference image.

SURF feature detection, which was briefly described above in **Section 3.1.2** is used as it is supported by OpenCV. SURF stands for Speeded Up Robust Features; the function of the algorithm is to identify keypoints and descriptors in an image. These key points are rotation -

invariant and act as a control between snapshots taken of the reference photo. Key points are computed quickly for efficient detection. Image keypoints are 2D points identified from the perceived image using the ***minHessian*** threshold variable, which was set to 800 for this contest. Using SURF and the Kinect camera and sensor, the TurtleBot compares the perceived photo to the reference photo and determines if there are enough "good matches" between the keypoints. An example of an image with keypoints and matches is shown in **Figure 4**. **Appendix A** details the full implementation of the contest code using the SURF feature.

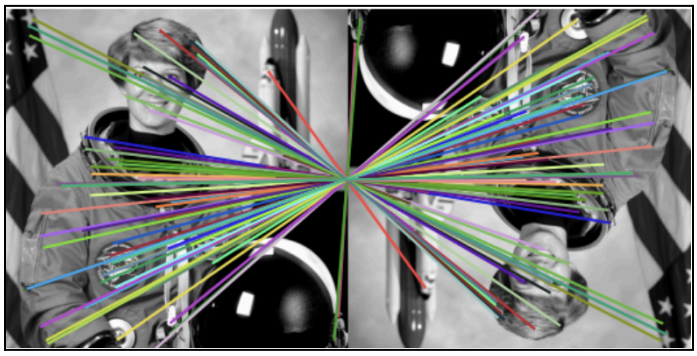

Figure 4. Image With Keypoints Identified Using SURF [11].

3.2.1.3 Image Transporter

The image transporter is a C++ file included in the code which is used to subscribe to and publish images [12]. It provides the capabilities of displaying images on the computer screen which is used throughout the contest to display various emotions. The file makes use of OpenCV's simple GUI capabilities to display images [13].

To display images throughout the contest a function called ***void imshow_emotion(Mat_img)*** is created. This makes use of the ***imshow()*** function to convert a ROS image message to an OpenCV image with BGR pixel encoding [13]. A wait key is implemented to show the image for one millisecond. The image then remains on the screen whenever called on until all windows are destroyed. It is important to note that whenever displaying an image, a ***Mat*** variable must first be created in combinated with the ***imread()*** function to point to the path directory of the file. The image can then be displayed on the screen using the ***imshow_emotion()*** function. The image window must be killed using ***cv::destroyAllWindows()*** when the image is no longer desired on the screen.

3.2.1.4 Timer Functions: *timerCB, void set_start_time(),* and *uint64_t get_time_elapsed();*

The purpose of the ***timerCB*** function is to serve as a timer function to control *state_lockou*t for 30-second time duration and automatically return to state 0 and reset *state_lockout* to false. *state_lockout* is a condition to lock the robot in a state while being triggered, allowing

the robot to perform the current state reaction while preventing any potential trigger of transition to other states before the 30-second timeout period.

The ***set_start_time()*** function retrieves the current time being kept by the system clock (`std::chrono::system_clock::now()`) and stores the value in the variable *start_time*. This function is called whenever one of the emotion states is triggered in order to set a time point for the start of a state.

The ***get_time_elapsed()*** function calls the current time and subtracts it by the start time point set by ***set_start_time()*** to obtain the time elapsed. This is used in emotion states where controlling the timing of certain visual or sound reactions is critical in maintaining the flow of logic, making the reaction and transition seem natural and true to life.

3.2.1.5 Image Detection: *bool matchImage(Mat inp_frame)*

Image detection utilizes SURF feature detection to obtain features from raw RGB images retrieved from the kinect camera, which are then passed through Lowe's ratio test to achieve feature matching. This is essentially a modified version of the contest 2 version, optimized for a constant stream of images to maintain a ROS loop rate of 10Hz. The template image that the scene is compared to is preprocessed and stored in a global array. As such, feature detection only happens on the scene images, and are then compared to the global feature array to determine matches.

3.2.1.6 Excited Emotion: *void excited()*

The excited emotion consists of simply spinning in place while playing a sound file, *excited.wav* periodically and displaying the image in **Figure 5**. In terms of complexity, it is the simplest emotion to convey, as well as the quickest.

Figure 5. Image displayed for the excited emotion.

3.2.1.7 Surprised Emotion: *void rightBumper()* and *void leftBumper()*

The surprised emotion is triggered by a change to states 6 or 8. State 6 is dedicated to the left bumper while state 8 is dedicated to the right bumper (see **section 3.2.2.4**). These states are triggered due to a person kicking the TurtleBot's left or right bumper. The team's inspiration for this emotion was the feeling one gets when getting bumped into aggressively in the subway. In this scenario one becomes surprised, usually looks in the direction one was bumped but then carries on with their previously projected action. The ***void rightBumper()*** and ***void leftBumper()*** therefore function similarly with the turtlebot turning 90 degrees towards the direction it was bumped in. If it gets hit on the left bumper it will turn to the left and if it gets hit on the right bumper it will turn right. After turning to look what bumped into him, he returns to state zero as that was its previously projected action. Additionally, a surprised cat image (**Figure 6**) is displayed on the monitor using the ***imshow_emotion*** accompanied by a surprised cat sound. The functions are structured in a way where the image is displayed first, the TurtleBot then carries out its movement with the short audio being played while the movement is carried out.

Figure 6. Image displayed for the surprised emotion.

3.2.1.8 Fear Emotion: *void fear()*

The fear emotion is triggered by a change to state 4 (see **Section 3.2.2.2**) and is in reaction to losing track of the user the TurtleBot is following. When the function is called, an image of a scared cat as shown in **Figure 7** is first displayed using the ***imshow_emotion*** function. The image remains on the screen for the duration of the function.

Figure 7. Image displayed throughout the fear emotion.

This function controls movements of the Turtlebot based on a timer. The ***get_time_elapsed()*** function is called on along with various if statements to trigger a sequence of movements the TurtleBot follows as shown in **Figure 8.**

Figure 8. Sequence of Turtlebot movements in the ***fear()*** function.

Looking to the left, right, and turning back to the center are set on a timer with a specific angular velocity to ensure a 90° turn to the left and right to mimic a cat looking side to side. The shaking from side to side is also implemented on a timer by continuously alternating directions of linear and angular velocities for a duration of 8 seconds. This shaking is conveying the fear of the cat. After all movements are complete, the image is taken off the screen through ***cv::destroyAllWindows()*** and the world state is set back to 0. The implementation of this function with the high level control and overall state machine in how the emotion is conveyed overall is explained in **Section 3.2.2**.

3.2.1.9 Rage Emotion: *void rage()*

The rage emotion is triggered by a change to state 7 (see **Section 3.2.2.5**) and is in reaction to encountering a static obstacle in its path and is therefore unable to follow the user. Similar to the fear emotion (**Section 3.2.1.8**), this function controls the movement of the TurtleBot using a timer by using the ***get_time_elapsed()*** function. Using a series of time constraints, the rage reaction is implemented by using a sequence of if statements to trigger TurtleBot movement. This sequence is depicted in **Figure 9** below.

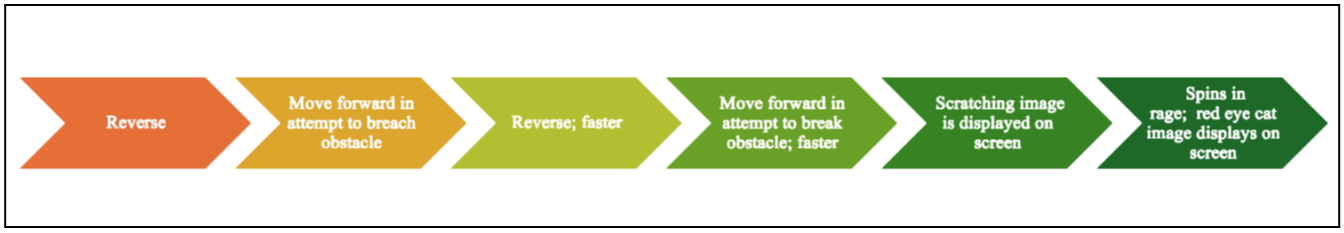

Figure 9. Sequence of TurtleBot movements in the ***rage()*** function.

The reversing and forward movements described in **Figure 9** above are set on a timer with specific linear velocities. On the second time this is done, the linear velocities are increased. These actions are to mimic the ramp up in anger and building frustration of the cat as they are unable to reach the user due to the obstacle. The TurtleBot repeats this action to try to reach the user again, only to be blocked by the same obstacle, therefore becoming violent and beginning to scratch at the obstacle as depicted in **Figure 10**. Then the function moves to spinning by setting the angular velocity for a specified amount of time using a timer. This mimics a temper tantrum and extreme frustration as the user is unable to be reached.

The ***rage()*** function uses two images to relay feelings of extreme anger and violence. The first image is of a frustrated and violent cat scratching and tearing up the carpet, as shown in **Figure 10**. This image is displayed after the cat approaches the obstacle for the second time. The second image is of a cat fuelled with red hot rage, as shown in **Figure 11**, and is displayed as the TurtleBot spins and remains on the screen for the duration of the function. The ***imshow_emotion*** function is used to display both images.

Figure 10. First image for the rage emotion.

Figure 11. Second image for the rage emotion.

After all movements are complete, the image is taken off the screen through ***cv::destroyAllWindows()*** and the world state is set back to 0. The implementation of this function with the high level control and overall state machine in how the emotion is conveyed overall is explained in **Section 3.2.2**.

3.2.1.10 Sound and Image Output

The two functions used for audio-visual output, ***imshow_emotion()*** and ***sc.playwave()*** utilize the CV and ROS libraries respectively to echo image or sound files from file storage. In order to optimize the code, and to prevent overlap when calling these functions, they run only once at specific timings, or at the start of a state. This is further elaborated in the next section on control and timing.

3.2.1.11 Follower Modifications

The follower code is modified slightly to accommodate another publisher that tracks follow velocity commands that are published, and publishes a boolean alongside it which tells whether or not a good centroid is found by the follower.

3.2.1.12 State Machine Control and Timing

The state machine tracks the current state of the robot using an integer variable, *world_state*. When *world_state* is set to 0, any of the four other states (emotions) can be entered upon a proper stimulus. However, to prevent behavior where a state may immediately enter another state while already in another emotional state, a *state_lockout* variable is set upon changing states which prevents any state changes, and is reset upon a timer, after which the *world_state* is also set back to 0. This timer is a callback based timer, using the **ros::Timer** library, as opposed to the timer functions *get_time_elapsed()* in order to act as a failsafe in case of bad state behavior. This timer is set to start upon entering a new state, and stops itself after the interrupt is triggered to wait for the next state change.

3.2.2 High Level Control - State Machine

The state machine, as previously explained, is based on timer callbacks and stimuli in order to switch between states. We can consider this as being a combination of both deliberative and reactive control measures, where while in state 0, the default following state, it is using reactive control, but while in emotional states, it is instead using deliberative control, and follows a set order of actions at predetermined timings. Note that each state might not use the same amount of time to complete, however the *state_lockout* will remain until the timer ends, preventing any state changes. While waiting however, the robot will still detect and follow people.

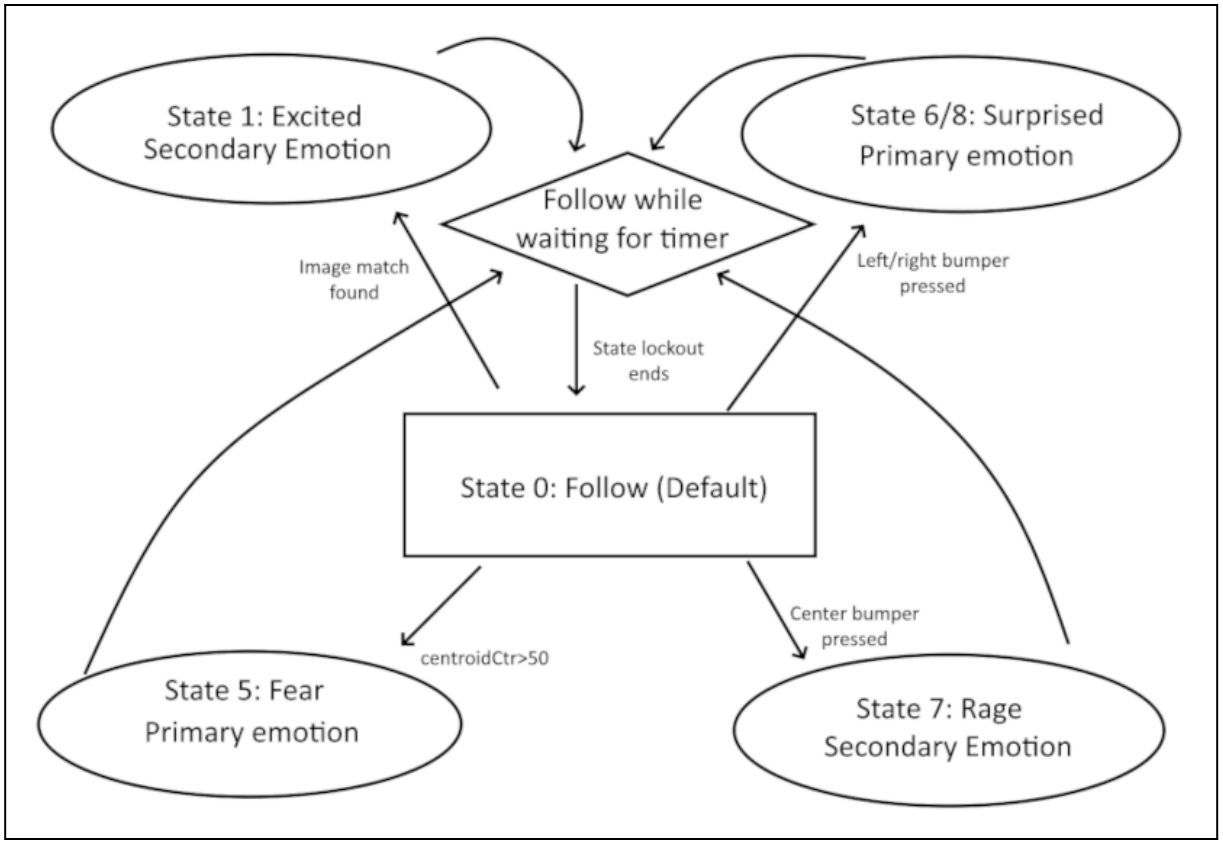

Figure 12. State machine structure and layout

The state machine as shown in **Figure 12** is implemented in two parts: a state evaluation portion which judges several conditional statements, and decides when to switch states, and a state execution portion which executes functions based on what state it is in.

The following sections explain each state of the machine and how the emotions are implemented. The combination of movement, audio, and visual to convey emotions is also explained.

3.2.2.1 State 0: Default Follower State

World state 0 in the state machine is the default movement of the TurtleBot. Under this condition, the Turtlebot is calling on the follower callback, enabling it to detect a person and follow them around the environment. It is in this state that the bot is able to navigate to all the marked points in the environment through following the user. In this state, an image of a neutral cat to convey no emotions is displayed on the computer screen as shown in **Figure 13.** The Turtlebot will not exit this follower state unless it is triggered by one of the stimuli which is further explained in **Sections 3.2.2.2** to **3.2.2.5.** In addition, when in any other emotion state, once the emotional response is complete, the TurtleBot is returned to this default state.

Figure 13. Neutral cat image displayed during the default follower.

3.2.2.2 State 4: Excited

The excited state is triggered by a change to state 4, which occurs upon the kinect camera detecting enough matching features in the raw rgb image to facilitate a state change. The object used to trigger the state change is a can of tuna, which is at the moment simply shown to the camera using a phone, but a real can of tuna would likely suffice, given it was the same brand logo used. Upon feature matching detecting enough good matches, it calls the ***excited()*** function and begins playing an excited sound every 2 seconds.

3.2.2.3 State 5: Fear

This state is triggered when the TurtleBot loses track of the user it is following. The emotion being displayed in this scenario is fear to mimic a real world response of a pet losing its owner. This follows zoomorphism theory in giving inanimate objects animal-like attributes.

In order to trigger the state change, a counter called *centroid_counter* must be greater than 50. The centroid counter is a timer that is implemented to determine if there is an object detected in front of the TurtleBot or not. If the counter is greater than 50, this means the TurtleBot has not detected an object in front of it for five seconds. This means the bot has lost track of the user and at this point will effectively respond to this stimulus by entering state 5.

Once the switch to state 5 is made, the sound file of a cat screaming is first played. This audio file plays for the entirety of the emotional response. The ***fear()*** function which was described in **Section 3.2.1.7** is then called to trigger the movement and visual responses of the emotion. Once the function is finished running, the bot is returned to state 0. The movement, visual, and audio called on this state work together to effectively convey the emotion of fear.

3.2.2.4 States 6 and 8: Surprised

This state is triggered when the TurtleBot is being kicked on its right and left bumpers. The emotion being displayed in this scenario is surprised to mimic a real response of a pet cat

being surprised when it is being kicked from the side by its owner. While the TurtleBot is not in state lockout and the previous state is set to state 0 (*prev_state = 0*), indicating that it is in the default follower state, this emotion can be triggered by pressing on the left or right bumper. This can be done either by having a person kicking on its left or right bumper, or if the TurtleBot bumped its left or right bumper onto something physically while trying to follow a person. When the state machine is switched to state 6 or 8, a .wav file of a short cat shout sound is played and the ***rightBumper()*** or ***leftBumper()*** function is called. After the function is finished, the TurtleBot returns to state 0 to continue with the default follower algorithm.

3.2.2.5 State 7: Rage

This state is triggered when the TurtleBot encounters an obstacle in the environment that blocks it from following the user. Physically, this is triggered when the center bumper is pressed. The emotion portrayed in this scenario is rage to mimic a real response of a pet being unable to reach its owner, after experiencing a sequence of negative emotions prior according to the algorithm storyline (see **Section 2.1**). Similar to the prior emotions, the TurtleBot must not be in state lockout completing a different emotional response thereby ensuring that the TurtleBot completes its emotional response before another is triggered. The previous state must be set to state zero (*prev_state = 0*) so that it is in the default follower state before encountering the next stimulus. When the state machine is switched to state 7, a .wav file of a cat growling and hissing is played and the ***rage()*** function is called; the .wav file plays for the duration of the function. After the function is finished, the TurtleBot returns to state 0.

4.0 Future Recommendations for Design Improvements

Although satisfied with the final product, if more time was available the team would have liked to implement a variety of improvements and additions to the design. Firstly the team wishes to have used videos instead of pictures in order to convey emotions more clearly. If this was accomplished the team would have liked to go further and have videos in sync with the movement of the robot. For example, if the TurtleBot is shaking in fear, the monitor could show a cat shaking at the same time as the robot.

Additionally the team wishes it would have more time to work on the “shaking” during the fear function to make it faster. The current shaking looks more like swaying due to it not shaking fast enough.

The team would also wish to make the states and the state machine more complex by allowing for more complex reactions. Currently, the timer set up is restrictive in that once a state is triggered, it cannot change to another and does not react to stimuli. It would be interesting if one emotion could lead to another if a stimulus is provided in the middle of a state, ie if the

states were more reactive instead of being purely deliberative. An example of this is the robot detects a person again when in the fear state, leading to a relieved state.

Lastly, the team is aware that the kinect sensor possesses a microphone that is often used for voice recognition [14]. The team would have gone a step further and found a way to utilize this aspect of the kinect to interact with the robot. An example of this would be being able to have the TurtleBot react to the user saying words like “treat” and having it come over to where it hears the noise. Another example would be having the TurtleBot react to move away when it hears loud sounds much like cats do.

Overall the team wished it had time to implement video display to convey emotions more clearly, as well as developing ways to interact with the robot through the kinect voice microphone.

5.0 Attribution Table

The following sections detail the contributions of each group member to the Contest 3 deliverables including the robot code development and report writing.

Table 4. Attribution Table

Section	Jenna	Tiger	Stephanie	Sophie	Kevin
Response #1: Positively Excited		TL			
Response #2: Surprised			SB		KH
Response #3: Fear	JD				
Response #4: Rage				SM	
State Machine		TL			
Image Implementation	JD			SM	
Sound Implementation	JD	TL	SB	SM	KH
Code Debug	JD	TL	BS	SM	KH
Simulations & Testing	JD	TL	SB	SM	KH
Report Sections	3.1.3, 3.1.4, 3.2, 3.2.1, 3.2.1.3, 3.2.1.9, 3.2.2.1, 3.2.2.3, 6.0, 7.0	3.2.1.5, 3.2.1.6, 3.2.1.10-12 3.2.2 3.2.2.1 3.2.2.4	3.2.1.7, 3.2.1.4, 3.2.2.4, 4.0	1.0, 1.1, 1.2, 2.0, 2.1, 3.0, 3.1, 3.1.1, 3.1.2, 3.2.1.2, 3.2.1.9, 3.2.2.5	3.2.1.7, 3.2.1.4, 3.2.2.4, 4.0

6.0 References

[1] G. Nejat, “MIE443 Lecture 6/7 Robot Emotions” [Online]. Available: https://q.utoronto.ca/courses/330231/files/folder/Lecture%20Slides?preview=30805229. (accessed Mar.30, 2024).

[2] A. Paiva, I. Leite, and T. Ribeiro, “Emotion modelling for Social Robots,” Emotion Modelling for Social Robots, https://people.ict.usc.edu/~gratch/CSCI534/Readings/ACII-Handbook-Robots.pdf (accessed Mar. 30, 2024).

[3] O. Guy-Evans, “Primary and secondary emotions: What’s the difference?,” Primary And Secondary Emotions: Recognizing The Difference, https://www.simplypsychology.org/primary-and-secondary-emotions.html#:~:text=Primary%20emotions%20are%20immediate%2C%20instinctual,experiences%2C%20beliefs%2C%20and%20thoughts (accessed Mar. 30, 2024).

[4] G. Nejat et al. “MIE443H1S: Contest 3 Follow Me Robot Companion” [Online]. Available: https://q.utoronto.ca/courses/330231/files/folder/Contests/Contest%20%233?preview=29694282. (accessed Mar.30, 2024).

[5] E. M. C. Bouma, M. L. Reijgwart, and A. Dijkstra, “Family member, Best Friend, child or ‘just’ a pet, owners’ relationship perceptions and consequences for their cats,” International journal of environmental research and public health, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8750854/ (accessed Mar. 30, 2024).

[6] Microsoft, “Kinect Sensor Components and Specifications.” [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx.

[7] G. Nejat et al. “TurtleBot Technical Manual for MIE 443H1S.” [Online]. Available: https://q.utoronto.ca/courses/330231/files/folder/Lab%20Manual?preview=29695423 . (accessed Mar.17, 2024)

[8] G. Nejat et al. “TurtleBot Technical Manual for MIE 443H1S.” [Online]. Available: https://q.utoronto.ca/courses/330231/files/folder/Lab%20Manual?preview=29695423 . (accessed Mar.17, 2024)

[9] N. Q. B. Nguyen, “OpenCV 4: Surf algorithm for feature detection,” LinkedIn, https://www.linkedin.com/pulse/using-opencv-4-surf-algorithm-feature-detection-quoc-bao-n

guyen#:~:text=Speeded%2DUp%20Robust%20Features%20(SURF,scale%2Dinvariant%20and%20rotation%20invariant (accessed Apr. 3, 2024).

[10] A. Fung et al. "Tutorial 5 - Follower Introduction and Finite State Machine." [Online]. Available: https://q.utoronto.ca/courses/330231/files/folder/Tutorials?preview=31171158. (accessed Apr. 9, 2024)

[11] "SIFT feature detector and Descriptor Extractor#," SIFT feature detector and descriptor extractor - skimage 0.22.0 documentation, https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_sift.html (accessed Apr. 5, 2024).

[12] "image_transport," ros.org, https://wiki.ros.org/image_transport (accessed Apr. 5, 2024).

[13] "Writing a Simple Image Subscriber (C++)," ros.org, https://wiki.ros.org/image_transport/Tutorials/SubscribingToImages (accessed Apr. 5, 2024).

[14] L. Pei, "Sound positioning using a small-scale linear microphone array," Science Direct, https://www.researchgate.net/publication/269306618_Sound_positioning_using_a_small-scale_linear_microphone_array (accessed Apr. 8, 2024).

7.0 Appendices

Appendix A - Full C++ ROS Code

```
#include <header.h>
#include <ros/package.h>
#include <imageTransporter.hpp>
#include <chrono>
#include <inttypes.h>
#include <std_msgs/Bool.h>
#include <kobuki_msgs/BumperEvent.h>

//BUMPER
#define N_BUMPER (3)

//BUMPER
//GLOBAL VARIABLE FOR BUMPER
uint8_t bumper[3]={kobuki_msgs::BumperEvent::RELEASED, kobuki_msgs::BumperEvent::RELEASED, kobuki_msgs::BumperEvent::RELEASED};

using namespace std;
using namespace cv;
using namespace cv::xfeatures2d;

//State or nav related
geometry_msgs::Twist follow_cmd;
int world_state;
int prev_state;
bool state_lockout = false;
bool obj_detected = false;

//Timer related global vars/structs
ros::Timer state_timer;
std::chrono::time_point<std::chrono::system_clock> state_start;
void set_start_time(); //Start time is set on state change
uint64_t get_time_elapsed(); //Returns millis since state start

bool enable_show_img = false;
int key_time = 0;
int minHessian = 800;
const float ratio_thresh = 0.75f;
//Preallocate memory needed for image matching to make things faster
std::vector<KeyPoint> keypoints_ref, keypoints_img;
Mat descriptor_ref, descriptor_img;
Ptr<SURF> detector = SURF::create( minHessian );
Mat img_ref;

bool matchImage(Mat inp_frame);
void excited();
void rightBumper();
void leftBumper();
void rage();
void fear();
void turnLeft();
void turnRight();


void followerCB(const geometry_msgs::Twist msg){
    follow_cmd = msg;
    // if (follow_cmd.linear.x == 0 && follow_cmd.angular.z == 0){
    //  obj_detected=false;
    // } else obj_detected=true;
}

void centroidCB(const std_msgs::Bool msg){
    obj_detected=msg.data;
    // if (!obj_detected) ROS_INFO("Can't see person!");
    // else ROS_INFO("I see person!");
}
```

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "image_listener");
    ros::NodeHandle nh;
    state_timer = nh.createTimer(ros::Duration(30.0), timerCB);
    state_timer.stop();
    sound_play::SoundClient sc;
    string path_to_sounds = ros::package::getPath("mie443_contest3") + "/sounds/";
    teleController eStop;

    //publishers
    ros::Publisher vel_pub = nh.advertise<geometry_msgs::Twist>("cmd_vel_mux/input/teleop",1);

    //subscribers
    ros::Subscriber follower = nh.subscribe("follower_velocity_smoother/smooth_cmd_vel", 10, &followerCB);
    ros::Subscriber bumper = nh.subscribe("mobile_base/events/bumper", 10, &bumperCB);
    ros::Subscriber centroid_bool = nh.subscribe("centroid_bool",1,&centroidCB);

    // contest count down timer
    ros::Rate loop_rate(10);
    std::chrono::time_point<std::chrono::system_clock> start;
    start = std::chrono::system_clock::now();
    uint64_t secondsElapsed = 0;

    //imageTransporter rgbTransport("camera/image/", sensor_msgs::image_encodings::BGR8); //--for Webcam
    imageTransporter rgbTransport("camera/rgb/image_raw", sensor_msgs::image_encodings::BGR8); //--for turtlebot Camera
    imageTransporter depthTransport("camera/depth_registered/image_raw", sensor_msgs::image_encodings::TYPE_32FC1);

    world_state = 0;
    prev_state = 0;
    bool ranOnce=false;

    geometry_msgs::Twist vel;
    vel.angular.z = angular;
    vel.linear.x = linear;

    initAll();

    //sc.playWave(path_to_sounds + "rage.wav");
    ros::Duration(0.5).sleep();
    //Mat img_test=imread(ros::package::getPath("mie443_contest3")+"/imgs/imgtest.jpg");
    //imshow_fast(img_test);
    //imshow_emotion(img_test);

    set_start_time();

    int emotionCtr = 0;
    int centroidCtr = 0;
    world_state = 8;
    state_lockout = true;

    while(ros::ok() && secondsElapsed <= 480){
        ros::spinOnce();
        //State evaluation, put the most important states first! timerCB will automatically return to state 0 and reset state_lockout
        //ROS_INFO("Time elapsed: %" PRIu64 "\n", get_time_elapsed());


        if (world_state == 0 && !obj_detected){
            centroidCtr++;
        } else if (world_state != 0 ){
            centroidCtr = 0;
        } else {
            centroidCtr=0;
        }

        if (!state_lockout && prev_state == 0 && centroidCtr>50) { //Each ctr is 100ms
            world_state = 5;
            centroidCtr = 0;
        } else if (!state_lockout && prev_state == 0 && get_time_elapsed()%500 < 100 &&
        matchImage(rgbTransport.getImg())){
            world_state = 4;
        }


        if (prev_state != world_state && world_state != 0){
            state_timer.start();
            state_lockout=true;
            set_start_time();
            ROS_INFO("Starting lockout");
            ROS_INFO("State = %d", world_state);
        }
        if (prev_state != world_state && world_state == 0) ROS_INFO("State = 0");
        prev_state=world_state;
```

```
    if(world_state == 0){
        //robot follows person

        //display neutral emotion image
        Mat neutral=imread(ros::package::getPath("mie443_contest3")+"/imgs/neutralCat.jpeg");
        imshow_emotion(neutral);

        ranOnce=false;
        vel_pub.publish(follow_cmd);

    }else if(world_state == 1){
        /*
        nothing happens here can delete
        ...
        ...
        */
        vel.angular.z = angular;
        vel.linear.x = linear;
        vel_pub.publish(vel);

    }else if (world_state == 4){
        //cat sees picture of tuna and gets excited
        if (!ranOnce) {
            sc.playWave(path_to_sounds + "excited.wav");
            ranOnce=true;
        }
        if (get_time_elapsed()%2000 < 200) {
            sc.playWave(path_to_sounds + "excited.wav");
        }
        excited();
        vel.angular.z = angular;
        vel.linear.x = linear;
        vel_pub.publish(vel);

    }else if(world_state == 5){
        //cat has lost track of person and gets scared
        if (!ranOnce) {
            sc.playWave(path_to_sounds + "fear1.wav");
            ranOnce=true;
        }
        if (get_time_elapsed()>10000 && get_time_elapsed()<10200){
            sc.playWave(path_to_sounds + "fear2.wav");
        }

        fear();
        vel.angular.z = angular;
        vel.linear.x = linear;
        vel_pub.publish(vel);

    }else if(world_state == 6){
        //left bumper has been kicked and cat gets surprised
        if (!ranOnce) {
            sc.playWave(path_to_sounds + "surprise.wav"); // default sounds
            ranOnce=true;
        }
        leftBumper();
        vel.angular.z = angular;
        vel.linear.x = linear;
        vel_pub.publish(vel);


    }else if(world_state == 7){
        //center bumper is pressed due to obstacle and cat cannot get to person and rages
        if (!ranOnce) {
            sc.playWave(path_to_sounds + "rage1.wav");
            ranOnce=true;
        }
        if (get_time_elapsed()>10000 && get_time_elapsed()<10200) {
            sc.playWave(path_to_sounds + "rage2.wav");
        }
        rage();
        vel.angular.z = angular;
        vel.linear.x = linear;
        vel_pub.publish(vel);
```

```
        }else if(world_state == 8){
            //right bumper has been kicked and cat gets surprised
            //sc.playWave(path_to_sounds + "surprise.wav");
            if (!ranOnce) {
                sc.playWave(path_to_sounds + "surprise.wav");
                ranOnce=true;
            }
            rightBumper();
            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();
    }
    return 0;
}


void excited(){
//robot turns in a circle in excitement and the excited cat image is displayed
    if(get_time_elapsed()<10000) {
        Mat excited=imread(ros::package::getPath("mie443_contest3")+"/imgs/excitedCat.jpeg"); //compiles but does not work
        imshow_emotion(excited);
        linear = 0.5;
        angular = 1;
        return;
    }

cv::destroyAllWindows();
world_state=0;
return;

}

bool matchImage(Mat inp_frame){
    detector -> detectAndCompute( inp_frame, noArray(), keypoints_img, descriptor_img );

    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create(DescriptorMatcher::FLANNBASED);
    std::vector< std::vector<DMatch> > knn_matches;
    matcher->knnMatch( descriptor_ref, descriptor_img, knn_matches, 2 );

    std::vector<DMatch> good_matches;
    for (size_t i = 0; i < knn_matches.size(); i++)
    {
        if (knn_matches[i][0].distance < ratio_thresh * knn_matches[i][1].distance)
        {
            good_matches.push_back(knn_matches[i][0]);
        }
    }
    //ROS_INFO("Good matches: %lu", good_matches.size());

    Mat img_matches;
    drawMatches( img_ref, keypoints_ref, inp_frame, keypoints_img, good_matches,img_matches, Scalar::all(-1),
    Scalar::all(-1), std::vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
    imshow_fast(img_matches);

    if (good_matches.size()>80) return true;
    else return false;
}

void set_start_time(){
    state_start = std::chrono::system_clock::now();
    return;
}
uint64_t get_time_elapsed(){
    uint64_t mil_elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now()-state_start).count();
    return mil_elapsed;
}
```

```
void rightBumper() {
    //the rightBumper function turns the robot right by 90 degrees if the right bumper is hit
    //make a surprised cat sound
    //Return to its original position
    //returns state 0 to move forward
    if(get_time_elapsed()<=1500){
        //ROS_INFO("in right bumper");
        Mat surpriseRight=imread(ros::package::getPath("mie443_contest3")+"/imgs/surprisedCat.jpeg"); //compiles but does not work
        imshow_emotion(surpriseRight);
        angular=-1.4;
        linear=-0.2;
        return;
    }
    if(get_time_elapsed() > 1500 && get_time_elapsed() <= 4000) {
        angular =0;
        linear = 0;
    }
    /*if(get_time_elapsed()>3000 && get_time_elapsed()<6000 ){
        angular=0.8;
        linear=0;
        return;

    }*/else {
        cv::destroyAllWindows();
        world_state=0;
        state_lockout = false;
        return;
    }
}

//BUMPER
void leftBumper() {
    //the rightBumper function turns the robot left by 90 degrees if the right bumper is hit
    //make a surprised cat sound
    //Return to its original position
    //returns state 0 to move forward
    if(get_time_elapsed()<=1500){
        Mat surpriseLeft=imread(ros::package::getPath("mie443_contest3")+"/imgs/surprisedCat.jpeg"); //compiles but does not work
        imshow_emotion(surpriseLeft);
        angular=1.4;
        linear=-0.2;
        return;
    }
    if(get_time_elapsed() > 1500 && get_time_elapsed() <= 4000) {
        angular =0;
        linear = 0;
    }
    /*if(get_time_elapsed()>3000 && get_time_elapsed()<=6000 ){
        // Mat img_test_sur=imread(ros::package::getPath("mie443_contest3")+"/imgs/meow_surprised_yoga.jpg");
        // imshow_emotion(img_test_sur);
        angular=0.0;
        linear=0;
        return;
    }
    if(get_time_elapsed()>6000 && get_time_elapsed()<9000 ){
        angular=-0.8;
        linear=0;
        return;
    }*/else {
        cv::destroyAllWindows();
        world_state=0;
        state_lockout = false;
        return;
    }
}
```

```
void rage() {

    //function definition for when bot cannot reach user, triggered by obstacle preventing bot from following


    //ROS_INFO("in the rage emotion");
    // reverse for 2 seconds

    if (get_time_elapsed() >= 0 && get_time_elapsed() <= 2500){
        linear  = -0.25;
        angular = 0.0;
        return;

        // move forward for 2 seconds
    }

    if (get_time_elapsed() > 2500 && get_time_elapsed() <= 4500){ // moving forwards back to obstacle
        linear = 0.25;
        angular = 0.0;
        return;

    // reverse  faster to build frustration
    }

    if (get_time_elapsed() > 4500 && get_time_elapsed() <= 6500){
        linear  = -0.3;
        angular = 0.0;
        //ros::Duration(500).sleep();
        return;

    }

    if (get_time_elapsed()> 6500 && get_time_elapsed() <= 7000){
        linear = 0.0;
        angular = 0.0;
        return;

        // move forward for 2 seconds
    }

    if (get_time_elapsed() > 7000 && get_time_elapsed() <= 8500){ // moving forwards back to obstacle
        linear = 0.4;
        angular = 0.0;
        return;

    }

    if (get_time_elapsed() > 8500 && get_time_elapsed() <=10700) {
        // scratching image
        Mat rageOne=imread(ros::package::getPath("mie443_contest3")+"/imgs/scratchCat.jpeg");
        imshow_emotion(rageOne);

        linear = 0;
        angular = 0;
        return;

    }

    if (get_time_elapsed() > 10700 && get_time_elapsed() <= 30000){
        // rage image #2
        //cv::destroyAllWindows();
        Mat rageTwo=imread(ros::package::getPath("mie443_contest3")+"/imgs/catRage.jpeg");
        imshow_emotion(rageTwo);

        linear = 0.0;
        angular = 1.7;
        return;
    }

    else {
        linear = 0;
        angular = 0;
        cv::destroyAllWindows();
        world_state=0;
        return; // end function
    }
}
```

```
void fear() {
    // this emotional state is triggered when the robot loses track of the person it's following
    //normal picture up;

    //the scared image is put up
    Mat fear=imread(ros::package::getPath("mie443_contest3")+"/imgs/catFear.jpeg");
    imshow_emotion(fear);

    //look left
    if(get_time_elapsed()<=3500){
        angular=0.8;
        linear=0;
        return;
    }
    //look right
    if(get_time_elapsed()>3500 && get_time_elapsed()<=9000){
        angular=-0.8;
        linear=0;
        return;
    }
    //turn back to center
    if(get_time_elapsed()>9000 && get_time_elapsed()<=12000){
        angular=0.8;
        linear=0;
        return;
    }
    if(get_time_elapsed()>12000 && get_time_elapsed()<=13000){
        angular = 0;
        linear = 0;
        return;
    }

    //robot starts shaking in fear
    if (get_time_elapsed()==13000) ROS_INFO("Start shaking");

    if (get_time_elapsed() > 13000 && get_time_elapsed()%1000<500){
        angular = 0.3;
        linear = 0.1;
        return;
    }
    if (get_time_elapsed() > 13000 && get_time_elapsed()%1000>=500){
        angular = -0.3;
        linear = -0.1;
        return;
    }
    if (get_time_elapsed()>20000){
        cv::destroyAllWindows();
        world_state=0;
        return;
    }

}
```