**Project 4 – Fall Detection**

**SOFTWARE DESIGN DOCUMENT**

|  |  |
| --- | --- |
| **Gavin Burris** | [**gbb123@live.missouristate.edu**](mailto:gbb123@live.missouristate.edu) |
| **Logan Koch** | [**koch170@live.missouristate.edu**](mailto:koch170@live.missouristate.edu) |
| **Logan Kilfoy** | [**lck1699@live.missouirstate.edu**](mailto:lck1699@live.missouirstate.edu) |
| **Kyle Lepine** | [**kml2008@live.missouristate.edu**](mailto:kml2008@live.missouristate.edu) |
| **Mason Sanborn** | [**sa99@live.missouristate.edu**](mailto:sa99@live.missouristate.edu) |

**1.0 Introduction**

The purpose of this document is to provide an insight into the structure of our software, go in-depth on the inner workings of the software, touch on each component of the software, and look at how we approached designing the software. This project’s primary focus will be to create a system that receives video input and accurately detects falls using data from that video feed. This project is useful for individuals who are prone to falls and require in-home fall detection. This document is intended as a reference point for the development team to use while designing and developing the software.

**1.1 Goals and objectives**

For this project, our team has been tasked with implementing and expanding on our stakeholders, Chaudhari Snigdha, vision for a more consistent and reliable fall detection system. We divided our project up into seven realistic goals that fall in line with Chaudhari Snigdha’s vision.

Our first goal was to take a video and parse it into frames to allow the software to analyze the frames. Second, we want our system to connect to a database that would hold our templates. Third, we want our system to be able to identify the human form in any position. Fourth, we want our system to track our subjects as they move from position to position and between various lighting situations. Fifth, we want the system to identify the difference between a person and anything else in view. Sixth, we want the system to determine when a person falls on camera and recognize the difference between falling and sitting or lying down. Finally, we want our system to identify a fall when a person is partially obstructed.

**1.2 Statement of scope**

This software is designed to be a fall detection security system for elderly or disabled people. It will act as a last resort measure for people who have fallen and can’t get up. To make this software, we set some requirements at the beginning of the semester that we needed to implement to say that the project is complete confidently.

1. **Essential Requirements**
2. Read input data from a video feed.
3. Read input data from our database.
4. Take data from the database and sort templates into arrays based on distance and position.
5. Process each frame individually to identify people in the frame.
6. Output a bounding box to the screen with a distance calculation on each person in the frame.
7. Process each frame to remove all background noise.
8. Compare each person in the frame to a set of templates to identify their current position.
9. Using data from previous frames, determine if each person goes to sit or lay down of their own will or not.
10. Output a comparison percentage and position to the console for each person in the frame.
11. Using input data for each person, detect if a person in the frame is falling, including situations where they are partially obstructed.
12. Output response when a person falls.
13. **Desirable Requirements**
14. Allow the user to input commands to the system using a robust Graphical User Interface.
15. **Future Requirements**
16. Allow the user to input emergency contacts in the event of a fall.
17. Send falling output data to a smart home device, which could then call for help.

**1.3 Software context**

The product that we will be producing will consistently and accurately detect when a person falls utilizing machine learning and templating techniques. Using these techniques, the software will go through a training phase by learning what to look for and differentiate a person who is acting normally versus a person who has fallen. Pre-made templates will help with fall detection by comparing the subject to a template database.

Other fall detection products that are already in use can be divided into two categories, sensor-based detection and vision-based detection. With our machine learning algorithm, we hope to follow more along the side of vision-based detection products by creating a program that works efficiently and accurately to detect and respond to people falling.

**1.4 Major constraints**

The constraints for this software include the following:

1. The system must support the Windows operating system.
2. The system must use an RGB USB web camera to obtain the necessary data to determine falls.
3. The system will detect humans by utilizing machine learning.
4. The system will detect falls utilizing templating techniques.
5. The system will use Python and its libraries to operate.
6. The system will use PostgreSQL to manage a database of templates.

**2.0 Data design**

**Dictionary**: Consists of key: value pairs. All the keys in a dictionary must be unique and immutable. Dictionaries are unordered and implemented in Python as an associative array, a subcategory of hash tables in which each element points to another object. Python’s Dictionary data structure has notable performance benefits, such as constant lookup time complexity, constant getting and setting time complexity, and linear iteration time complexity on average.

**Numpy Ndarray**: A multidimensional, homogeneous vectorized array consisting of items of the same size. The Numpy Ndarrays used in the system consist of elements of data type unsigned 8-bit integers.

**Max Heap**: Otherwise known as a priority queue, a max heap data structure maintains an ordered list of elements. Although a max heap is implemented as a list, it is maintained as a binary tree. Because of this, a max heap has linear time complexity when accessing the maximum valued element in the list and log base 2 of n time complexity when extracting the max or inserting a new value into the list. The max heap implemented in our system consists of tuples of two elements: distance measure, a float, and a classification, a string.

**TemplateDatabase**: A singleton implemented by the system. TemplateDatabase contains attributes that represent the type and characteristic of templates that can be stored in the database, each of which are lists of strings; these are intended to be used when creating the template dictionary. It also holds the database name and password, which are strings; these are used to construct the TemplateDatabase object. A connection to the database is created, which is an instance of the class connection from the Psycopg2 library. The TemplateDatabase class has methods to connect and disconnect to the database, check if the database is currently connected, retrieve the current database version, add and delete templates to and from the connected database by their ID, access the byte array image stored in the database by its ID, access all byte array images by their corresponding template type and matching characteristic, retrieve a list of all current IDs stored in the database, and a method to return a template dictionary with all the images stored in the database. The template dictionary details are discussed further in section 2.2.

**TemplateViewer**: A singleton implemented by the system. This class is used to crop and prune templates stored in the database with a visual aspect aiding its features. TemplateModifier contains attributes that consist of a dictionary of templates, specified in section 2.2, a 3-Dimensional Numpy Ndarray--a list of 2-Dimensional Ndarrays that represent binary images. Their elements are of unsigned 8-bit integer data type. It stores the length of the list of images, which is stored as an integer.

Additionally, it stores the current image being viewed in the list of images, which is a Numpy Ndarray specified in section 2.0. It also maintains the current index of the image being viewed in the list of images as an integer. It records an array containing tuples of the reference points or x-y coordinates, which are integers. Lastly, it maintains a boolean value that describes whether the cropping feature is being used or not and the type and characteristic of the template being viewed. The TemplateModifier has methods to crop the template and to update its attributes when iterating through the different templates.

**DistanceAndClass**: A class used in the KnnClassifier subcomponent of the Human State Classifier component. It has member attributes for the distance, or similarity, and measure, which is saved as an integer. It also maintains an attribute for the corresponding classification to the distance measure; this is saved as a string. The built-in comparison operator should be overridden to compare only the saved distance values of the object. This is a requirement for its usability in the max heap data structure.

**KNeighborsClassifier**: A class implemented in the KnnClassifier subcomponent. Its member attributes consist of a training dataset, which is a dictionary with keys as the template types: “upright”, “falling”, “lying”, and “sitting”. Each key is a string; the matching values are a 3-Dimensional Numpy Ndarray, which represents a list of images. Each element in the Numpy Ndarray is of data type unsigned 8-bit integer and represents a binary pixel. Its other data member is “k”, which is an integer. Both of these are needed in the construction of the KNeighborsClassifier class. There are class methods to set the training dataset and k value. There is a method to classify a single testing item, which is a 2-Dimensional Numpy Ndarray of unsigned 8-bit integers, a binary image, to one of the four specified human states using the K Nearest Neighbors algorithm. Lastly, there is a method to calculate the Euclidean Distance between two Numpy arrays of the same shape. The Euclidean Distance method returns a float.

**ImageManipulator**: The ImageManipulator class has attributes for the source image, which is intended to be an RGB frame of the live video with data type 3-Dimensional Numpy Ndarray of unsigned 8-bit integers. It maintains a detection frame, which is a copy of the source image. It contains a gray image, which is a grayscale conversion of the source image. It contains a foreground image, which is the extracted foreground of the source image. The class also contains an instance of the BoundingBox class that represents the coordinates of the image that movement was detected in; its data members are specified in section 2.0.

If movement is detected in the frame and is instanced, a movement detected boolean is set to True, and if there is no movement detected in the frame, it is set to False. The ImageManipulator class has one get method to retrieve the movement detected boolean. It also has a method to convert an image to a grayscale frame with bilateral filtering applied to the image. The class has methods to perform foreground detection on a source image and a method to perform edge detection on a source image. There are methods to detect the movement in a source image and a method to draw the bounding box detected in the frame on the detection frame. Lastly, there is a method to display the frames in a window.

**Bounding Box**: The Bounding Box class has six data members that contain information in regard to the size of a bounding box. It has data members x1 and x2 that represent the left and right x coordinates respectively and are integers. Y1 and y2 that represent top and bottom y coordinates respectively and are integers. Bounding Box also has attributes for width and height of the bounding box that are stored as integers. The only methods of the bounding box are get methods for the x-coordinates, y-coordinates, width, and height. The get x-coordinates and get y-coordinates return both x’s and y’s as a tuple. There are also methods to change dimensions and retrieve the center of the bounding box. The center of the bounding box is calculated by taking the difference between the x2 and x1 coordinate points and adding it to the x1 coordinate, then determining the same for the y coordinates and returning the newly calculated x and y coordinates as tuple. Change dimensions takes a width and height, which are both integers, and using the center of the current bounding box, and add half of the width to the center x to get x2, subtracts half of the width to get x1, subtracts half of the height to get y1 and adds half of the height to get y2.

**FrameInfo:** The FrameInfo class contains information regarding a frames edge and foreground classification (determined by the HumanStateClassifier component). Each are classification are saved as strings.

**FrameHistory**: FrameHistory objects are created with two integers, one for the number of BoundingBox objects to be saved from previous frames, and one to determine the number of FrameInfo objects to be saved. It has 4 attributes: an integer for each value corresponding with the arguments to create the FrameHistory object, and two empty numpy arrays, bounding\_boxes and frame\_classifications to hold their respective objects. It has a method to average the dimensions of the width and height of the BoundingBox objects saved in the bounding\_boxes numpy array. It has a method to determine if the bounding boxes experience a continuous decrease, meaning that in order from the oldest recorded BoundingBox, to the most recent, whether or not the area of each bounding box has been continuously and strictly decreasing. A method for averaging the area of the bounding boxes in the bounding\_boxes array. Methods for adding and forgetting bounding boxes in the bounding\_boxes array, added boxes will be at the end of the array, forgotten bounding boxes will be removed from the beginning of the array. Lastly there is a method to check if the bounding\_boxes array is at maximum capacity, in other words, the number of elements equals the bounding\_box\_save\_count member attribute. These same methods are implemented for the frame\_classifications array as add\_frame\_info, forget\_frame\_info, and frame\_info\_full methods.

**2.1 Internal software data structure**

1. Extracted Edges is a 2-Dimensional Numpy Ndarray of dtype uint8 (unsigned 8-bit integers). It stores the binary representation of the current live frame retrieved from the web camera that has been applied with the Canny edge detection algorithm. It is generated in the ComputerVision component of the system and delivered to the HumanStateClassification component of the system.
2. Extracted Foreground is a 2-Dimensional Numpy Ndarray of dtype uint8 (unsigned 8-bit integers). It stores the binary representation of the current live frame retrieved from the web camera that has been applied with the foreground detection algorithm and image smoothing algorithm. It is generated in the computer vision component of the system and delivered to the HumanStateClassification component of the system.

**2.2 Global data structure**

1. **Templates**: an instance of a dictionary that contains a nested dictionary of a list of Numpy Ndarrays. It is initialized in the database functionality component of the system and sent to the main component of the system, which manages the events of the system. Based on user interaction, Templates can be passed to the frame classification component and the template visualization component.
2. **EdgeClassifier**: an instance of the KnnClassifier that is constructed with a subset of the templates in the system that have an edge characteristic.
3. **ForegroundClassifier**: an instance of the KnnClassifier that is constructed with a subset of the templates in the system that have a foreground characteristic.
4. **Database**: an instance of the TemplateDatabase that is used for retrieving templates from the database and also modifying the database through the system.

**2.3 Temporary data structure**

1. **Contours:** an instance of a list that contains Numpy Ndarrays. The list of contours is created using the cv2.findContours function on the grayscale frame of the video feed. The largest contour in the list is then determined and used to define the main focus of the video frame for the bounding box functionality.
2. **BoundingBox**: an instance of the BoundingBox class that only exists in the ImageManipulator object; it is deleted when the ImageManipulator object is deleted.
3. **CurrentFrame:** an instance of the ImageManipulator class that is constructed with the current frame of the live video feed. It only exists in one frame of the live video feed and is overwritten each frame.

­

1. **DistanceAndClass:** an instance of the DistanceAndClass class that is constructed in the body of the loop inside KnnClassifier’s classify method and overwritten for every iteration of the loop.
2. **FrameInfo:** an instance of FrameInfo is saved and maintained in an array of other FrameInfo objects representing the classifications of the previous 15 frames.

**2.4 Database description**

There is only one database used in the Fall Detection System, the FDSDatabase. FDSDatabase is written in PostgreSQL and has only one relation. In this relation, templates are stored. The relation has attributes: Template\_ID, template\_type, template\_characteristic, image\_name, image. The template ID is a serial number and primary key of this relation--it does not need to be specified when adding an entry to the relation. Template\_type is a varchar(10) that contains information on the state of the human in the template. The possible states that can be used to describe the human are: “upright”, “falling”, “sitting” and “lying”.

Template type must not be null and is constrained to the possible states listed prior. Template characteristic is also a varchar(10) and is limited to two possible entries: “edge” and “foreground”. These describe what characteristics of the original image the template is derived from. This field should not be null and must be constrained to these two possibilities. The image name is the relative name to each template entry; its data type is varchar(50). Lastly, the image field is of type bytea. All images in the relation are intended to be unique and not null.

**3.0 Architectural and component-level design**

The Fall Detection (FD) system implements a Layered Architectural design. With this design, the program is able to be divided into separate functional layers. The FD system is separated into three distinct layers, Presentation, Persistence and Business, and Database. The Presentation layer is able to communicate with the Database layer directly; therefore, the Persistence and Database layer is an open layer.

The Presentation layer contains classes and components responsible for presenting the User Interface (UI) to the end-user. The Presentation layer will be able to directly communicate with the Database and Persistence and Business layers.

The Persistence and Business layer contains components and logic responsible for determining what source the system will use as video input, generating computer vision, and classifying images with human state classification. The Persistence and Business layer can only communicate with the Database layer.

The Database (DB) layer is responsible for interacting with the system’s template database. The Database layer can communicate with the Persistence and Business layer.

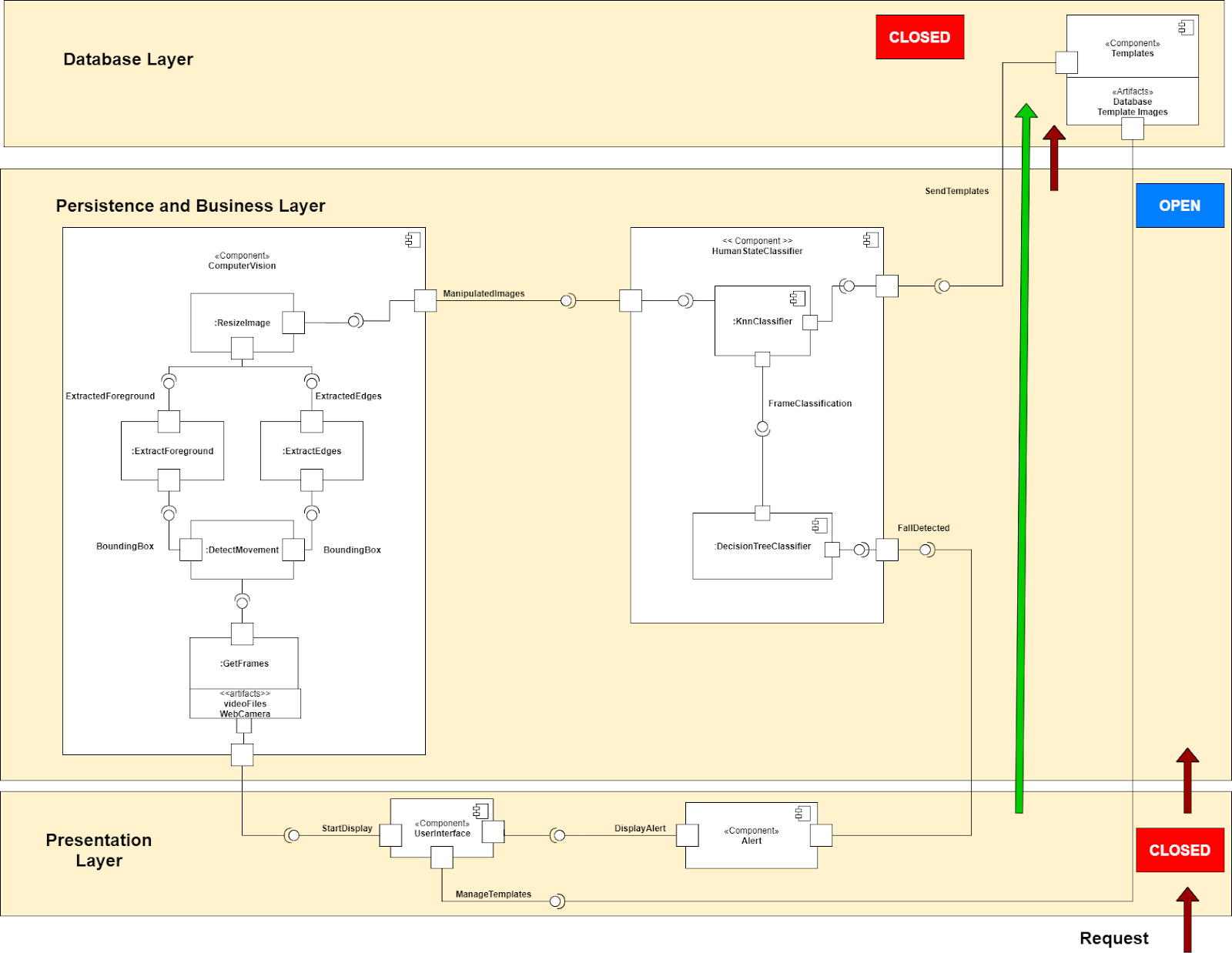
**3.1 System Structure**

The Fall Detection (FD) system implements the Chain of Responsibilities design pattern. This pattern accepts a request from a client and directs the request to a handler. The handler sends the request to a segmented chain of processes until the request has been completed. The chaining mechanism allows for an unbounded amount of processes to be chained together; Allowing the system to become more dynamic and adaptable.

The request will come from the Presentation layer, which will direct the request to the Persistence and Business (PB) layer. The user interface is the design pattern’s handler. The PB layer contains the pattern’s process chain. The PB layer will handle and complete the request processes and request.

The Presentation layer, User Interface Component, would start the request Process Chain, sending a request to the Computer Vision component to perform image manipulation required for human state classification, the next step in the chain. The Human state classification process would communicate with its dependency, the Template component, to complete the request from the Computer Vision handler and then pass its result to the Alert handler. The Alert handler would finish the chain and return to the User Interface handler.

**3.1.1 Architecture diagram**



**3.2 Description for Component n**

1. **Computer Vision**

The Computer Vision component handles all of the image processing jobs for our program.

1. **Human State Classifier**

The Human State Classifier component is responsible for handling the logic when determining a fall in the current frame of the live video feed.

1. **Templates**

The Templates component handles all jobs relating to our database, including writing to and retrieving templates.

1. **Alert**

The Alert component handles the sole task of alerting the user to any falls that happen on screen.

1. **User Interface**

The User Interface component invokes interactions between other components. It handles initiating the software.

**3.2.1 Processing narrative for component n**

1. **Computer Vision**

The Computer Vision component is responsible for everything relating to the actual video feed and how it is processed. Computer Vision is able to take data from the video feed and detect motion within it by comparing it to the previous frame. From there, the component will extract the foreground and edges from the moving objects within the frame. Next, it resizes the extracted elements to a size that can be used for comparison. Finally, it sends that information to the Human State Classifier.

1. **Human State Classifier**

The Human State Classifier component is responsible for classifying each person in the frame. These classifier positions include upright, sitting, falling, and lying down. The Human State Classifier takes two inputs. First is the templates from our database, and the second is data from the current frame. After receiving the inputs, they are sent through the KNN classifier that uses our templates for comparison with the current frame. From there, information is sent to a decision tree that makes the choice of if there is a fall to be reported or not. If there is a fall, a message is sent to the Alert component to be used.

1. **Templates**

The Templates component is related to all operations involving template retrieval from our database. This component sends our templates to the Human State Classifier for comparison with the current frame.

1. **Alert**

The Alert component is responsible for throwing a flag when a fall has been detected. It takes input from the Human State Classifier and sends the appropriate output to the User Interface.

1. **User Interface**

The User Interface component is the component in charge of providing the user with the information that is relevant to them. This includes such things as general menu options like starting and stopping the program as well as outputting critical information such as a fall detected alert from the Alert component.

**3.2.2 Component n interface description.**

1. **Computer Vision**
2. The Computer Vision component provides an interface to the Human State Classifier component. It will send the processed frames to the Human State Classifier to be classified.
3. The Computer Vision component requires interface from the User Interface Component as it needs a starting signal as an input to begin capturing frames using the camera.
4. **Human State Classifier**
5. The Human State Classifier component requires an interface from the Computer Vision component in order to begin classifying images. The Computer Vision component will send manipulated images to the KnnClassifier subcomponent of the Human State Classifier to begin classification. The manipulated images are the extracted edges and extracted foreground mentioned in section 2.1
6. The Human State Classifier component requires an interface from the Templates component before any classifications can be performed. The interface provides the Human State Classifier component a dictionary of templates (mentioned in section 2.2), which serves as the training model for the KnnClassifier subcomponent.
7. **Templates**
8. The Templates component provides an interface to the Human State Classifier component. It sends templates from the database to the Human State Classifier so that the templates can be used in the classification process. If the system cannot connect to the database, the Templates will instead pass the templates found locally.
9. The Templates component requires an interface from the User Interface component. The interface is necessary for the addition or manipulation of templates within the database.
10. **Alert**
11. The Alert component requires an interface from the Human State Classifier component. It requires a boolean value that is used to confirm whether or not a fall was detected in the images classified inside the Human State Classifier component.
12. The Alert component provides an interface to the User Interface component, which communicates whether or not the User Interface component should display an alert to the user.
13. **User Interface**
14. The User Interface component provides an interface to invoke the ComputerVision component. It will communicate either a premade folder which contains a video file, a folder that contains an array of images that are intended to be read as frames of a video or to invoke the ComputerVision component to start reading live video frames from a connected web camera.
15. The User Interface component also provides an interface to the Templates component. This interface permits the user to have direct interaction with the database. This allows a user of the system to upload templates or delete templates from the database.
16. The User Interface component requires an interface from the Alert component. When this interface is invoked, the User Interface component will communicate to the user a fall was detected and print a message to the console.

**3.2.3 Component n processing detail**

1. **Computer Vision**

Get Frames

* Captures Individual Video Frames from the Web Camera.

Detect Movement

* Convert captured video images to grayscale and focus on that region.

Extract Foreground

* Splice the Foreground frame from the focused area.

Extract Edges

* Splice the Edge frame from the focused area.

Resize Image

* Resize the spliced frames to the proper size.

Continuation

* Pass Manipulated Images to the Human State Classifier

1. **Human State Classifier**

Knn Classifier

* Receive Manipulated Images and classify them to database templates for similarity.

Decision Tree Classifier

* Take frame classification, process results. If fall detected, send results to the Alert component, else return to Computer Vision (a).

1. **Templates**

Store Templates

* Template database stores data for all saved templates in various positions.

Send Templates

* Sends the stored templates to the Human State Classifier for comparison.

1. **Alert**

Receive

* Receives Fall Detected Signal from Human State Classifier.

Process

* Responds to the Fall Signal as necessary for the system.

1. **User Interface**

Initialization

* Appears on system startup for the user to choose functionality.

Processing

* System processes user response.

Start Display

* Sends signal to Computer Vision to begin processing data.

Manage Templates

* Allows users to interact with templates stored in the Database.

**3.2.3.1 Design Class hierarchy for component n**

1. **Computer Vision**

The ImageManipulator uses the bounding box class. Neither inherits from another class.

1. **Human State Classifier**

The KNeighborsClassifier uses the DistanceAndClass class. Neither inherits from another class.

1. **Templates**

The TemplateDatabase class does not inherit from another class.

1. **Alert**

The Alert component does not contain any classes.

1. **User Interface**

The User Interface component uses the Template Modifier, Templates, Image Manipulator. None of the classes inherit from another class.

**3.2.3.2 Restrictions/limitations for component n**

1. **Computer Vision**
2. The Web Camera must be able to create an accurate video feed of the environment.
3. **Human State Classifier**
4. There must be accurate templates for the Manipulated Images to be compared to.
5. The more templates that Manipulated Images are compared to, the longer the processing will take.
6. **Templates**
7. Templates must be stored where the system can access them.
8. Templates of different types must be stored separately.
9. **Alert**
10. N/A
11. **User Interface**
12. The user must use the command line to interact with the current system

**3.2.3.3 Performance issues for component n**

1. **Computer Vision**
2. A higher frame rate camera would create more frames for the system to process.
3. **Human State Classifier**
4. The more templates that Manipulated Images are compared to, the longer the processing will take.
5. **Templates**
6. Templates are not exact in their creation. They will not have perfect edges or, in some cases, completely filled in bodies. This can lead to less accurate comparisons.
7. **Alert**
8. False positives can lead to inaccurate alerts.
9. **User Interface**
10. N/A

**3.2.3.4 Design constraints for component n**

1. **Computer Vision**
2. Converting images to manipulated images must not overload the system running the program.
3. **Human State Classifier**
4. The human state classifier will depend on multiple methods and variables to determine the human state classification.
5. Classify edge and foreground image with human state classification within 20 milliseconds.
6. **Templates**
7. Provide bulk template records efficiently so that the system maintains its current state.
8. **Alert**
9. N/A
10. **User Interface**
11. Provide the user with direct access via the Command Line Interface (CLI).
    * + 1. **Processing detail for each operation of component n**

**ComputerVision**

**Get Frames**

The Get Frames operation uses the OpenCV library to capture frames from the system’s camera.

**Detect Movement**

The Detect Movement operation uses OpenCV’s createBackgroundSubtractorMOG2 algorithm, with parameters (history=200, detectShadows=False), to subtract pixel values that are the same between frames (images).

**Extract Edges**

The Extract Edges operation uses OpenCV’s Canny algorithm to detect edges in an image.

**Extract Foreground**

The Extract Foreground operation uses OpenCV’s morphologyEx algorithm to convert the foreground into “computer vision”.

**Resize Image**

The Resize Image operation uses OpenCV’s resize algorithm to resize images to a standard size defined by the system.

**HumanStateClassifier**

**KnnClassifier**

The KnnClassifier, specified further in section 2.0, has the main purpose of classifying a given binary image as a human state. The human states specified for this system are: “upright”, “falling”, “sitting”, and “lying”. The approach our KnnClassifier uses to classify an image to one of these states is through the implementation of the famous K Nearest Neighbors algorithm. The K Nearest Neighbors algorithm takes a testing tuple--in our case, a binary image--and a training dataset, which is a subset of our Templates data structure mentioned in section 2.2. The K Nearest Neighbors algorithm iterates through every element in the training dataset and calculates a distance measurement between the training tuple and testing tuple. If the length of the neighbors, our max heap mentioned in section 2.0, is less than the specified k value, then the distance and class are stored as an instance of the DistanceAndClass class, specified in section 2.0, and added to the max heap. If neighbors has more elements than the k value, then the distance measurement calculated is compared with the largest (least similar) distance measurement associated with one of the neighbors. The neighbor is then replaced with the value less than it.

The K Nearest Neighbor algorithm can be used to classify data, and, in this scenario, it is used to classify binary images from a live video feed. In the implementation of the K Nearest Neighbors algorithm, our system utilizes the Euclidean distance measurement in calculating the distance, or similarity, measurement. When classifying an entry, the KNN implementation also uses weighted voting, which counts votes as the reciprocal of its similarity measurement previously calculated. In addition, to generate an appropriate value for k, our algorithm uses the square root of the total number of entries in the training dataset of quotient three if one is not provided.

The data structure best suited to handle the needs of finding the max value of a list and extracting it would be a max heap (also known as a priority queue). This allows the system to find the max value in the list in O(1) time complexity and extract it in O(lg(n)) time.

The purpose of the DistanceAndClass class is to allow our max heap to store both the distance measurement and the classification in association with the image in the training data and still be ordered by distance. If this precaution was not taken, the classification would have been taken into consideration when ordering the max heap, which is not useful in this implementation.

**DecisionTreeClassifier**

In the DecisionTreeClassifier, our system implements the ID3 Decision Tree Induction Algorithm. The ID3 Decision Tree Induction is a greedy divide-and-conquer algorithm. It constructs decision Trees in a top-down recursive manner. Initially, all training data is at the root of the tree and is partitioned recursively based on attributes with the highest information gain.

The algorithm to generate the decision tree should take training samples and a list of attributes as input. Firstly, the algorithm will create a node N. Secondly the algorithm will check if the given samples are all of the same class C; if that is the case, then the N can be returned as a leaf node labeled with class C. If there are still samples available, then it will check if the attribute list is empty. If that is the case, it will return N as a leaf node with the majority class C. If neither of those is the case, the algorithm will select the attribute from the attribute list with the most information gain. N will be labeled with that attribute, and for every known value of that attribute, the algorithm will grow a branch from N to the value. The algorithm will calculate a subset of samples from the given samples in which the attribute equals that value. If there are no samples, then a leaf node will be attached with the majority class in D (the original set of data); otherwise, the algorithm will recursively call itself with the subset of samples and the updated attribute list.

**Templates**

**Create Templates**

The Create Templates operation uses the psycopg2 library to insert templates into the template database.

**Read Templates**

The Read Templates operation uses the psycopg2 library to read templates from the template database.

**Update Templated**

The Update Templates operation uses the psycopg2 library to update templates in the template database.

**Delete Templates**

The Delete Templates operation uses the psycopg2 library to delete templates from the template database.

**Alert**

**Send Alert**

N/A

**User Interface**

**Display UI**

N/A

**3.2.3.5.1 Processing narrative for each operation**

1. **Computer Vision**

Get Frames

The Get Frames operation is used to get video frames (images) from the system’s camera.

Detect Movement

The Detect Movement operation is used to detect movement in images using the images passed from the Get Frames operation.

Extract Edges

The Extract Edges operation is used to manipulate and extract edges from the images passed from the Detect Movement operation.

Extract Foreground

The Extract Foreground operation is used to determine and extract the foreground from images passed from the Detect Movement operation.

Resize Image

The Resize Image operation is used to resize images from the Extract Edges and Extract Foreground operations to a standard size defined by the system.

1. **Human State Classifier**

KnnClassifier

The KnnClassifier operation is used to predict the human state classification of an image.

DecisionTreeClassifier

The DecisionTreeClassifier operation is used to improve the confidence of the human state classification provided by the KnnClassifier operation.

1. **Templates**

Create Templates

The Create Templates operation is used to insert new templates into the template database.

Read Templates

The Read Templates operation is used to read templates from the template database.

Update Templates

The Update Templates operation is used to update templates in the template database.

Delete Templates

The Delete Templates operation is used to delete templates from the template database.

1. **Alert**

Send Alert

The Send Alert operation is used to send alert messages to the User Interface component.

1. **User Interface**

Display UI

The Display UI operation is used to display the user interface, which consists of multiple options, to the user via the command-line interface (CLI)

**3.2.3.5.2 Algorithmic model for each operation**

**User Interface**

Has constant time and space complexity O(1).

**ComputerVision**

Has linear time and space complexity O(N); N is the number of pixels in an image.

**Templates**

Has linear time and space complexity O(N \* M); N is the number of pixels in an image, and M is the number of templates stored in the database.

**Alert**

Has constant time and space complexity O(1).

**HumanStateClassifier**

**KnnClassifier**

* Has time complexity O(N\*lg(N)) and linear space complexity O(N).

**DecisionTreeClassifier**

* Has time complexity O(N \* M) where N is the number of attributes of a given dataset, and M is the number of options for each corresponding attribute. It has linear space complexity O(N).

**3.3 Dynamic Behavior for Component n**

1. **User Interface**

Template Modifier

The Template Modifier class is used to modify templates that exist in the template database when the user selects the appropriate command.

Templates

The Templates class is used to connect and disconnect from a database. Template’s method and properties are also used to upload and delete templates from the template database.

1. **Computer Vision**

BoundingBox

The Bounding Box class is used by the focus\_movement function described in the Image Manipulator class. It is used to draw bounding boxes around movement.

ImageManipulator

Image Manipulator’s methods and properties are used in the Computer Vision component to see if movement has been detected. If movement has been detected, a bounding box is used to “box” the movement.

1. **Human State Classifier**

DistanceAndClass

The DistanceAndClass class is used by the classify function, described by the KNeighborsClassififer class, to create an instance of DistanceAndClass. This instance will be used to calculate distance and classification for the images provided by the KNeighborsClassifier.

KNeighborsClassifier

The KNeighborsClassifier constructor is used in the system’s main module to create edge and foreground classifiers

1. **Templates**

TemplateDatabase

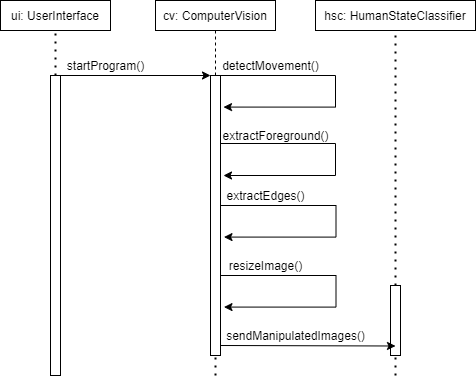
The TemplateDatabase class contains methods and properties used to interact with a template database. Methods that allow the creation, reading, updates, and deletion of templates in the template database.

1. **Alert**

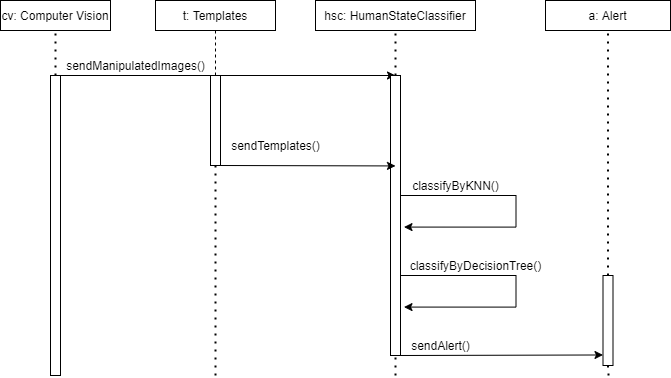
N/A

**3.3.1 Interaction Diagrams**

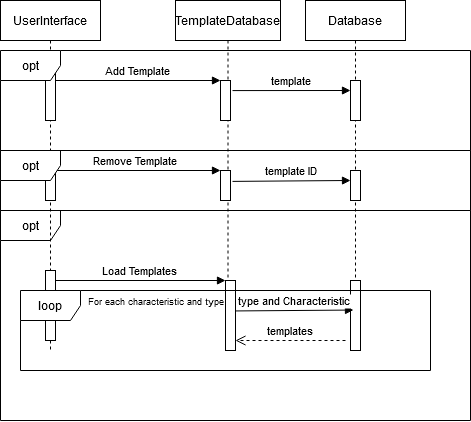
1. Computer Vision



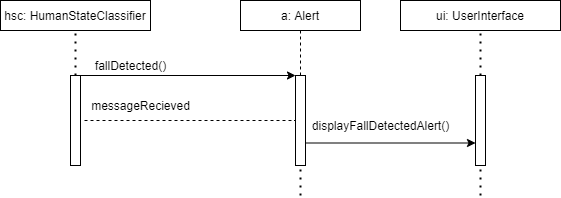
1. Human State Classifier



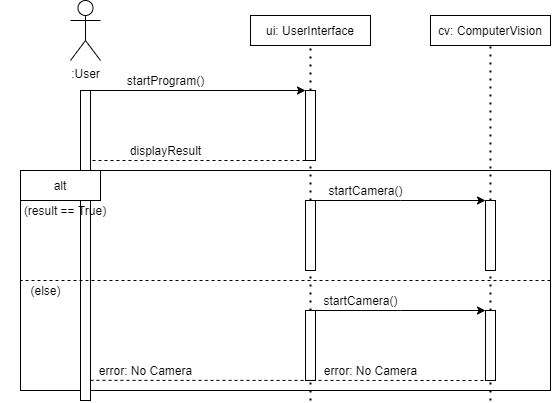
1. Templates



1. Alert



1. User Interface



**4.0 User interface design**

The user interface design for this project will use the command-line interface (CLI).

**4.1 Description of the user interface**

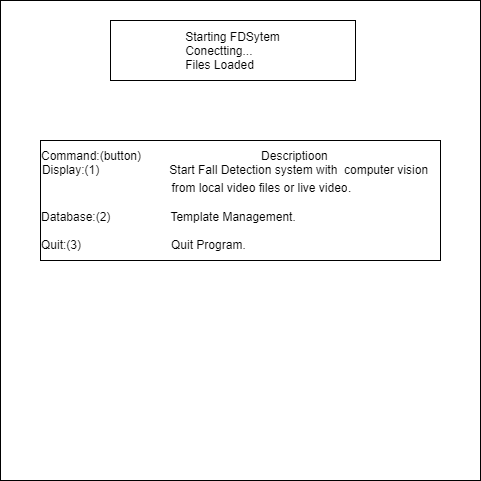
The user is presented with a menu on the terminal and awaits user input. The menu will be two columns and four rows. The first column is the name of the command, and the second column is the description of the command on the corresponding row. The user will have three commands to choose from:

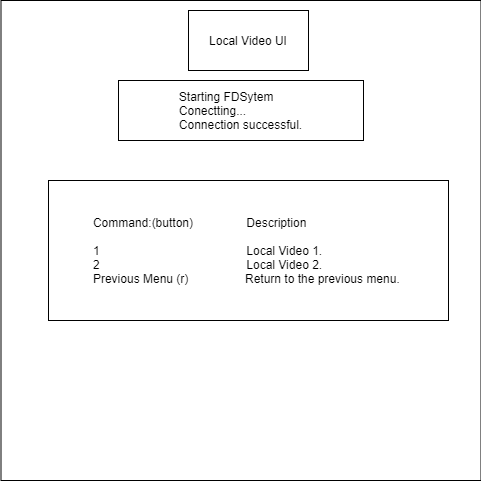
Display :(1) - Start Fall Detection system with computer vision from local video files or live video.

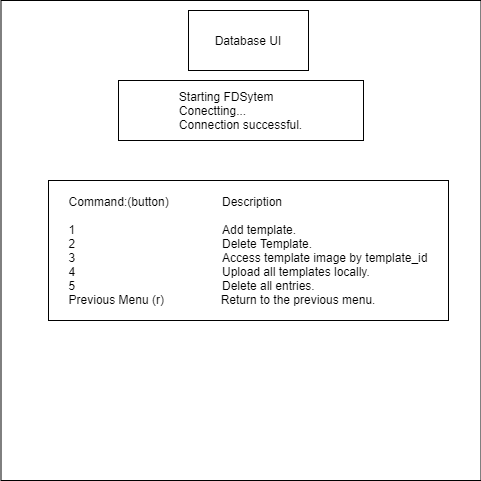
Database:(2) - Template management

Quit:(q) - Quit the program

**4.1.1 Screen images**

**

**

**

**4.1.2 Objects and actions**

1. Starting the program will prompt the user that the system is starting and connecting to the database.
2. If the system cannot connect to the database, the system will inform the user, load local templates, and display the main menu.
3. The first option for the user is “Display(1)”. This option will present a new menu to the user with options to use live video or test video files in local files.
4. The live video option will prompt the user for a session name. This session name is used for creating and organizing custom templates. Leaving the session command blank will start the program for a default session. Once a session has started, the user will be prompted with a new menu to toggle saving templates and to toggle classifications.
5. The user can select the optional video files by entering the corresponding number to their desired video file.
6. The second option for the user is “Database(2)”. The user will be presented with a new menu that allows the user to modify templates in the database.
7. The user will be able to manually label templates as Upright, Falling, Sitting Down, and Lying Down.
8. The final option for the user will be to quit the program.

**4.2 Interface design rules**

The CLI design is used for portability and resource efficiency.

**4.3 Components available**

N/A

**4.4 UIDS description**

N/A

**5.0 Restrictions, limitations, and constraints**

1. Lower-end hardware and processors to make the product more appealing to the average consumer.
2. The need for quick response times lead to the decision to handle the fall detection process on the consumer’s machine and not move towards a client and server-oriented architecture.
3. Fall detection should be performed in fewer than 20 milliseconds.
4. The system should be able to detect falls within 25 feet of the camera.
5. The system should be able to detect falls in an environment with low lighting and obstructed views.

**6.0 Testing Issues**

Test Strategy:

For the Fall Detection system, unit tests are used for creating and testing test cases. In our implementation, we used the python library, unittest, to create unit tests.

Testing Goals:

The goal of the Fall Detection system testing is to ensure that the system can detect falls in multiple types of environments with different conditions as per the functional requirements specified by stakeholders.

Testing Tools:

Our test cases are tested using an automated technique, as in the output can be predicted from the input.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Technique | Input | Test Description | Output/Result |
| 0 - 5 Feet Fall. | Automated | (pre-recorded)Frames that include a human falling within 5 feet of the camera. | Tests if the system is able to detect a fall within 5 feet from the camera. | Boolean: True |
| 5 - 10 Feet Fall | Automated | (pre-recorded)  Frames that include a human falling within 5 - 10 feet of the camera. | Tests if the system is able to detect a fall within 5 - 10 feet from the camera. | Boolean: True |
| 10 - 15 Feet Fall. | Automated | (pre-recorded)  Frames that include a human falling within 10 - 15 feet of the camera. | Tests if the system is able to detect a fall within 10 - 15 feet from the camera. | Boolean: True |
| 15 - 20 Feet Fall | Automated | (pre-recorded)  Frames that include a human falling within 15 - 20 feet of the camera. | Tests if the system is able to detect a fall within 15 - 20 feet from the camera. | Boolean: True |
| 20 - 25 Feet Fall | Automated | (pre-recorded)  Frames that include a human falling within 20 - 25 feet of the camera. | Tests if the system is able to detect a fall within 20 - 25 feet from the camera. | Boolean: True |
| Detect Fall in Low Light | Automated | (pre-recorded)  Frames that include a human falling in low lighting conditions. | Tests if the system can detect falls in an environment with low lighting conditions. | Boolean: True |
| Detect Fall with Obstructed View | Automated | (pre-recorded)  Frames that include a human falling out of frame with a portion of the body still visible to the camera | Tests if the system can detect falls with an obstructed camera view. | Boolean: True |
| Detect Good Fall | Automated | (pre-recorded)  Frames that include a human falling in a room with ample lighting conditions and is within 5 feet from the camera. | Tests if the system can detect a fall in an environment with ample lighting and non-obstructed camera views. | Boolean: True |
| Detect Lying Down | Automated | (pre-recorded)  Frames that include a human lying down in a horizontal position. | Tests if the system can tell the difference between a fall and a human lying down. | Boolean: True |

**6.1 Performance bounds**

1. The system must detect when a person falls within two seconds of the fall occurring.
2. The system must detect when a person falls between a distance of 5 feet and 25 feet inclusively from the viewing lens of the camera.
3. The system must detect when a person falls when the source image’s RGB pixel values have a standard deviation above 10.
4. The system must detect when a person falls when an object obstructs the person by less than 50%.
5. The system must be able to detect more than one person at a time.
6. The system must be able to query all templates from the database in under 15 seconds.

**6.2 Identification of critical components**

1. Our templates are critical to the inner workings of our software, so their creation has to be highly regulated to ensure maximum consistency and speed of the software.
2. Our template comparison method is critical to our software as our program’s speed can only be as fast as the time our comparison method takes to determine the current state of each person in a given frame.
3. Our distance calculations are critical to our software as our software’s templates are divided into arrays based on distance and position. Without these calculations being precise, our template comparison method would be many times slower having to look through all templates.