

Project FRIDA

Jonah Falk, Samuel Pete, Normandy River, Niko Robbins, Jacob Schmoll

July 29th, 2020

Submitted to:

Dr. Razib Iqbal

Professor of Computer Science

Missouri State University

In Fulfillment of the

CSC 450 Course Project

Summer 2020

1.0 Introduction

The Software Design Document (SDD) is a document to provide documentation which will be used to aid in software development by providing the details for how the FRIDA software should be built. Within the SDD are narrative and graphical documentation of the software design for the project, including algorithmic models, a component diagram, a sequence diagram, and other supporting requirement information.

1.1 Goals and Objectives

The purpose of the FRIDA software is to facilitate the detection of falls of people in their residence. Thus, the primary objective of the FRIDA project is to create a viable product for monitoring fall situations, namely for those who are at higher risk for falls, such as elderly and physically disable people.

Accordingly, the final product must be quick, efficient, and accurate in its detection of falls. It must offer simple usage to administrators, developers, and those installing or managing the software without overwhelming them. The user interface must be intuitively used within a system console. Beyond these general design principles, the software must also provide the following concrete functionalities:

- Usage of either a webcam or video file
- Fall and non-fall predictions via utilizing a convolutional neural network (CNN)
- Notification to a computer in the event of a fall
- Video frame(s) that shows video output with an informative heads-up display
- Messages that display software fall, loading, error, success, and termination data

1.2 Statement of Scope

The FRIDA software will take an input of either a live video feed or video file from the user. Once the video input is obtained, the FRIDA software compares the images to a trained model. It will continuously monitor for a human body to detect if it is a fall or non-fall state, and output the corresponding prediction data to the system's console accordingly. If a fall is detected, within 10 seconds the user will be notified via a message to the console.

Body detection, continuous monitoring, sending notifications to the user, and video input recognition are essential requirements.

Desirable and future requirements include detecting a body has fallen with an 80% or higher accuracy utilizing either the regular or condensed space model.

1.3 Software Context

The FRIDA project's ownership directly belongs to Dr. Razib Iqbal, Snighda Chaudhari, Shashi Khanal, and therefore they will ultimately use the software as they see fit. The

development and maintenance costs are nonexistent, so funding will not be an issue. The ends users will run it in within PyCharm's console.

As a large portion of the country becomes increasingly older, it becomes significantly more important to have the ability to track falls in those who are elderly due to its serious health implications. While there likely is software that already exists to do such thing, there has yet to be one made to where one can both affordably obtain it and easily install it on their computer then run it through their aftermarket webcam. Once ran, the software will be able to detect a fall quickly and it will be built to run efficiently as to not utilize too much processing power. It will also have a built-in fall and non-fall prediction analysis as to facilitate seeing the fall data. These functions will allow an administrator of some sort, such as a caretaker, to monitor those who are at risk for falls and to prevent further injury.

1.4 Major Constraints

The greatest constraint for the FRIDA project is time. There are roughly 8 weeks allocated to the development, model training, testing, and documentation of this project. Collectively, the development team has very little to no experience with the OpenCV, ONNX, and PyTorch libraries, nor their included supporting libraries and video tracking software in general, so a significant portion of this time will be dedicated toward learning and implementing the tools. Moreover, if for any reason the OpenCV library, which supports the overall framework for the software, does not suit the project's needs, additional time will need to be dedicated toward finding alternative solutions.

2.0 Data Design

The data design used is an input NumPy array frame which is how OpenCV processes frames. Each frame will then be resized and reshaped to fit into the CNN MobileNetV2 model. The shape expected of the CNN model is (32, 3, 256, 224).

2.1 Internal Software Data Structure

Internal data structures include NumPy array, PyTorch tensor, strings, integers, floats.

2.2 Global Data Structure

The global data structures for FRIDA are implemented in several classes. The LoadModel class loads the ONNX CNN model into the program. The CreateBatch class has attributes corresponding to the batch lists and condensed_batch list. The class CameraSetUpLiveVideo sets up the camera for using live video feed. The class CameraSetUpVideoPlayBack setups the camera to be used for an external .mp4 file to be used as input. The class TransformShape begins the transformation process of making sure the NumPy array is an acceptable input shape for the model to accept. The class MotionHistoryTransform setups the motion history image (MHI). To begin the transformation process, the class MotionHistoryDifferenceMaker sets up the other half of

the MHI to be used to calculate the difference between the previous frame and the current frame. Other global data structures, including those provided by OpenCV library, which are available through the `import cv2`.

2.3 Temporary Data Structure

The FRIDA software uses several temporary NumPy array data structures to hold temporary frames which are used to transform, copy, and create the MHI used as input into the model.

2.4 Database Description

We do not have any database aspects related to the FRIDA project.

3.0 Architectural and Component-Level Design

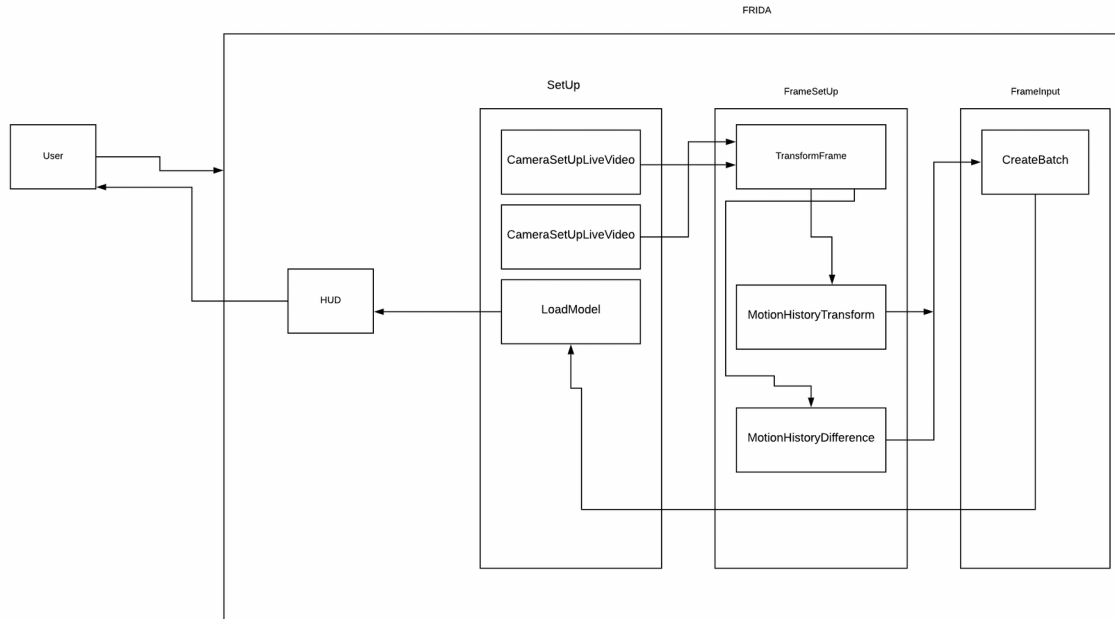
The FRIDA software uses an event-driven architecture, due to the fact that it tracks changes on state and performs different operations based on that state. The main catalyst in our event-driven architecture is each frame as a video is played or as a live video is provided. Upon starting the program, the program begins to import the necessary libraries such as OpenCV, after which the program aims to locate the model in the path provided, load the model, read the model, run the model, and finally set the models input name. After this event has finished, the model then proceeds to either run the component of `CameraSetUpLiveVideo` or `CameraSetUpVideoPlayBack` depending on if a user is connecting to another external webcam or to another external .mp4 file. After the camera setup has finished, the process of the event-driven architecture moves onto setting up the frame to be transformed with the `TransformFrame` class being initialized, then the `MotionHistoryTransform` class, and finally `MotionHistoryDifference` class. Together, the `MotionHistoryTransform` and `MotionHistoryDifference` are used to make an MHI, which is then appended to the list batch inside the class `CreateBatch`. After 32 MHIs have been appended into the batch, the batch is sent as input into the model from the `LoadModel` Class. This cycle represents for as long as the live video feed is on, or as long as the video mp4 file plays, or until the user decides to quit the program.

3.1 System Structure

The structure of the system is comprised into several class components. As noted above in Global Data Structures. Overall, the system structure is as follows. The `LoadModel` class loads the ONNX CNN model into the program. The `CreateBatch` class has attributes corresponding to the batch lists and `condensed_batch` list. The class `CameraSetUpLiveVideo` sets up the camera for using live video feed. The class `CameraSetUpVideoPlayBack` sets up the camera to be used for an external .mp4 file to be used as an input. The class `TransformShape` begins the transformation process of making sure the NumPy array is an acceptable input shape for the model to accept. The class `MotionHistoryTransform` setups the MHI to begin the transformation process. The class

MotionHistoryDifferenceMaker setups the other half of the MHI to be used to calculate the difference between the previous frame and the current frame.

3.1.1 Architecture Diagram



3.2 Description for Components

LoadModel

- It loads the model and acts as the model itself for which input is feed into it so that feedback can be outputted regarding the status of fall versus not fall.

CameraSetLiveVideo

- Acts as the camera setup component by providing all the necessary requirements for the camera to work when using live video feed.

CameraSetUpVideoPlayback

- Acts as the camera setup component by providing all the necessary requirements for the program to work when the user desires to test the model on an external .mp4 file or a provided video dataset file.

CreateBatch

- Responsible for knowing the batch length along with creating the list data structure used to hold each frame up to and including 32.

TransformShape

- Responsible for creating a temporary frame which has a shape like the current frame, of which can be transformed into a type needed to create the MHI.

MotionHistoryTransform

- Starts the Motion History transformation process

MotionHistoryDifference

- Provides the last aspect needed to complete the MHI.

3.2.1 Processing Narrative for Components

Processing Narrative for cv2

cv2 is the interface into the OpenCV library which provides access to many of the necessary methods for the FRIDA program to run. Specifically, the component is used to help setup the camera to be used with live video and to setup the video when using an .mp4 external video file through cv2.VideoCapture. cv2 has the processing responsibility of resizing a given frame to make sure the frame shape is acceptable to what the model accepts through cv2.resize. cv2 is further used to select the method of pixel resampling when the resizing occurs with cv2.INTER_Area. Color change to the pixel from 3-channel to 1-channel RGB is also used through cv2.cvtColor which takes the given frame and the color change require. Another aspect cv2 is responsible for in processing is the reading of the ONNX model after it has been loaded.

Processing Narrative for LoadModel

LoadModel serves as a class object, which is responsible for processing the requirements of finding the model in the given path provided to the model. It does so by loading the model into the programming, turning the model into a readable form to be used by the program, along with running the model to start an inference session, and finally it is needed to define the input name expected by the model to run a given inference session.

Processing Narrative for CameraSetUpLiveVideo

CameraSetUpLiveVideo serves as a class object which is responsible for setting up the camera to be used in a live video feed situation. The goal of the class is to help instantiate a CameraSetUpLiveVideo object, which has set the webcam frames per second needed for the program, the video brightness, along with the camera port required to ensure the video capturing can run smoothly.

Processing Narrative for CameraSetUpVideoPlayBack

CameraSetUpVideoPlayBack serves as a class object which is responsible for setting up the camera to be used in a video playback situation (i.e., playing a video file). The goal of the class is to help instantiate a CameraSetUpVideoPlayBack object, which has set the frames per second needed for the program, the video brightness, along with the path needed to reach the external video file.

Processing Narrative for CreateBatch

CreateBatch is a class responsibility for processing input related to the acquired frames up to the batch_size of 32. The class creates 2 lists to be used depending on which model is run: either Regular Model or Condensed-Spaced Model. These lists hold the transformed frames, which then receive one final transformation to ensure a proper input shape to be used by the model.

Processing Narrative for TransformShape

The goal of TransformShape as a class is to provide the mechanisms which can transform a given frame from the shape OpenCV provides into the shape needed by the model. TransformShape just starts the process by accepting a given frame along with creating a temporary frame, which has a shape like the current frame in the form of a NumPy array.

Processing Narrative for MotionHistoryTransform

The goal of MotionHistoryTransform as a class is to provide the beginning mechanisms to transform a given frame into the start of a motion history image. It has as attributes the width, height, and color channel RGB value expected by the model along with a previous frame holder and mhi_zeros holder which shaped after a given frame.

Processing Narrative for MotionHistoryDifference

The processing of the MotionHistoryDifference class creates the frame which will be used with the frame provided by the MotionHistoryTransform to create the actual MHI. The class accepts a current frame and then provides access to a new frame, which has been resized based on the required parameters of width, height and the interpolation.

3.2.2 Component Interface Descriptions

The attributes associated with each class are able to be accessed when an instance object of the class is created. There are no methods provided by any of the class as the behavior needed by the user does not need to allow the user to make method calls to change values to the class other than what has been provided by the program itself.

3.2.3 Component Processing Details

Processing Details for cv2

- Type: Interface
- Purpose: cv2 is to act as an interface to functionalities inside OpenCV.
- Functions: See 3.2.3.5 Processing Detail for Each Operation
- Subordinates: LoadModel, CameraSetUpLiveVideo, CameraSetUpVideoPlayBack, CreateBatch, TransformShape, MotionHistoryTransform, MotionHistoryDifference.
- Dependencies: python-opencv4.1.2.3

Processing Details for LoadModel

- Type: Class
- Purpose: To load the ONNX model, make the model readable, and run the inference session of the model, and lastly set the input parameter name for the model in the form of a dictionary.
- Functions: See 3.2.3.5 Processing Detail for Each Operation
- Subordinates: CameraSetUpLiveVideo, CameraSetUpVideoPlayBack, CreateBatch, TransformShape, MotionHistoryTransform, MotionHistoryDifference.

- Dependencies: python-opencv4.1.2.3, tensorflow 2.2.0, onnx 1.7.0, onnxruntime: 1.3.0, torchvision 0.6.0, numpy 1.18.5, torch 1.5.0, sklearn.utils.extmath

Processing Details for CameraSetUpLiveVideo

- Type: Class
- Purpose: To setup the camera to be used with a live video feed.
- Functions: See 3.2.3.5 Processing Detail for Each Operation.
- Subordinates: CreateBatch, TransformShape, MotionHistoryTransform, MotionHistoryDifference.
- Dependencies: python-opencv4.1.2.3, time, webcam (either internal or external)

Processing Details for CameraSetUpVideoPlayBack

- Type: Class
- Purpose: To setup the camera to be used with an .mp4 video.
- Functions: See 3.2.3.5 Processing Detail for Each Operation.
- Subordinates: CreateBatch, TransformShape, MotionHistoryTransform, MotionHistoryDifference.
- Dependencies: python-opencv4.1.2.3, time, path to mp4 file

Processing Details for CreateBatch

- Type: Class
- Purpose: To create the batch list along with the batch size for both the regular model and the condensed-spaced model.
- Functions: See 3.2.3.5 Processing Detail for Each Operation.
- Subordinates: TransformShape, MotionHistoryTransform, MotionHistoryDifference.
- Dependencies: None.

Processing Details for TransformShape

- Type: Class
- Purpose: To begin the transformation process of the current frame to be used as input into the model.
- Functions: See 3.2.3.5 Processing Detail for Each Operation.
- Subordinates: MotionHistoryTransform, MotionHistoryDifference.
- Dependencies: python-opencv4.1.2.3, numpy 1.18.5, frame as input to the class constructor

Processing Details for MotionHistoryTransform

- Type: Class
- Purpose: To begin the creation of a motion history image.
- Functions: See 3.2.3.5 Processing Detail for Each Operation.
- Subordinates: MotionHistoryDifference.
- Dependencies: python-opencv4.1.2.3, numpy 1.18.5, frame as input to the class constructor, cv2 interface.

Processing Details for MotionHistoryDifference

- Type: Class
- Purpose: To create the resized frame used to calculate the difference between the previous frame and this frame.
- Functions: See 3.2.3.5 Processing Detail for Each Operation.
- Subordinates: None.
- Dependencies: python-opencv4.1.2.3, numpy 1.18.5, frame as input to the class constructor, cv2 interface, MotionHistoryTransform.dim.

3.2.3.1 Design Class Hierarchy for Components

Within the cv2 library, any related calls to an OpenCV function or class can only be completed after cv2 has been imported. As such, it is the most necessary class before all other calls to OpenCV classes/functions. The rest of the hierarchy is as follows. LoadModel should be instantiated first to ensure the model has properly be loaded into the program. CameraSetUpLiveVideo and CameraSetUpVideoPlayBack need to be instantiated after the LoadModel class. Then the class CreateBatch must be instantiated before the while loop. When inside the while loop, TransformShape is next, followed by MotionHistoryTransform, and MotionHistoryDifference.

3.2.3.2 Restrictions/Limitations for Components

Restrictions/limitations for cv2.VideoCapture()

The only limitations for this component are the ability of the system to capture video with at least the resolution and quality expected by the CNN model. L

Restrictions/limitations for LoadModel

An ONNX model must be provided as the class expects such a model when utilize various ONNX specific method calls.

Restrictions/limitations for CameraSetUpLiveVideo

Must have a camera capable of running at 32 fps.

Restrictions/limitations for CameraSetUpVideoPlayBack

Must have a camera capable of running at 32 fps and the path to the video file must be provided as a string in the proper format.

The system has only been tested for .mp4 files.

Restrictions/limitations for CreateBatch

The explicit restriction on batch size is a length of 32.

3.2.3.3 Performance issues for Components

Performance issues for cv2.VideoCapture()

There are no major anticipated performance issues for this component.

No other performance issues arise when utilizing the batch size of length 32.

3.2.3.4 Design Constraints

Design Constraint for cv2

- Must have access to a camera capable of running at 32 frames per second.

Design Constraint for LoadModel

- The main constraint is the model must be in ONNX format.

3.2.3.5 Processing Detail for Each Operation

Processing Detail for cv2

- cv2.imshow()
 - Loads the video feed window to show the live tracking of the person.
- cv2.COLOR_BGR2GRAY
 - Converts current 3-channel RGB into 1-channel grayscale.
- cv2.INTER_AREA
 - The method of resampling using pixel area relation when resizing an image/frame.
- cv2.resize
 - Resizes the image based on a specific height and width required by the model.
- cv2.CAP_PROP_FPS
 - Sets the frames per second.
- cv2.cvtColor
 - Is the method call used to change the channel color.
- cv2.absdiff
 - Takes the absolute difference from the previous frame and the current frame.
- cv2.waitKey
 - Waits for a pressed key.
- cv2.dnn.readNetFromONNX
 - Reads the ONNX CNN model so it can be used by the program.
- Cv2.VideoCapture
 - Captures the incoming frame from either a camera or a video.

Processing Detail for **LoadModel**

- Attribute `self.onnx_model`
 - Loads model from path.
- Attribute `self.model`
 - Receives the model from `cv2.dnn.readNetFromONNX` in a readable form.
- Attribute `self.ssess`
 - Uses `InferenceSession` to run/read the model.
- Attribute `self.input_name`
 - Sets the input name expected by the model when receiving input in the form of a dictionary such as `{model.input_name: model_input_x }`.

Processing Detail for **CameraSetUpLiveVideo**

- Attribute `self.camera` uses `cv2.VideoCapture()` along with several other key aspects
 - Initiates the video feed.
 - Sets the camera's frames per second.
 - Sets the video brightness.
- `camera.camera.read()`
 - Reads each frame of the video to analyze it for falls.

Processing Detail for **CameraSetUpVideoPlayback**

- Attribute `self.path_to_video` takes the path to the .mp4 file used.
- Attribute `self.camera` uses `cv2.VideoCapture()` along with several other key aspects
 - Initiates the video feed using `cv2.VideoCapture(self.path_to_video)`.
 - Sets the camera's frames per second.
 - Sets the video brightness.
- `camera.camera.read()`
 - Reads each frame of the video to analyze it for falls.

Processing Detail for **CreateBatch**

- Attribute `self.batch_size` sets the allowable batch size used for the list, which is 32.
- Attribute `self.batch`, which is a list used to hold each individual frame up to 32.
- Attribute `self.condense_batch`, which is a list used to hold each individual frame up to 32.

Processing Detail for **TransformShape**

- Attribute `self.frame`, which is a given NumPy array frame at time n .
- Attribute `self.frame_transform`, which is a NumPy array with a shape like the current frame at time n .

Processing Detail for **MotionHistoryTransform**

- Class Attribute `dim`, which is equal to the width and height of a frame.

- Attribute `self.frame`, which is a given NumPy array frame at time n but not $n+1$.
- Attribute `self.dims`, which is equal to the width, height, and channel value of the color of a frame.
- Attribute `self.dim`, which is equal to the width and height of a frame.
- Attribute `self.mhi_zeros`, which creates a NumPy array of zeros with the given shape, dtype, and order based on `self.dims`.
- Attribute `self.prev_frame`, which is a resized frame based on the current frame with a shape (width and height) of `self.dims` and a resampling pixel value when resizing of `INTER_AREA`.

Processing Detail for **MotionHistoryDifference**

- Attribute `self.frame`, which is a given NumPy array frame at time n , $n+1$, $n+2$, and *etc.*
- Attribute `self.resized`, which is a resized frame based on the current frame with a shape (width and height) of `MotionHistoryTransform.dim` and a resampling pixel value when resizing of `INTER_AREA`.

3.2.3.5.1 Processing Narrative for Each Operation

`cv2.VideoCapture()`

- When the program launches, the `cv2.VideoCapture()` starts the live video feed and reads the frames of the video. This allows for the program to analyze where the person in the video feed is located by accessing the frames from the video using the built-in operations.

`cv2.imshow()`

- After the motion history image has been computed, the frame is shown to the screen using `cv2.imshow()` for the user to see.

`cv2.COLOR_BGR2GRAY`

- Accepts a frame and converts the 3-channel RGB frame into a 1-channel RGB frame. This is used to further convert a given frame into an acceptable grayscale 3-channel frame, which is accepted as input by the model.

`cv2.INTER_AREA`

- The method for resampling pixels of a given frame during the resizing process, which is needed to create a frame, of which has an acceptable shape to be used as input into the CNN model. This occurs for each frame received by the component.

`cv2.resize`

- Is used to resize every frame into an acceptable shape, which takes a height and width parameter, a frame parameter, and an interpolation technique, of which is selected as `cv2.INTER_AREA`.

`LoadModel`

- The major processing inside this class include getting the model based on the path provided, reading the model in, running the model, and finally setting the input name required by the model.

`CameraSetUpLiveVideo`

- Only deals with operations related to the camera setup.

CameraSetUpVideoPlayBack

- Only deals with operations related to the input of an external video .mp4 file.

CreateBatch

- Will create the batch list to hold the 32 frames used as input into the model. Operations associated with this class will include appending to the list, checking the length of the list, transforming the list into a NumPy array, and finally resetting the list to empty or as a slice.

TransformShape

- Operations associated with TransformShape include creating a numpy array with a shape like the current frame. Another aspect related to TransformShape is assigning the 3-channel RGB to each 1-channel aspect in the TransformShape frame so for example: `frame_transform.frame_transform[:, :, 0] = frame` with 0-2 inclusive. The instance of this class will also take on the value of the motion history image and be further resized and have its dimensions expanded to fit the needed shape required by the model

MotionHistoryTransform

- Operations associated with this class include creating the first instance object of the class when the current frame value equals 1. The instance object will also be used when calculating the absolute difference between the current frame and the previous frame to get the correct blurring effect as seen through time when using the MHI.

MotionHistoryDifference

- Operations associated with this class include creating the last part of the MHI needed to calculate the difference between the previous frame and the current frame.

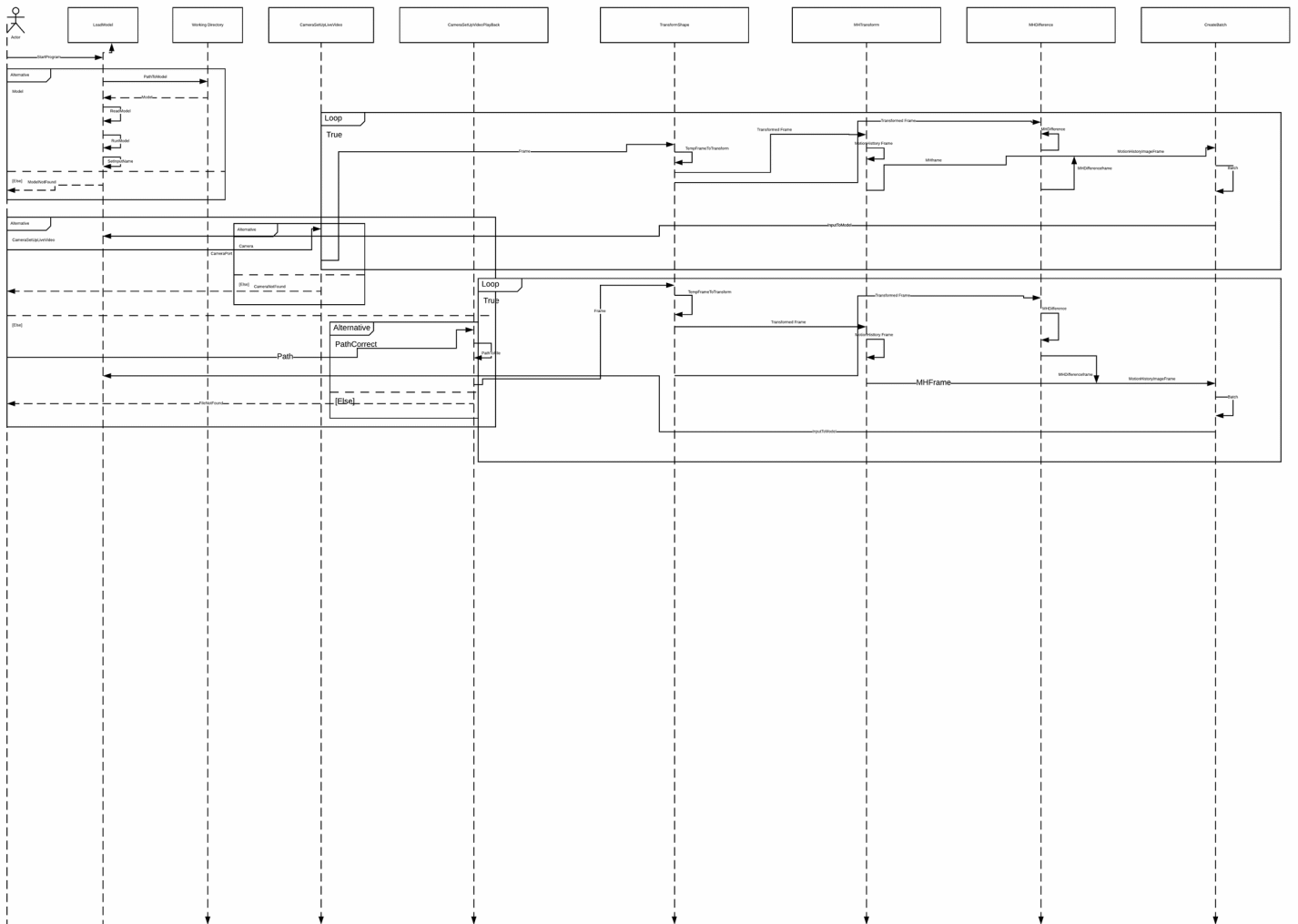
3.2.3.5.2 Algorithmic Model for Each Operation

None.

3.3 Dynamic Behavior for Components

Classes/Components do not directly interact with each other in the FRIDA project. If an interaction is taking place it is likely through some primitive, which has an expression. The only instance where this is not true is when using cv2 inside of the various classes, such as CameraSetUpVideoPlayBack and CameraSetUpLiveVideo, along with MotionHistoryTransform and MotionHistoryDifference. The only other interaction of a class inside another class is when MotionHistoryDifference gets instantiated by an object. When this occurs, MotionHistoryDifference makes a call to the class variable MotionHistoryTransform.dim inside the class MotionHistoryTransform.

3.3.1 Interaction Diagrams



4.0 User Interface Design

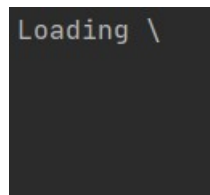
There will be a primary interface for the program: PyCharm's built-in system console. However, upon the installation of the software, the user will need to utilize the Anaconda distribution to import the provided libraries and to create a virtual environment for PyCharm to utilize as its project interpreter. Additional libraries, specifically OpenCV,

ONNX, and ONNX Runtime, will need to be installed within PyCharm's terminal as well.

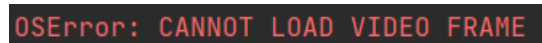
4.1 Description of the User Interface

Below will be examples of the user interface (UI) being used within PyCharm's console, alongside the video frame's heads-up display. After the administrator initiates the program, it will first detect and see if a webcam or video file is available. If none is detected, it will display an error. Otherwise, it will output a loading statement, and if the video frame loads, it will output a software success statement. Afterward, it will start continuously displaying fall and non-fall prediction statements, and when a fall occurs, alert(s) stating a fall has been detected will be displayed. In the video frame's heads-up display, it will display a status of idle when no fall is detected, and then a fall detection status when one is. By default, the video frame will be in grayscale, but an alternative or additional option of using the background subtraction option is available as well. When using the live video feed option, when the user terminates the program, it will output a corresponding termination message accordingly. When using the video dataset file option, the program will terminate automatically.

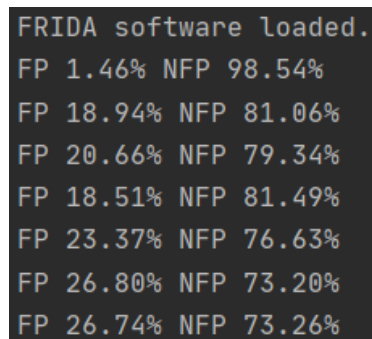
4.1.1 Screen Images



```
Loading \
```



```
OSError: CANNOT LOAD VIDEO FRAME
```

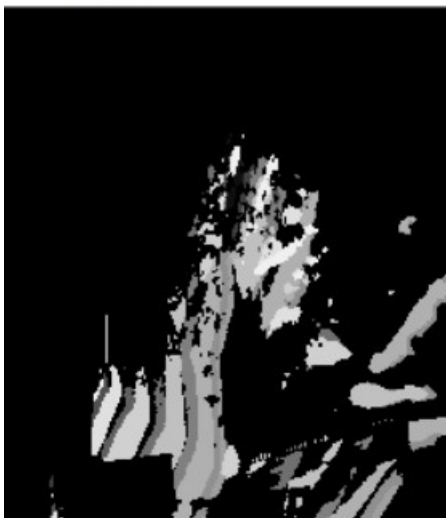


```
FRIDA software loaded.  
FP 1.46% NFP 98.54%  
FP 18.94% NFP 81.06%  
FP 20.66% NFP 79.34%  
FP 18.51% NFP 81.49%  
FP 23.37% NFP 76.63%  
FP 26.80% NFP 73.20%  
FP 26.74% NFP 73.26%
```

Video Feed



Backgro...



FP 33.93% NFP 66.07%
FP 49.11% NFP 50.89%
FP 54.87% NFP 45.13%
FALL DETECTED
FP 7.74% NFP 92.26%
FP 7.80% NFP 92.20%
FP 5.91% NFP 94.09%
FP 6.76% NFP 93.24%


```
FP 16.17% NFP 83.83%  
FP 13.26% NFP 86.74%  
FP 13.07% NFP 86.93%  
  
VIDEO FEED TERMINATED
```

4.1.2 Objects and Actions

```
Loading \  
FRIDA software loaded.
```

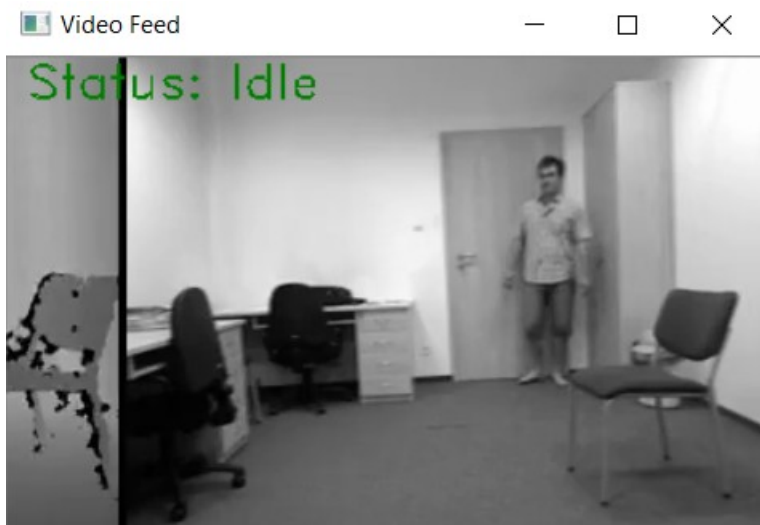
The software will print a loading status upon the initiation of the program, and then print a success statement if the video frame was able to be opened.

```
OSError: CANNOT LOAD VIDEO FRAME
```

The software will print an error message to the console and to terminate the program if a webcam or video file is not detected.

```
FP 7.48% NFP 92.52%  
FP 7.73% NFP 92.27%  
FP 3.34% NFP 96.66%  
FP 4.07% NFP 95.93%  
FP 1.09% NFP 98.91%  
FP 0.43% NFP 99.57%  
FP 0.61% NFP 99.39%  
FP 0.72% NFP 99.28%
```

The software will continuously print fall (FP) and non-fall (NFP) prediction percentages to the console while it is analyzing and monitoring for a fall.



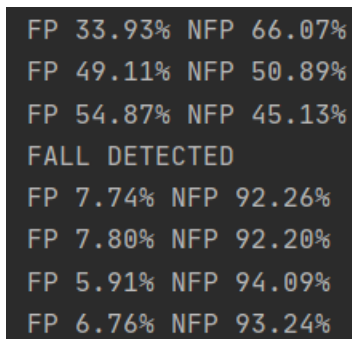
The software will display the video frame in grayscale by default.



If the background subtraction option is chosen, it will be displayed as a video frame either in place or grayscale video frame or as an additional video frame window.



The software will print a heads-up display within the video frame that shows the fall detection status as either being detected or idle.



The software will print a fall detection statement if a fall was detected, and then continue monitoring.



The software will print a termination statement if the user terminates the program by pressing 'q' on their keyboard while using the live video feed option.

4.2 Interface Design Rules

The interface design rules for the FRIDA software are based on the concept KISS (Keep It Simple, Stupid). The following is a list of the rules:

1. Be Accurate
 - a. The program should make the user feel confident about the accuracy of the FRIDA software by successfully identifying at least 80% of the falls.
2. Give Users Quick Feedback
 - a. The FRIDA software should give the user rapid feedback of less than 10 seconds of a fall event so that a quick reaction can take place.
3. Be Simple
 - a. Make the program user-friendly, so that any demographic of user can use the program with ease.
4. Keep Bandwidth Down
 - a. If a developer were to implement the software onto a cloud-based server, the program should be designed as to take up less processing power, therefore less bandwidth, for it to be more usable online.
5. Allow for False Alerts
 - a. The program should take the worry from the user and allow the person being tracked to send a false alert, keeping the user from having to react to the situation.

4.3 Components Available

None are available in the form of a GUI.

4.4 UIDS Description

There is no user interface to develop. The user of the FRIDA software will launch the program from within PyCharm and then utilize its built-in console. Upon launch, video frame(s) will be instantiated to display the video feed or video input file. After launch, the program runs uninterrupted, with the only interaction being the termination of a live video feed by the user and notifications sent from the console for fall, error, loading, success, or termination data.

5.0 Restrictions, Limitations, and Constraints

Time

Time is so far the biggest restriction or constraint for our project as we only have around 8 weeks to finish entire project. It is very important for us to watch the time we spend over every phase of the software development project. We could have done more training for the system model and included many more components to the software, but time restricts us from doing so.

Employee Skills

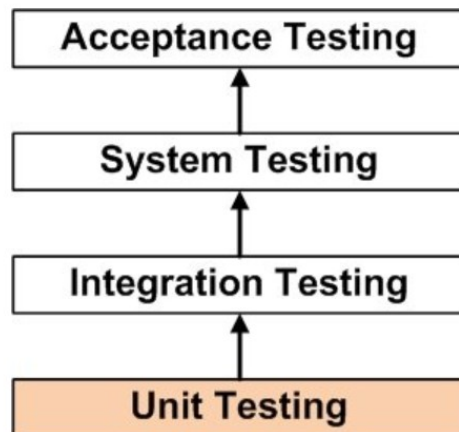
Employees' programming and design skills is also one of the restrictions. It does not have as big of an impact on the project as time, but it undoubtedly limits us from doing more addition to the projects.

Environment

A huge constraint for the FRIDA system is the narrowly contained ability to function in specific environments which provide adequate lighting, of which makes webcam and/or external cameras capable of producing quality live video input data that can be analyzed for fall states. At the same time, the person under monitoring must ensure they are in view of the camera (i.e., are not unobstructed so as to prevent monitoring from taking place).

6.0 Testing Issues

A combination of white-box and black-box testing will be utilized to ensure proper testing of the FRIDA system in an attempt to reduce the number of bugs and provide a system which is performance-based, functional, dependable, maintainable, and usable. The format of the testing strategy can be seen in the figure below.



White-Box Testing:

White-box testing implementations will include unit testing/component testing, integration testing, and system testing. For the white-box testing on this product, we will run the program against a number of datasets which depict falls and non-falls and watch the values that are coming in and going out. We will also be watching the variable values and changes in them as the program runs through subroutines variable values when the program path changes.

Black-Box Testing:

Black-box implementations will include primarily acceptance testing with a focus on including equivalence partitioning and boundary value analysis in an attempt to ensure the FRIDA system is able to detect falls at 80% or greater accuracy rate.

Test Case 1

Test Case #	Test Case Description	Test Data	Expected Result
1	Run the program with a webcam connected	Webcam	The program should run without a warning/error.

During test execution time, the tester will check expected results against actual results and assign a pass or fail status for when a webcam is connected to the FRIDA system.

Test Case 2

Test Case #	Test Case Description	Test Data	Expected Result
2	Run the program without a webcam connected	Webcam	The program should run and error out.

During test execution time, the tester will check expected results against actual results and assign a pass or fail status for when a Webcam is not connected to the FRIDA system.

Test Case 3

Test Case #	Test Case Description	Test Data	Expected Result
3	Checking fall and non-fall prediction data is being created	Webcam or video file, Person	The program should print the fall and non-fall prediction data to the system console.

During the test execution time, the tester will check the expected results against the actual results and assign a pass or fail status for whether or not the fall and non-fall prediction data is being printed to the system console.

Test Case 4

Test Case #	Test Case Description	Test Data	Expected Result
4	Checking the live video feed frame is displaying a heads-up display	Video input	The program should display a status stating either its in idle or fall detected state.

During the test execution time, the tester will check expected results against the actual results and assign a pass or fail status for when a webcam is connected to the FRIDA system and has a person in view who is moving.

Test Case 5

Test Case #	Test Case Description	Test Data	Expected Result
5	Check if the background subtraction video frame option displays.	Live video feed or video file	The program should display the background subtraction video frame if the user chooses it.

During the test execution time, the tester will check expected results against the actual results and assign a pass or fail status for whether or not the background subtraction video frame launches.

Test Case 6

Test Case #	Test Case Description	Test Data	Expected Result
6	Check if the default grayscale video frame option displays.	Live video feed or video file	The program should display the grayscale video frame.

During the test execution time, the tester will check expected results against the actual results and assign a pass or fail status for whether or not the grayscale video frame successfully loaded.

Test Case 7

Test Case #	Test Case Description	Test Data	Expected Result
7	Check for the fall state	Webcam or video file, Person	The program should run and check if a fall state is currently detected through a notification alert to the system console.

During the test execution time, the tester will check expected results against the actual results and assign a pass or fail status for when a webcam is connected to the FRIDA system and has a person in view and a fall state is active.

Test Case 8

Test Case #	Test Case Description	Test Data	Expected Result
8	Check for a live video frame termination notification	Live video feed	The program should display a termination notification to the system console when the 'q' key is pressed on their keyboard.

During the test execution time, the tester will check expected results against the actual results and assign a pass or fail status for whether or not the program successfully terminates upon the user pressing the 'q' key on their computer's keyboard.

6.1 Performance Bounds

We have set up certain bound or criteria for our software so that by following said criteria, we should be able to maintain high levels of accuracy.

Response time after detecting a fall state

Best Case Scenario: Immediate

Worst Case Scenario: Under 10 seconds

Fall prediction rate

Best Case Scenario: 100%

Worst Case Scenario: Less than 50%

The video input data is another performance bound we have to deal with as well. Currently, we are allowing for the maximum available video resolution at runtime. However, when used in the real-world, video resolution may need to be lowered so that data transfer cost can be minimized if a developer decides to implement the program to work on the cloud.

6.2 Identification of Critical Components

All components listed are critical. There is no component which can be removed that would allow the software to function as intended. However, in terms of components which demand attention during the testing phase, LoadModel component must be properly working to ensure fall detection can take place. The components of TransformFrame, MotionHistoryTransform, and MotionHistoryDifference are another place where testing needs to ensure all is functional as these 3 components are responsible for taking each frame and making it into a MHI, which has the correct shape needed by the model to run/provide results.