FALL DETECTION USING A SINGLE VIDEO CAMERA

by

Elizabeth Ann Schlegel

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Mechanical Engineering

Charlotte

2015

ABSTRACT

ELIZABETH ANN SCHLEGEL. Monitoring fall detection using a single video camera.
(Under the direction of DR. NIGEL ZHENG)

Falls among the elderly are concerning, especially if they go unnoticed for an extended period of time. Due to fear of falling, many people end up in assisted living facilities or nursing homes which are costly and the lack of independence greatly affects quality of life. Finding ways to allow the elderly to safely stay in their own homes longer not only reduces cost of living and increases well-being, but also alleviates the families' burden of worrying about their loved ones. Due to their low cost, cameras are a convenient option for monitoring. This study made use of this and developed an algorithm to be used with a single video camera in order to determine when someone has fallen or is about to fall, so an alarm could be set off to alert a caretaker for help.

The developed algorithm was tested for a variety of scenarios including falling down stairs, partially occluding the subject, low light levels, and placing the camera at a variety of angles. The acceleration values obtained were also compared with an Inertial Measurement Unit. The algorithm successfully tracked the person and detected falls in each scenario. The comparison with the IMU sensor points toward further potential for clinical applications in fall risk assessment.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my academic adviser, Dr. Nigel Zheng for the direction and guidance throughout the completion of this thesis and my graduate work. My sincere appreciation also goes to Dr. Alireza Tabarraei and Dr. Ron Smelser for serving on my thesis committee and providing professional advice.

I would also like to thank my labmates, Matt Edwards, SamWang, Peter Xu, and Tim No for their helpful comments and contributions.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Falls among the elderly are concerning, especially if they go unnoticed for an extended period of time. According to the CDC, 1800 die from falls every year [1]. Many of the elderly end up in either nursing homes or assisted living facilities due to family safety concerns of the person living alone which is costly and not typically preferred as independence is a primary factor in evaluating quality of life. As seen in the graph below from the U.S. Census Bureau, people in the United States are living longer and the projected trend of the number of elderly adults is steadily rising.

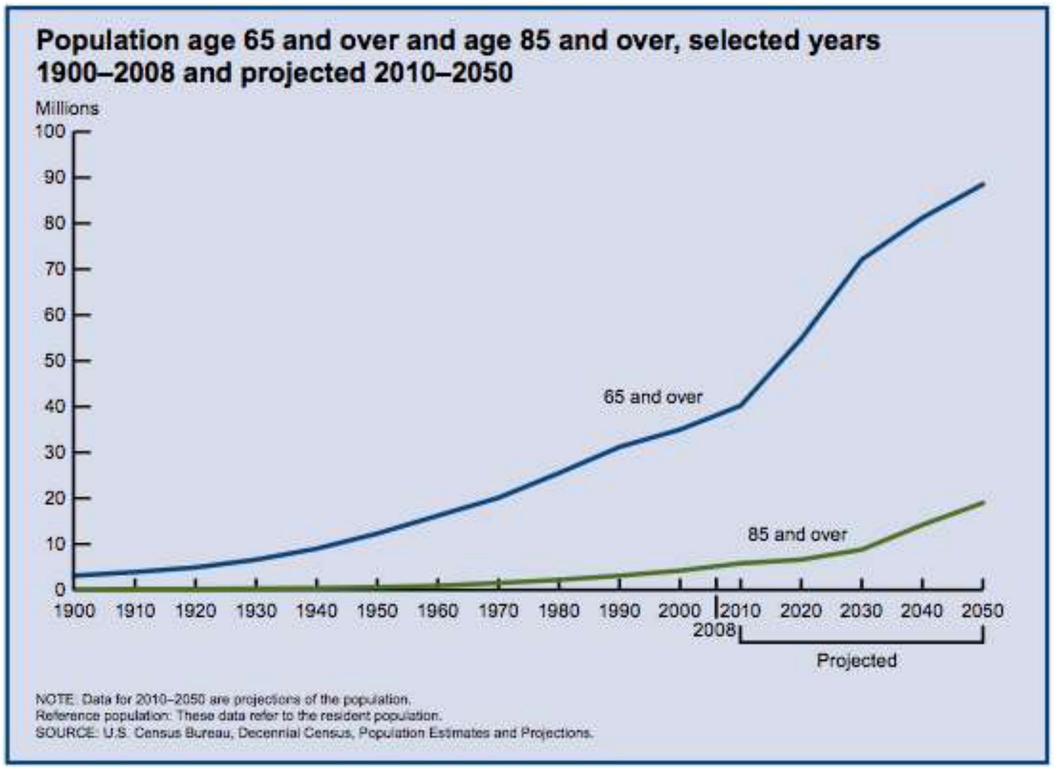

Figure 1: Current and projected elderly population in the United States [2]

Table 1: Average monthly cost of living [3]

Nursing Homes (semi-private)	**Nursing Homes (private)**	**Assisted Living (private)**
$5,430	$6,150	$2,714

This aging population attributes to the concerns of caring for these individuals with the increasing need for nursing homes and assisted living facilities. Table 1 shows that it is indeed costly to live in one of these facilities.

Because quality of life dramatically decreases with increasing financial burden and considering the rising number of people who are needing these services, finding ways for these people to remain independent and able to live on their own not only eases a little of the families' and societies' burden, but most importantly independence increases the quality of life for these individuals.

Risk levels for fall are often evaluated using mobility tests, such as timed up and go (TUG) and motion capture camera systems [4,5]. For fall detection and monitoring, there currently are a variety of methods, most notably LifeAlert. LifeAlert is a product that helps those who have fallen and cannot stand back up notify others for help. It is a button hung on a lanyard around the neck that when pushed, calls emergency services for help. This is a simple and effective product that many people rely on; however, it has its drawbacks. The person must always be wearing the device and most importantly, if and when the person does fall, they have to remain conscious and aware enough to both remember and be capable of pressing the button. Accelerometers are also occasionally used, but these also need to be worn and relying on acceleration alone often yields false positives in that the acceleration reaches a certain threshold value that indicates that a fall has occurred when in fact no fall has [6].

For tracking human motion for both risk assessment and monitoring, video cameras are now becoming more popular due to their low cost with depth cameras being frequently used. Depth cameras use a technique known as range imaging to create two dimensional images that depict the distances to specific points in a scene from one reference point. This resulting two dimensional image is called the range image and when it is calibrated, the pixel distances are scaled to the actual physical distances [7]. Because the depth cameras allow for analysis in three dimensions and options such as the Microsoft Kinect are relatively inexpensive, they are attractive options. However, some people already have other non-depth cameras in place, such as security cameras, and being able to use these already established cameras is beneficial.

Because an effective algorithm is both functional and computationally inexpensive, this research creates a new algorithm using and modifying previously established segmentation and edge detection techniques. It successfully tracks a person as they move throughout a scene and performs accurate fall detection. This variety of image processing techniques that were employed are described in Chapter 2.

CHAPTER 2: IMAGE PROCESSING

Image analysis is performed using a variety of algorithms and methods. Some techniques are more effective than others and usually depend on the particular application. These algorithms can be classified many ways into various logical groups which can then be combined and grouped further into application sub groups. The two major logical groups being employed in this work are segmentation and edge detection which combine to create various methods for object tracking.

2.1 Segmentation

Segmentation is the process of dividing an image based on shapes and pixel intensity values. If the algorithm uses both shape and intensity values to classify and segment, then it is known as Contextual Segmentation whereas if only intensity values are used, it is known as Non-Contextual Segmentation.

A frequently used example of contextual segmentation is Delaunay Triangulation. This algorithm is better recognized in the field of finite elements, but has related uses in the field of image classification as well. It is a triangulation of a set of points such that no point inside the circumcircle of any triangle lies within the set of points. It attempts to maximize the interior angles of the triangles i.e. it is the nerve of the cells in a Voronoi diagram. Remi and Bernard discuss in their paper a generic tracking algorithm based on Delaunay Triangulation that effectively tracks objects without needing a specific model to be initially defined, making it adaptable to work with other applications [8].

Used extensively throughout this research, thresholding is a prime example of non-contextual segmentation and is the basis for background removal. Accurate and efficient extracting of moving objects for detection and tracking relies on efficient background elimination and noise removal. Segregating the background from the foreground is accomplished by setting a threshold for the differences between two frames, typically of the same setting and at different times. By overlaying the frames and performing a pixel by pixel subtraction, pixels of the target frame whose difference in intensities from the reference frame fall below a certain difference threshold are considered to be the same in both frames and therefore to be part of the background while the target frame pixel is set to zero. Any set of pixels whose difference is above the threshold is considered to be part of the moving foreground and the corresponding pixels in the target frame are either unchanged or set to one, depending on the desired application. This subtraction and comparing of pixels is known as frame differencing, and while there are various algorithms for background removal, they all use this as a basis [9].

In the vast majority of cases, the background will not remain entirely unchanged. For instance, furniture gets moved, items initially not in the frame come in and are left, etc. To account for these changes, the reference frame should continuously be updated after a given amount of time so that excess noise can be minimized. This updating of the background frame is known as adaptive background subtraction. This method has been shown to be more effective and efficient at locating the foreground and minimizing background noise [10].

Even using modified differencing methods like adaptive background subtraction, some noise usually still exists and therefore a filter needs to be applied to obtain the desired isolated image. Three commonly used filters include Gaussian, mean, and the one used in this research, the Wiener filter.

The Gaussian filter is a low pass filter that modifies the input signal by convoluting it using the Gaussian function. This method is also known as the Weierstrass Transform after Karl Weierstrass. The Gaussian distribution is a standard bell curve which plays an important role in its application to filtering as it defines a probability distribution for noise. When applying the function to images, instead of being only one dimensional, it needs to be two dimensional; this is just the product of two one dimensional Gaussian functions, one in each direction. Drawbacks of using the Gaussian filter are that because it works by smoothing an image, it is not effective at removing what is known as salt and pepper noise, the random black and white specs that appear can appear in images, nor does it typically maintain the brightness of the original image [11].

The mean filter is much more effective at reducing this salt and pepper noise as it slides over each pixel and replaces its value with the mean of the surrounding pixels. This surrounding area is determined from an initial value preset by the user. Depending on the application, the typical noise amount present, and the desired effectiveness, different size windows are chosen. Similar to the mean filter is the median filter which replaces each pixel value with the surrounding area’s median value rather than its mean. An attractive advantage of using a median filter, rather than a mean filter, is that it is better at preserving the details of the original image [11].

Chosen for this research, the Wiener filter, named for Norbert Wiener, is a linear filter that minimizes the mean square error of the desired response and the actual output of the filter. The minimum of the mean square error occurs when the correlation between the signal and the error is zero. Mathematically, this is when the two signals are said to be orthogonal, meaning the dot product of the two is equal to zero. The Wiener filter is frequently used in deconvolution which leads to many applications in image processing, especially for adaptive background suppression [12].

2.2 Edge Detection

Locating the boundaries of desired objects is another fundamental aspect of image processing. Helping to further segment images in order to locate and match objects, edge detection works by determining the gradient. This involves locating the discontinuities in intensity values throughout the image. There are many effective methods for doing this but just two of the most common methods, Sobel and Canny, will be discussed here.

Frequently used in many mathematical applications, the fundamental definition of the gradient is the derivative of one function in one direction to the function in multiple directions. In image analysis, the gradient is used to locate the change in brightness levels, indicating edges. To fully define the edge, the gradient is computed in both the horizontal and vertical directions. Edge detection algorithms, including the Sobel and Canny methods described below, incorporate finding the magnitude of this gradient.

As mentioned, the Sobel method, named for Irwin Sobel, determines an approximate gradient of the image intensity function. It is based on convolving the image with a small and separable filter in both the horizontal and vertical directions. The simplicity of this algorithm makes it attractive for being computationally inexpensive.

However, while it is proven to be effective, for some applications it can be too crude of an approximation [13].

More effective than the Sobel operator, the Canny edge detection algorithm uses a multi-step algorithm to increase its robustness. The key steps within this method are first applying a Gaussian blur, next computing the intensity gradients, and finally tracing along the found edges and suppressing the non-maximum ones. The Gaussian blur is the convolving of an image using a Gaussian function which serves to reduce the image detail and noise. Next, the magnitude and directional derivative of the gradient is calculated to locate the edges. Once the edge has been located and the gradient directions known, the edge direction is related to a direction that can be traced in an image. Because pixels are arranged in a grid, when describing the directions of one pixel to surrounding pixels, there are only four directions that can be used. Zero degrees horizontally, ninety degrees vertically, forty-five degrees positively diagonal, and 135 degrees negatively diagonal. Therefore, the algorithm traces along the edge and rounds each edge direction to the closest applicable direction. To yield a more distinct edge, non-maximum suppression is applied. Again, the algorithm traces along the detected edge and it suppresses any point that it considers to be weak. To avoid image streaking, hysteresis is used. Hysteresis uses a high and a low threshold so any pixel that is higher than the high threshold is considered an edge pixel, and any pixel next to an already confirmed edge that has a value greater than the low threshold also remains as an edge [13].

2.3 Object Location and Tracking

Dividing and organizing an image in the desired way and detecting and isolating the edges of desired objects are the key components in recognizing and then tracking the objects. There are countless algorithms already in use for detecting and tracking objects, including people. Which application and outcome is targeted determines which one or which combination of multiple is used.

Many algorithms have a basis using a Bayesian approach. Bayes theorem is a statistical theorem that relates current probability to prior probability. Particle filters are density estimator algorithms that use Bayesian recursion to estimate the probability of hidden parameters based on observable data. These filters are known to be effective and are frequently used in object tracking [14].

An often explored algorithm used both by itself and in conjunction with other approaches is the Iterative Closest Point Algorithm (ICP). This algorithm relates corresponding sets of points and attempts to minimize the difference between them by iteratively translating and rotating points until a solution converges. Aligning the point sets allows for efficient feature recognition used in object recognition and motion tracking. Although this algorithm is efficient in both two and three dimensional applications, often times it is unable to effectively function when applied to rapidly moving human body motion. Various modifications have been applied to solve this problem including fitting the ICP algorithm with a modified particle filter algorithm discussed previously [15].

Template matching is another common method used to identify objects and object positions in a noisy environment. A template, either binary, color, or greyscale, translates over the target image, and at each place, a comparison is made to determine whether the template matches the object. Template matching can be an effective and efficient method when it is known that a specific object exists in the frame, and an estimate of the pattern is known. Medical image processing, facial and body part recognition, and traffic feature detection are just a few common applications.

Cross correlation, also known as the sliding dot product, measures the similarity between two inputs. In image processing, the inputs are usually a target image and a template that is to be matched to this image. Geometrically, the dot product can be defined as the product of the Euclidean norm of the two vectors and the cosine of the angle between them. Within template matching, this means that in order to have a maximum similarity between the template and the target image, the cosine of the angle needs to be at a minimum to therefore maximize the dot product. Cross correlation is called a sliding dot product as the dot product is continuously calculated as the template translates across the target image in discrete steps until an acceptable correlation value is calculated. This value is typically normalized, as normalized cross correlation values allow for better comparison. As this can be a tedious process, especially if no known approximation of where the target object is located exists, modifications are typically made for a more efficient algorithm. Creating pre-calculated sum tables and approximating the template function aids in more efficient calculations [16].

Chamfer matching is, at the moment, the most efficient method of shape matching. This algorithm reduces the number of edges to match and instead uses the minimum number of defined and unvarying edges to yield quick and accurate matches. This is especially useful when analyzing low level images. Some have improved the basic form of this algorithm by incorporating edge orientation which creates a smoother loss function [17]. This method can be further simplified while still maintaining an acceptable level of accuracy by incorporating a theory based on centroidal Voronoi tessellations to use fewer pixels. Reducing the number of points used is accomplished in two ways. One of these is by adjusting the point densities of both the target image and the template. The other is optimizing the measured distances between points so that removing the points keeps the distance map the same [18].

To further increase the efficiency of template matching, a hierarchy of templates can be created. Considering how computationally expensive it is to compare a single template over an entire image using discrete steps, even if using the Fast Normalized Cross Correlation method described above, improvements are always sought to improve its effectiveness. Rather than searching through every template one by one, and possibly missing matches due to slight scaling and resolution differences, a hierarchy allows the templates to be grouped so that an initial general template grouped is first determined and then sorted like so through the lower levels of that group. Because objects in a scene can be depicted at different scales and resolutions, having a pyramid of templates to sort through significantly reduces computational time [19].

CHAPTER 3: ALGORITHM

This chapter details the steps of the proposed algorithm with the flowchart below outlining the major functions.

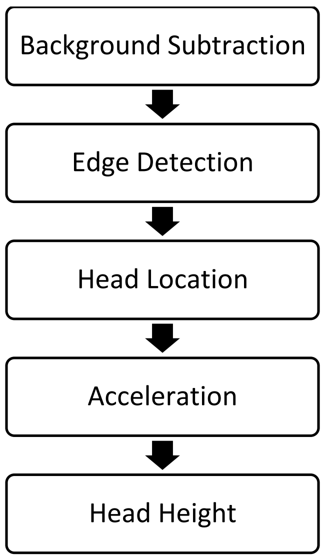

Figure 2: Algorithm outline

3.1 Background Removal

Segmenting the background from the moving foreground is used for object detection and localization of moving objects. This is accomplished using a frame differencing method and applying a filter to eliminate noise. Because this application involves a possibly changing background environment, an adaptive background subtraction method is used. As a person moves throughout a room, typically chairs, lamps, and other furniture get shifted and move through the frame. Items like books and phones are brought in to the frame and rearranged. Initially the reference frame for frame comparison and subsequent background and foreground segmentation is an initial frame of the room/setting with no person present. If this initial frame remains the reference frame, every time an object is shifted, it remains recognized as the foreground. As more and more gets shifted, the frame becomes cluttered with noise and the person will no longer be localized for analysis. Figure 3 below shows an example of this.

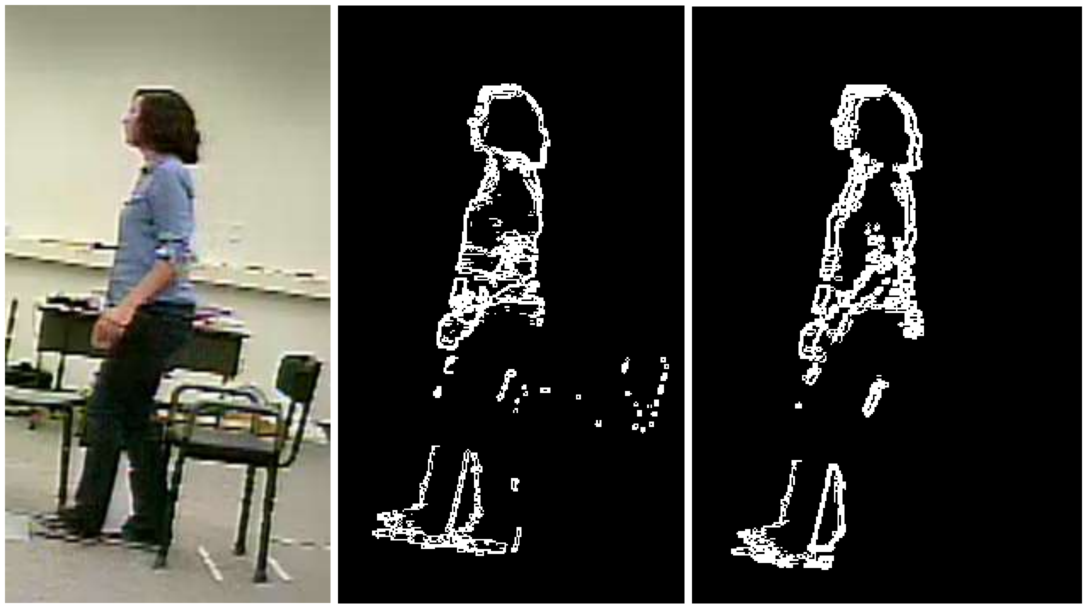

Figure 3: Adaptive background subtraction

In Figure 3 above, the middle picture shows the background subtraction using the initial frame as the reference frame while the far right picture shows the background subtraction using the adaptive method. Updating the reference frame allows for this smoother image with less noise. In this application, the reference frame is updated every two seconds.

Although using adaptive background subtraction eliminates the noise due to pieces of the environment shifting, other random noise due to lower image quality and variations in the environment illumination still exists. As discussed previously, many various filters exist for removing noise and they each are attractive for different applications. For this tracking application, the Wiener filter is found to be the most effective in eliminating noise and is therefore used. Dependent on both the desired final quality and the size of the noise particles, the window size for the filter varies.

The window size chosen here is 16 pixels by 12 pixels. Figure 4 below shows the result of using this filter. Eliminating this noise allows for a clean picture with an isolated subject that can be more easily located and tracked.

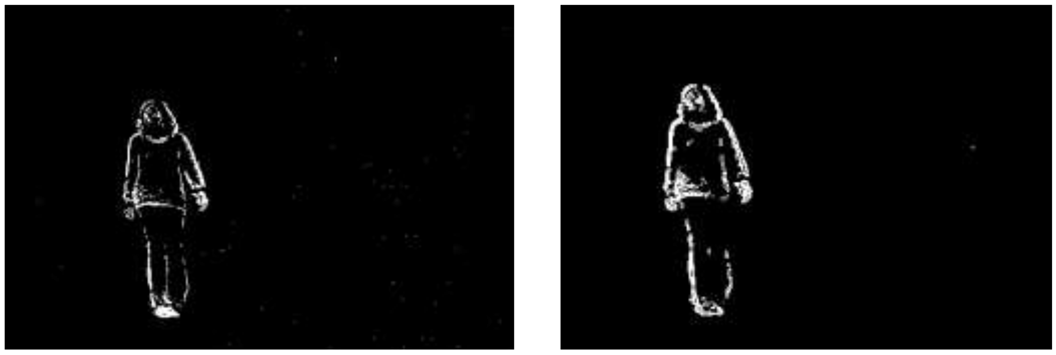

Figure 4: Effect of using the Wiener Filter. The original is on the left

3.2 Edge Detection

Isolating the subject from the background environment is a key step leading up to locating and tracking them, however an intermediary step still needs to be taken. In order for the algorithm to consistently locate the subject precisely and accurately, the outline of the subject should be well defined. Shown in Figure 5, an edge detection algorithm is applied to detect and define this outline.

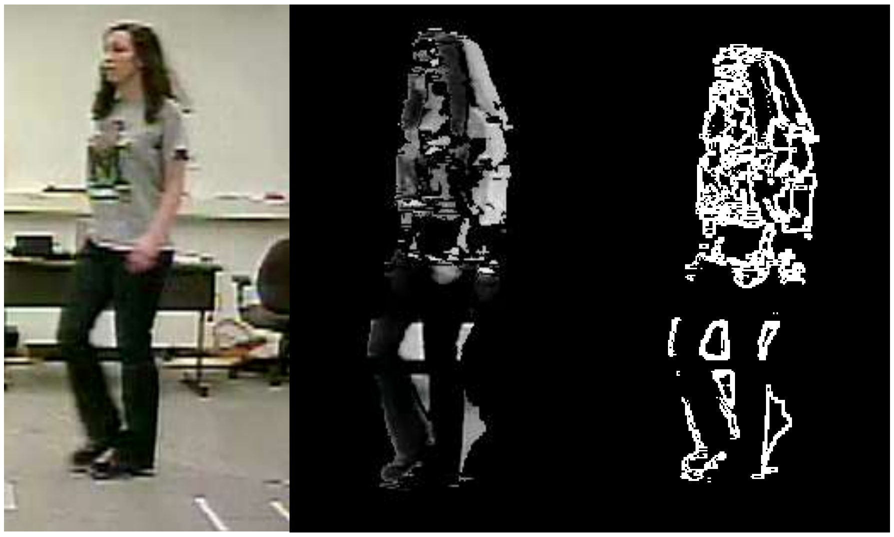

Figure 5: Defining a distinct edge.

The magnitude of the image gradient using the Sobel method defined the edges in Figure 5. The middle picture depicts the initial step of removing the background while the last picture on the right shows the image after applying the edge detection function.

3.3 Tracking

Once the person has been isolated with a defined edge, the next part of the algorithm locates the person and continues to track them as they move in and out of the frame. Because the person is isolated from the background and is considered to be walking upright, the algorithm searches through the rows from top to bottom until it finds a pixel value equal to one. This is the highest point of the person and is considered the head. Once a point is located, limits are set for future locating and accurate tracking. To help eliminate false identifiers because of noise, future search limits are established as possible places where the head could be located in the next frame. These limits are set to be thirty pixels to the left and to the right of the current head location. Each time a new valid point is located, the limits are shifted for the next search. If a valid point is not found within a certain amount of tries, the algorithm resets the limits to their initial values to include the entire frame. This works for not only the person leaving and reentering the frame, but also if any noise remains after the filter and the algorithm locates it first instead of the person. Because the noise is not static, the algorithm does not mistakenly track it. It instead hits the limit of attempts at trying to find a valid point and resets itself.

With consistent accurate tracking of the person's head, the acceleration of the movement is continuously calculated and monitored. The first indication that the person may have fallen occurs when this acceleration reaches a certain threshold. This threshold is set low enough to detect potential falls but sometimes gets triggered when the person first begins to walk into the frame because of sudden changes due to the person moving from being partly in the frame to finally being there entirely. Because the acceleration tends to spike as the person first walks into the frame, a weighted moving average was used to help reduce false instances.

3.4 Fall Detection

Because acceleration can yield false positives, it is only used as an initial indication. If the acceleration threshold is reached, the algorithm then runs through a series of steps checking the relative height changes of the head to confirm that a fall has indeed taken place. The frames for determining the height change are chosen to give the person a chance to stand back up and also to take into account that some falls include slower stumbles of somebody attempting to right themselves before actually falling. If the reference frames are chosen in too quick a succession, the algorithm may determine that the person is still standing and are therefore fine when in reality they may be just about to finally fall. With this, even if the person does fall, sometimes they are uninjured and able to stand. Again, if the subsequent reference frames are not spaced far enough apart, the algorithm may detect a fall and send an alarm when indeed the person needs no alarm sent. The first reference frame is the frame where the acceleration value initially reaches the threshold while the second reference frame is the frame two seconds after this event. Next, a series of checks are repeated keeping the initial reference frame and updating the second reference frame over the course of a few seconds to more accurately confirm a fall. A scaling factor is used to account for relative pixel height differences in different areas of the frame.

Because the background updates every couple of seconds, if a person falls, there is a risk that they will disappear into the background and be unseen by the program. To account for this, once the acceleration threshold is reached, the background stops updating and the reference frame becomes the frame where the acceleration value initially reached the threshold.

3.5 TUG Comparison

TUG (Timed Up and Go) testing is used in fall prevention research as a means to assess the risk of falling in an aging population. The subjects were fitted with motion capture markers and accelerometers, and a variety of walking tests were performed and recorded. Using the recordings, the acceleration values were calculated using the fall detection algorithm and compared with the values obtained from the accelerometers.

Because the background does not change during the TUG tests, as nothing is brought in and nothing is moved, adaptive background subtraction is not needed. Therefore, in order to create a more distinct and continuous edge, an initial frame with the subject not present is used as the reference frame for background subtraction throughout the process instead of updating it every few frames which causes a slight shadow to follow behind. In fall detection, this shadow does not affect the acceleration trends and height calculations however it would affect precise acceleration calculations desired in TUG testing.

CHAPTER 4: RESULTS

4.1 Fall Detection Results

This algorithm has been found to successfully track a person as they move in and out of the frame and accurately perform fall detection. Different scenarios were tested using previously recorded videos of people falling. Falling at various camera angles, distances from the camera, and in varying levels of light were all tested, and falls were successfully detected with minimal false positives. The algorithm also successfully detected falls down flights of stairs. Figure 6 below depicts one example of the algorithm successfully tracking a fall. The red X in the following figures depict the algorithm's location of the head.

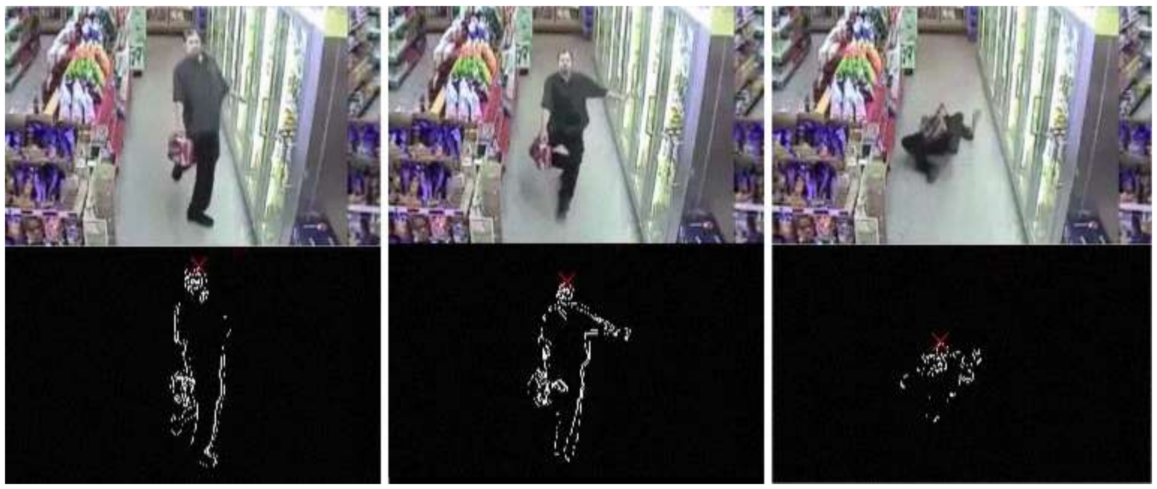

Figure 6: Successfully detecting a fall

The first concept to note concerns confirming a positive fall detection. Because the program employs adaptive background subtraction, the subject should disappear into the background once they fall due to their little or no movement. To counteract this, once the acceleration reaches the threshold, the background stops updating to allow the person to remain visible. This allows the algorithm to continue stepping through its height change checks to accurately perform a fall confirmation or rejection.

Figure 6 also demonstrates the algorithm’s ability to overcome the difficulty of portraying a three dimensional scene in two dimensions and successfully analyzing this. The last image in the sequence shows the person on the ground. The human eye views the top view of the scene and sees that the person has fallen backwards. However, this is not obvious when the background is removed and only the outline of the person remains. This illustrates the importance of checking the person’s relative change in head position rather than checking the height of their head from the ground. Using the latter method, the algorithm would locate the ground at the person’s feet and check the height from the top of their head. In this instance of the person falling directly backwards, this may yield the false impression that the person is standing. Because the algorithm employs determining the change in the person’s head position from them standing just before the fall to the position afterwards, it successfully recognizes and confirms the fall.

The subject recovering after an initial fall and lower video quality and light levels are other challenges that the algorithm successfully overcame. Figures 7 and 8 show the successful tracking despite these issues.

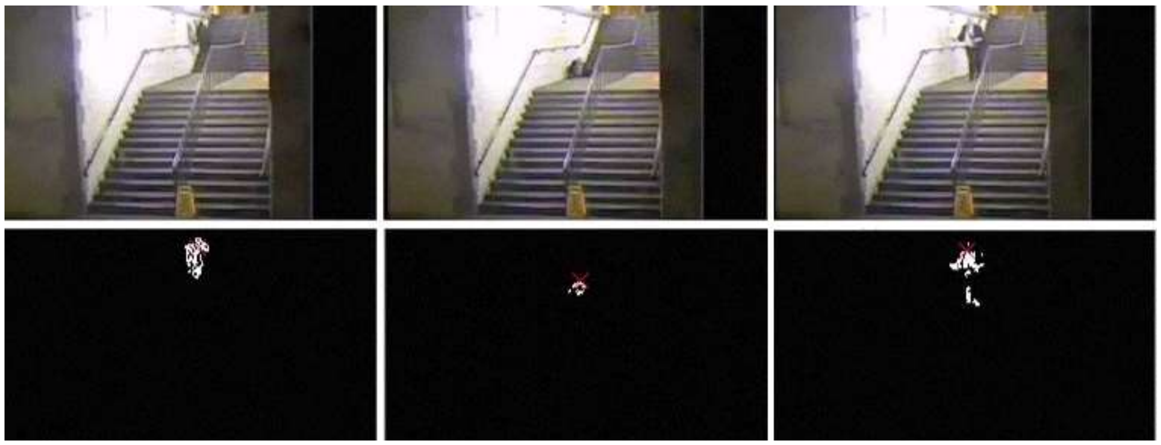

Figure 7: Demonstrating successful tracking and recognizing recovery after a fall

The figure above shows the algorithm accurately tracking the subject as they initially fall down the stairs and then stand. Stairs are a high risk environment for falling so accurate tracking and monitoring for staircase scenarios is vital. Figure 7 depicts low quality images and while the subject is not fully visible, the algorithm sees and locates enough of the person's head and body to accurately detect them fall and then also recover back to a standing position.

Figure 8 demonstrates another practical application. Nursing homes are frequently understaffed and there are known issues and even lawsuits pertaining to injuries sustained by people who fell and were not found for days [20].

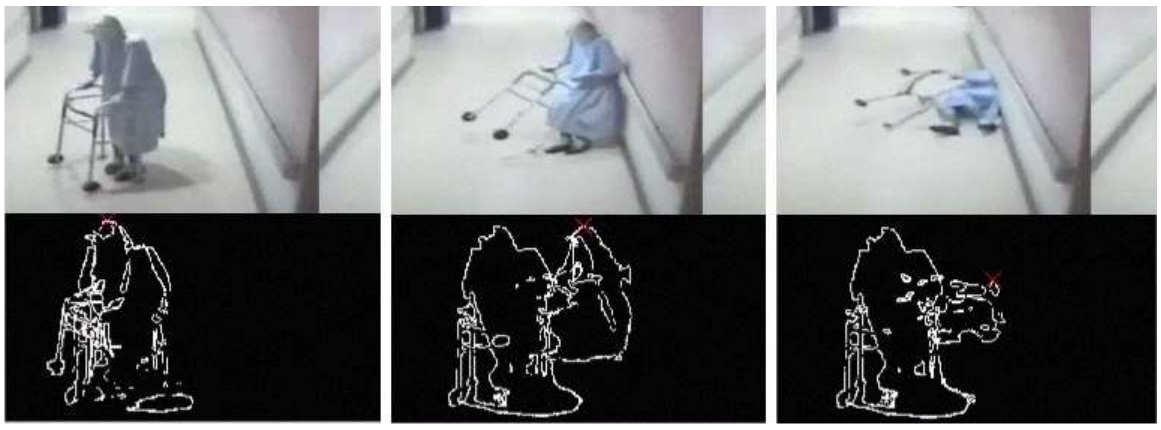

Figure 8: Elderly fall in a nursing home

Figure 8 depicts a shadow of the person that remains throughout the person falling. Typically, this shadow does not remain as the reference frame for background subtraction updates. However, it was intentionally left in for this scenario to imitate the situation where a person falls into something as they fall. Many scenarios will involve the person falling into furniture or other objects. As discussed earlier, to allow the person to remain in the foreground when they fall, the background stops updating once the acceleration reaches the threshold. This means that as they fall, if they bump into something after the background ceases to update, a shadow of that object will remain. The result of Figure 8 shows that due to the robustness of the tracking method, the algorithm successfully tracks the head as it falls to the floor despite the large amount of noise present.

For quantitative results, the following set of graphs, Figures 9-13, depicts the acceleration values calculated in the algorithm versus manually calculated values for walking, sitting down, and falling. For the walking and stand-to-sit scenarios, values were calculated for the person being near to the camera and farther away from it. For the stand-to-sit, the subject sat down from the standing position, stood up, and repeated this a second time.

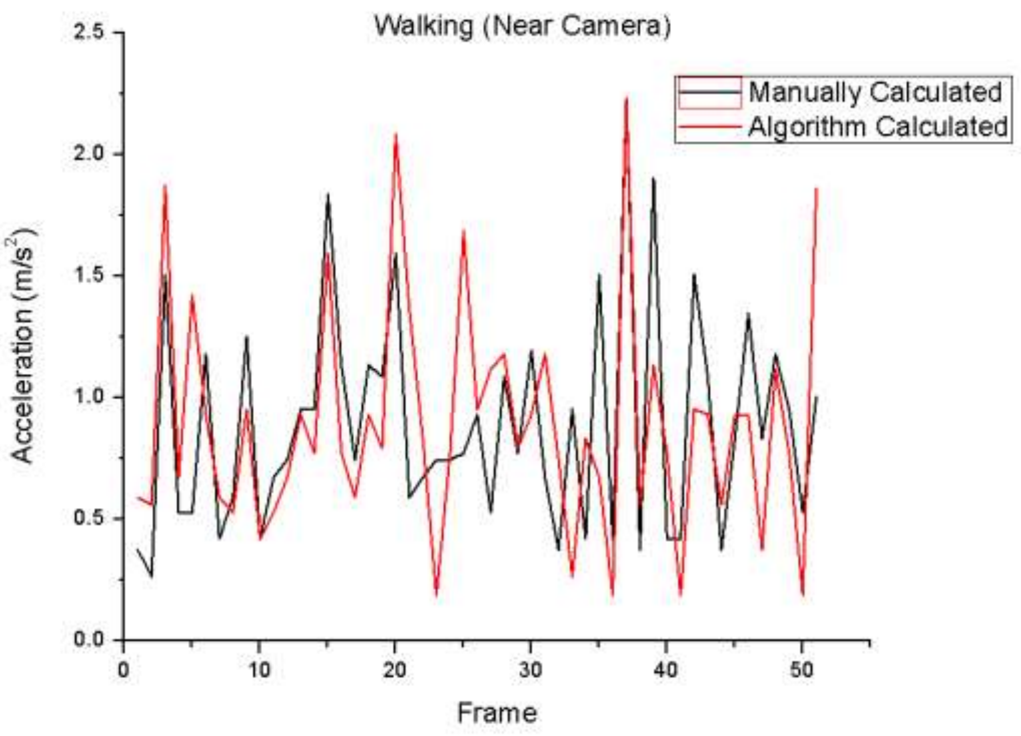

Figure 9: Calculated acceleration values for normal walking pace (near camera)

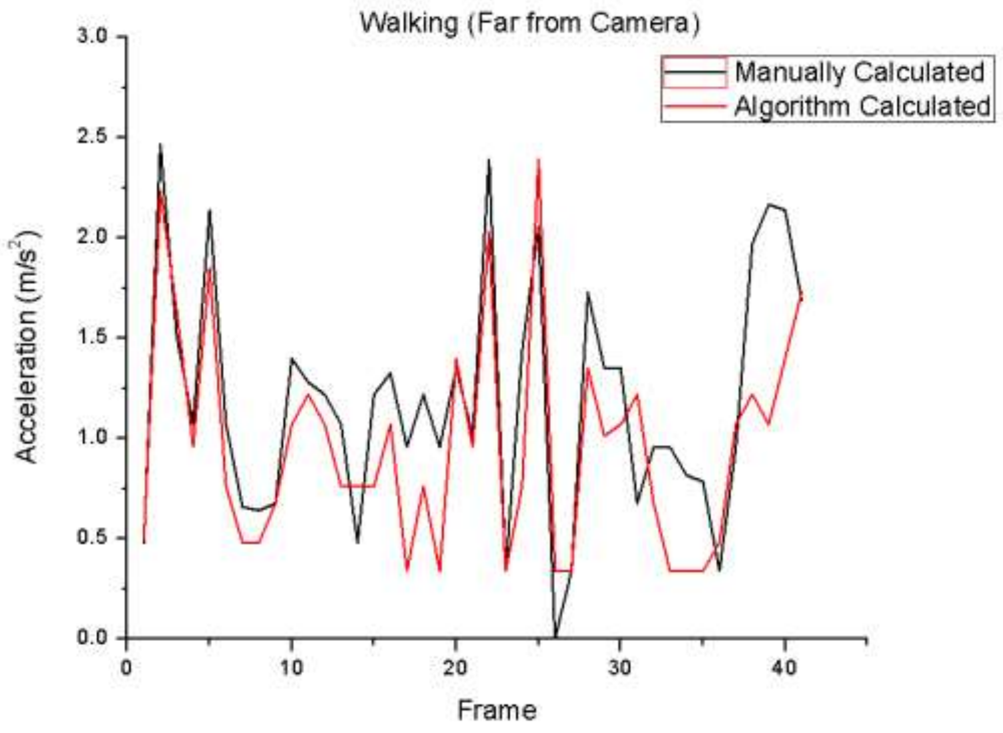

Figure 10: Calculated acceleration values for normal walking pace (far from camera)

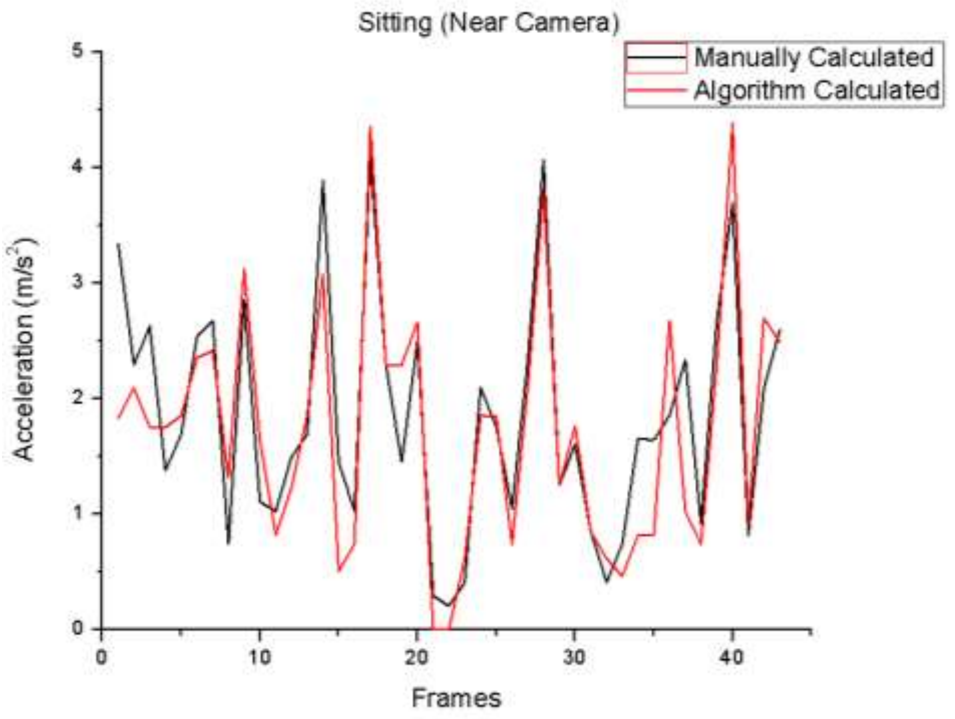

Figure 11: Calculated acceleration values for stand-to-sit (near camera)

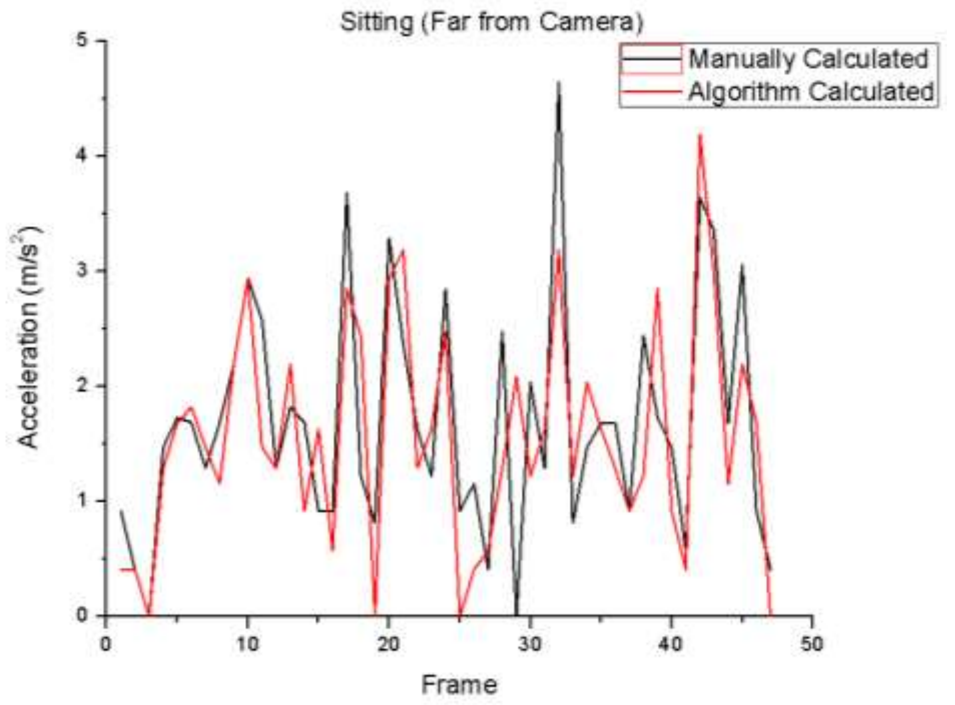

Figure 12: Calculated acceleration values for stand-to-sit (far from camera)

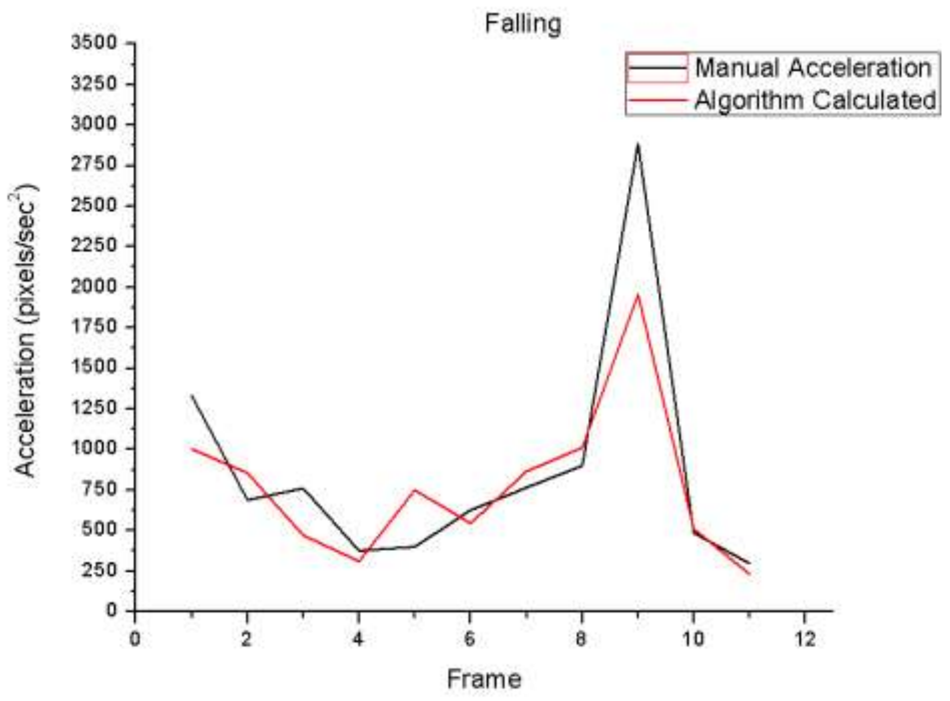

Figure 13: Calculated accelerations values during a fall

The acceleration values during the falling scenario are not converted to m/s^2 and instead left in pixels/s^2. During monitoring, this reduces computation time since the scaling factor would need to be adjusted continuously as the person moves throughout the frame. The relative change in the trend determines a fall indication so the exact values are not significant. Because the person is relatively stationary during the sitting scenarios and remaining the same distance from the camera at all times during the walking scenarios, these values were converted so that they could be compared to accelerometer values later. These accelerometer comparisons are shown in the next section.

The acceleration trends above for walking at a normal pace, sitting, and falling all match closely for both the values calculated in the algorithm and calculated manually with slight differences occurring due to manually choosing points a few pixels off from the points located in the detection function. This is especially true for the values associated with falling due to the nature of falling and how quickly the acceleration changes.

However, for purposes of fall detection, the trends and the relative changes in acceleration from one frame to the next are what is important. For both the algorithm and manually calculated values, the peak acceleration during the falling both reach above the set threshold of 1500 pixels/sec to accurately indicate the fall.

4.2 TUG Results

The results for comparing the acceleration values from the IMU sensors and from the detection algorithm are shown in Figures 11-14 below. The compared actions are walking and sitting, both at two set distances from the camera view.

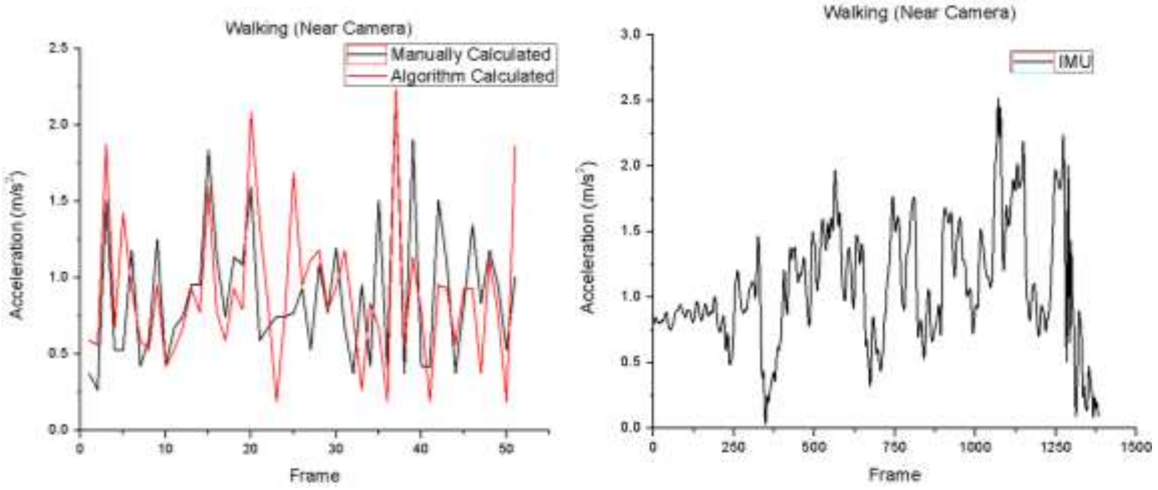

Figure 11: Calculated acceleration values for normal walking pace (near camera)

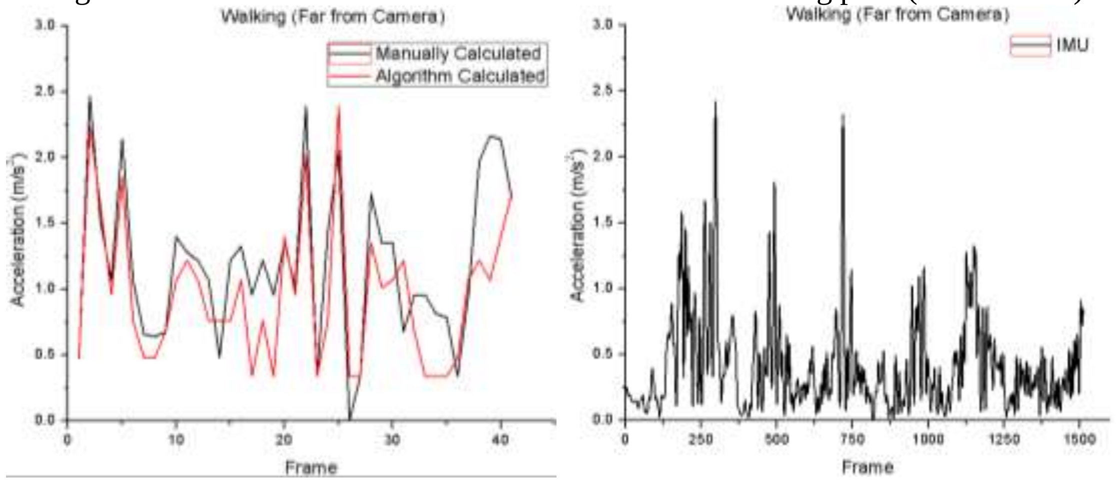

Figure 12: Calculated acceleration values for normal walking pace (far from camera)

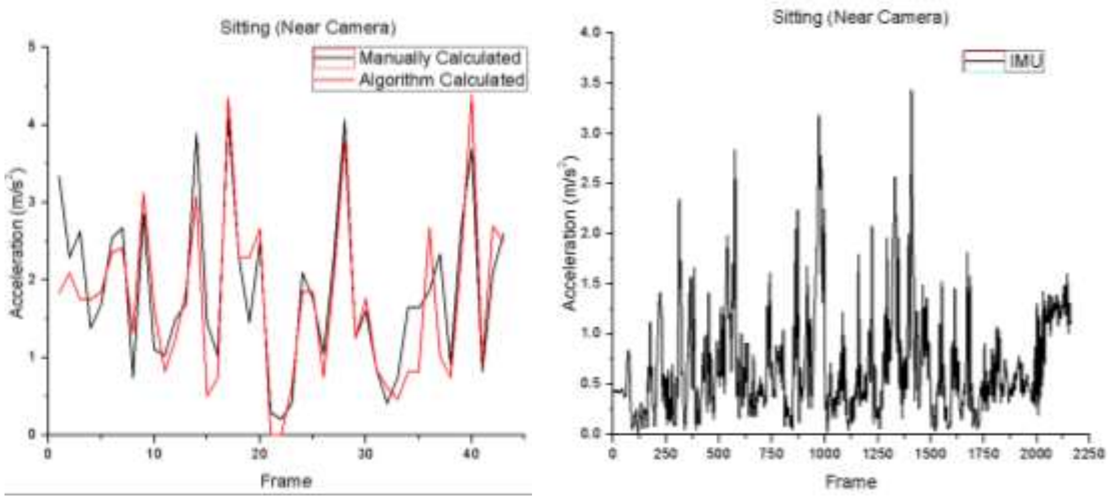

Figure 13: Calculated acceleration values for sitting (near camera)

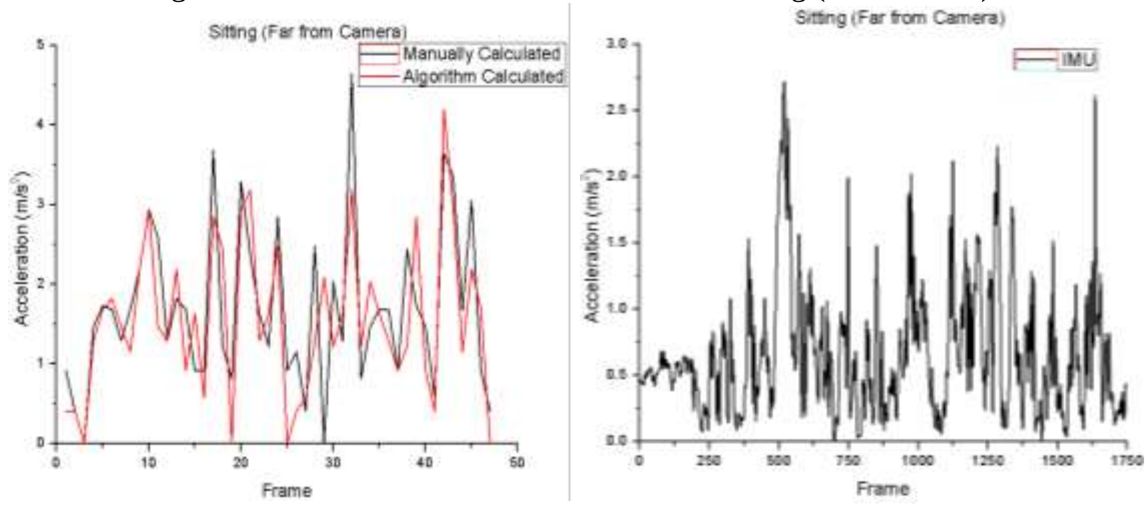

Figure 14: Calculated acceleration values for sitting (far from camera)

Like the fall detection results earlier, the values from the IMU sensors and calculated in the algorithm have matching trends but the exact values differ. The following reasons explain this. Most notably, the IMU sensors measure acceleration 128 times every second while the frame rate for the camera is only 30 $\frac{frames}{second}$. This alone will yield less accurate results. Also, the IMU sensors measure acceleration in three planes while the camera view is two dimensional and therefore so are the acceleration values. Furthermore, due to the sensitivity of the IMU sensors, they are susceptible to noise, even after passing the data through a low pass filter.

CHAPTER 5: DISCUSSION

5.1 Fall Detection

The developed fall detection algorithm was determined to successfully detect falls. It was tested in multiple scenarios including difficult environments such as stairs and partial occlusions. Also, because of the built in checks, it was successfully able to distinguish between an almost fall, a fall with recovery, and a fall with no recovery. This aids in eliminating false positives, as an alarm will only be triggered if a person falls and is unable to stand back up.

One limitation in the accurate detecting of falls is dealing with low level lighting. Lower level lighting leads to increased noise that is more difficult to filter. Natural light from a window with no artificial light source was deemed sufficient as long as the person's head does not remain in a shadow. If in a shadow, the algorithm was still successful in tracking the person's head, but not as consistently. However, because the algorithm depends on weighted trends, the low level lighting only affects the detection results to a limited extent.

To quantify results for fall detection accuracy, acceleration values calculated by the algorithm were compared with those calculated manually. It was found that the trends matched well while the actual values were similar. This comes from the way the algorithm locates the top of the head and the way the human eye does. The method used for fall detection locates all of the points on the uppermost part of the head and then uses the average of these positions. Doing it manually, the top point of the head is located visually which leads to the minor discrepancies. However, because the values were similar and the trends matched, the results of the fall detection algorithm were considered accurate and successful.

5.2 TUG comparison

This algorithm was also compared to IMU accelerometer data for possible clinical use. It was found that acceleration value trends for both the accelerometer and from the fall detection algorithm were similar. However, no significant differences were found between the values obtained with the person close in view to the camera and with the person farther away in view to the camera.

The assumptions used and the approximations made lead to these inaccuracies. One approximation was that when converting the pixel distance to physical distance, each pixel was approximated to correspond to equal physical distances despite pixel distortions as the person moved throughout the frame. The pixels closer to the edges of the frame will correspond to a slightly larger distance. The person's input height also affects the conversion approximation. To calculate the pixels per inch or pixels per meter, the person's actual height is compared to their height in pixels. Because a person's height is typically rounded to the nearest whole number and people do not always stand erect like they would if they were being directly measured at the doctor's office, the input height may not be exact. This leads to discrepancies, and while these discrepancies may seem slight, they are significant enough to affect accurate measurement conversion and subsequent acceleration calculations.

Another key assumption made is that the point on the head used in calculations is the exact same point in subsequent frames. Due to the nature of the algorithm, this is not always true. With motion capture systems, marker sets are placed on the body and remain relatively stationary throughout the process. IMU sensors are also placed on the body and remain relatively stationary. The fall detection algorithm, however, locates the middle point of the top of the head in each frame. While not significantly different, this point may not be the exact one from frame to frame. Although the distance between two located points may not be large, the continuous small differences when used in calculations contribute to less accurate values.

Noise and the lower framerate of the camera also contribute to less accurate calculations. Despite using a filter, some noise occasionally remains. While this does not affect the fall detection accuracy, as there are checks in place to overcome this such as including a weighted average, it can affect the acceleration values obtained for comparison with the IMU sensor. As for the framerate, the IMU sensors used record values 128 times every second while the framerate of the camera is only 30 frames every second, leading to significantly larger spaced and therefore fewer data points. This higher framerate of the IMU sensors makes them much more susceptible to minor changes in acceleration.

Another factor attributing to the higher sensitivity of the sensors is that they record values in all three dimensions. Because one of the objectives of this research was to develop an algorithm to be used with any video camera, a depth camera was not used. This means that the acceleration values could only be calculated in two dimensions using the algorithm. Modifying the algorithm to be used with a depth camera, and therefore including the third dimension would positively affect the value comparisons. While the significance of this effect is not likely to be high, further studies would need to be performed to gain a proper conclusion.

Because of the exact acceleration values obtained using the algorithm were not consistently close to those obtained with the IMU sensors, the algorithm would not be a suitable replacement for the sensors. Modifications to the algorithm, including adapting it for use with a depth camera, could increase its efficacy.

5.3 Practical Applications

Although it was found that the algorithm is not suitable to replace accelerometers if accurate acceleration values are desired, it is useful and reliable for determining the changing trends. Many times when performing motion analysis, key frames are found to determine when desired events occurred such as when a subject begins walking or when they sit down. Because these key frames are determined by specific changes in data, a general trend is all that is needed for an accurate event detection. This is why the algorithm is effective for fall detection despite not accurately calculating the person's actual acceleration. It identifies and makes use of the deviations in the changing acceleration data.

The algorithm is also applicable for the initial intended purpose as a monitoring system. Many people have security cameras in place at their home in case of potential burglaries. There is even an app already developed that alerts the homeowner if someone is detected in the house and allows them to see a live video stream on their phone no matter where they are. In the guise of this application, the algorithm developed in this study could be applied. If and when the algorithm detects and confirms a potential fall, an alert could be sent to a caretaker's phone and they could then check the live video to see if everything is alright. In addition to home monitoring, the same concept could be applied to workplace monitoring. Workers often work alone in potentially hazardous conditions, and having a system like this in place could save lives. Even more than homes, most workplaces have security cameras already in place; so making use of these already in place systems would be convenient.

This algorithm is not limited to just monitoring. This algorithm has definite applications in clinical settings despite its limitations. Although it is not desirable for use in lieu of physical accelerometers, for studies with prerecorded video where accelerometers were not used, this algorithm could be applied as a general assessment. Also, though not implemented in this research, a template matching function was also developed that could easily be incorporated for aiding in event detection and also for posture detection.

5.4 Further improvements

Limiting this study is the lack of available scenarios to test. Acceleration values were able to be validated and prerecorded videos of various falling scenarios were found and tested, but it was not possible to test every scenario that may happen in all of the potential applications. Future work lies in finding ways to test more possible events and developing a phone application for people to download and use.

Further improvement is also needed in making the application more viable for clinical use. This includes quantifying the error in the acceleration calculations. IMU sensors themselves have known inaccuracies. In order to quantify and determine how suitable this detection algorithm is for acceleration calculations, it should be compared with values obtained from a motion capture system since these systems have less error.

Continuing with the clinical applications, because the acceleration trends are comparable between the IMU data and the algorithm calculations, future work comparing the two methods for fall risk assessment is warranted. Risk assessment is already determined using the IMU and motion capture systems. It would be beneficial to determine how effective the detection algorithm could be when modified and applied to determine fall risk.

CHAPTER 6: SUMMARY AND CONCLUSION

6.1 Brief summary

This research proposed a new algorithm to effectively detect falls using nothing more than a single video camera with no need for it to be a depth camera. This was accomplished by creating a series of functions that segmented the process into logical steps. First, the background is subtracted from the image leaving only the moving foreground. Next, an image gradient is calculated to distinctly define the edge of the person to make the locating and tracking portion more effective. The head is then located, its location validated, and then it is tracked as the person moves in and out of the frame. As the person is tracked throughout the frame, the acceleration of the person's head is continuously calculated and monitored. An initial indication that the person has fallen comes when this acceleration value reaches a certain threshold. If and when this threshold is reached, the algorithm goes into a function to calculate the relative change in head height. This height change goes through a series of checks to make a confirmation that the person has indeed fallen. This series of checks is also used to determine if the person has been able to stand up even if they did initially fall.

6.2 Conclusions

This algorithm is robust and it runs efficiently with little computational expense. It overcomes certain challenges such as dealing with partial occlusions and stairs, and can be used with any camera, either greyscale or color, that may already be in place. However, there are also some limitations. First, it is not designed to allow for multiple people to be in the frame. This is not an issue for the initial purpose that was considered, monitoring people who are alone and have a high risk of falling. Low light levels may affect accuracy and efficiency, but natural light has been deemed sufficient as long as the person does not spend extended periods of time in shadows. Also, although the algorithm is successful in dealing with partial occlusions, if the person falls behind something and is entirely missing from the frame, the algorithm will be unable to recognize this and will instead assume that they simply left the frame of their own accord.

The algorithm has also shown an indication as a viable aid in clinical fall risk assessments. Because of this potential for fall risk assessment and its effectiveness in fall detection, the algorithm displays a potentially positive impact on the quality of life for many individuals. It can help to reduce their cost of living and significantly increase their self-confidence and well-being.

REFERENCES

[1] Falls Among the Elderly. Centers for Disease Control and Prevention.2013.

[2]US Census Bureau

[3] http ://ww w.aplaceformom.com/senior-care-resources/articles/elder-care-costs. 2015

[4] Greene, B.R. Quantitative Falls Risk Assessment Using the Timed Up and Go Test. IEEE Transanctions on Biomedical Engineering. 2010.

[5] Zheng, N. et al. "Quantifying Risk Level for Fall Using the Inertial Measurement Unit." University of North Carolina at Charlotte.

[6] Bagala, F., Cappello, A., Ciari, L. et al. 2012. "Evaluation of Accelerometer Based Fall Detection Algorithms on Real-World Falls." PLoS ONE. 7(5).

[7] Nirjon, S., Stankovic, J. 2012. "Kinsight: Localizing and Tracking Household Objects Using-Depth Camera Sensors." Distributed Computing in Sensor Systems. P. 67-74.

[8] Remi, T., Bernard, M. 2007. "Probabilistic Matching Algorithm for Keypoint Based Object Tracking Using a Delaunay Triangulation." Image Analysis for Multimedia Interactive Services. p 67-74.

[9] Desa, S.M., Salih, Q.A. 2004. "Image Subtraction for Real Time Moving Object Extraction." IEEE International Conference on Computer Graphics, Imaging and Visualization.

[10] Zhang, R., Ding, J. 2012. "Object Tracking and Detecting Based on Adaptive Background Subtraction." Procedia Engineering 29. p. 1351-1355.

[11] Verma, R., Ali, J. 2013. "A ComparativeStudy of Various Types of Image Noise and Efficient Noise Removal Techniques." International Journal of Advanced Research in Computer Science and Software Engineering 3(10). p. 617-622.

[12] Li, X., Bilgutay, N. 1993. "Wiener Filter Realization for Target Detection Using Group Delay Statistics." IEEE Transactions on Signal Processing 41(6). p. 2067-2074.

[13] Maini, R., Aggarwal, H. "Study and Comparison of Various Image Edge Detection Techniques." International Journal of Image Processing 3(1).

[14] Giebel, J., Gavrila, D.M., Schnörr, C. 2004. "A Bayesian Framework for Multi-Cue 3D Object Tracking." EECV 3024.

[15] Kim, D., Kim, D. “A Novel Fitting Algorithm Using the ICP and the Particle Filters for Robust 3D Human Body Motion Tracking.” VNBA. p. 69-76.

[16] Briechle, K., Hanebeck, U. 2001. “Template Matching Using Fast Normalized Cross-Correlation.” Optical Pattern Recognition XII 4387.

[17] Liu, M. et al. 2010. “Fast Directional Chamfer Matching.” TR 045.

[18] Hadju, A., Pitas, I. 2007. “Optimal Approach for Object-Template Matching.” IEEE Transactions on Image Processing 16(8). p. 2048-2057.

[19] Borgefors, G. 1988. “Hierarchical Chamfer Matching: A Parametric Edge Matching Algorithm.” IEEE Transactions on Pattern Analysis and Machine Intelligence 10(6). p. 849-864.

[20] http ://ww w.nursing-home-neglect.com/assisted-living-center-sued-after-residents-fall-went-unnoticed-for-days. 2015

APPENDIX: MATLAB CODE

```
% Beth Schlegel
% 2014-10-13
% Todo Pasa

%%%%%%%%%%%%%%%%%%%
%%% Main script %%%
%%%%%%%%%%%%%%%%%%%

clear
clc

%% Todo Pasa Part One

% Initializing Counters and Thresholds
fs=5;            % Spacing between captured frames (ie k=1:5:length)
p=0;             % Counter for X and Y points used in Acceleration
a=55;           % Beginning Frame Number
b=55;           % Ending Frame Number
j=1;        % Counter used for updating background reference frame
f=1;             % Initial f to pass into BSAcFunction
AccCounter=0 % Counter for displaying head distances when
               % AccThreshold is reached

AccThreshold=1500;         % 11-19 Changed from 2100 to 1500
HeadDistanceThreshold=45;  % Initial start

%%% Inputs set by user/calibrated
ActualHeight=65  % Height of user. Right now it is me and I am
                   % 65 inches tall. Inputted by user
MOTRH=250;       % Pixel height of user while standing in the
                   % middle of the room. Calibrated initially
framerate=30;    % Frames per second
%%%

xh=-.5;   % Initial xh point having a value that xh will never
            % actually return. Used to tell the program if it's
            % the first frame or not
SkipCounter1=0;
ST=0;             % Skip Threshold
AccPos=2;         % Initializing it to 2 because the first 2 entries
                    % will be manually entered as 0
Acceleration(1)=0;% Setting the first 2 entries to 0 to fill in spots
                    % for the table
Acceleration(2)=0;%
AccFrame=1;
Wait=0;     % Wait gets set to one if AccThreshold is reached.
              % This is passed into the BSAcFunction so that it
              % will stop updating the background. If the
              % background continued to update, a person lying
              % still on the ground after falling would disappear
              % into the background making determinations
              % impossible
Seconds=2;  % Initially check head distance 2 seconds later
```

```
hd=0;       % Head distance place holder
psf=0;      % Possible standing frame place holder
Fallen=0;   % Initializing fallen variable

%% Todo Pasa Part Two
tic
% Importing the video
[Length,height,width,mov]=VideoImporting('OoeyGooey.avi'); % 1 %
toc
%% Todo Pasa Part Three

% Looping through the frames
for k=a:fs:b
     j=j+1;           % Counter for background reference frame
     p=p+1;           % Counter for saving x and y positions
    % Background Subtraction
    FrameInput=k;% Unless otherwise specified, the current frame
                 % is used as the input frame
    SC=0; % Currently not determining if a person is standing and
            % what their standing height is

[Target,fg,f,bg]=BSAcFunction(Length,height,width,mov,k,j,f,Wait,...
        AccCounter,AccFrame,FrameInput,SC);              %%%% 2 %%%
    Target=im2double(Target);

    % Finding Head Using Location Function and Gradient Magnitude
    [GM,GD]=imgradient(Target);
    GradientTarget=im2bw(GM);
    figure(1)
    imshow(mov(k).cdata)
    hold on;
[xh,yh,skip,SkipCounter]=LocationFunctionH(GradientTarget,...
    xh,height,width);                                %%%% 3 %%%

    % Continues on to the next iteration if too many invalid
      % points are found so that it doesn't get stuck and freeze
    if skip==1
       SkipCounter1=SkipCounter1+SkipCounter;
       continue
    end

    % Saving frame and head location values to be later
      % displayed in a table
    frame(p)=k;
    XPosition(p)=xh;
    YPosition(p)=yh;

    %% Todo Pasa Part Four
    % Acceleration

    % Assigning x and y positions to determine acceleration
    if p>2
       AccPos=AccPos+1;
       XM3=XPosition(p);
       XM2=XPosition(p-1);
       XM1=XPosition(p-2);

       YM3=YPosition(p);
       YM2=YPosition(p-1);
       YM1=YPosition(p-2);

[Acc]=AccelerationFunction(XM1,XM2,XM3,YM1,YM2,YM3,framerate,fs);%4%
       Acceleration(AccPos)=Acc;

        % If the acceleration is greater than the threshold
        if Acc>AccThreshold
            AccCounter=AccCounter+1;
            AccFrame=k-fs;

             Wait=1;% Set Wait=1 so that background stops updating
              if Wait==1
                 Seconds=2;  % Resetting seconds back to its
                               % initial value every time it
                               % reenters the loop
    % Checking to see if person exists in frame attempting to
      % get a standing height from. If acceleration threshold
      % is reached from person walking into frame, there will
      % be no head height to use k-.5*framerate beforehand.
      % If this is the case, just use the head height of the
      % person if they were standing in the middle of the
      % room

                 FrameInput=AccFrame;%Checking to see if there is a
                                       % person standing here
                 SC=1;     % Checking to see if person is standing
                 % Background Subtraction
   [Target,fg,f,bg]=BSAcFunction(Length,height,width,mov,...
       k,j,f,Wait,AccCounter,AccFrame,FrameInput,SC); %2.2%
                 Target=im2double(Target);

                 % Finding the edge
                 [GM,GD]=imgradient(Target);
                 GradientTarget=im2bw(GM);

                 % Finding the head and feet
                 xh=-.5;
[xh,yh,skip,SkipCounter]=LocationFunctionH(GradientTarget,...
     xh,height,width);                          %%%% 3.2 %%%

                 if skip==1          % If no person was found
                    HeadDistanceStanding=MOTRH;%MOTRH-Middle Of
                                                 % The Room Height.
                                                 % Height of them
                                                 % in middle of room.
                    SC=0;
                 else
```

```
                  %%%%%%% STANDING HEIGHT %%%%%%%

                   FrameInput=AccFrame;% Possible standing frame
                                        % half a second before
                                        % acceleration
                                        % reached threshold

                   % Take FrameInput, find head and base, call it
                     % HeadDistanceStanding
                   % Background Subtraction
    [Target,fg,f,bg]=BSAcFunction(Length,height,width,...
        mov,k,j,f,Wait,AccCounter,AccFrame,FrameInput,SC);%2.3%
                   Target=im2double(Target);

                   % Finding the edge
                   [GM,GD]=imgradient(Target);
                   GradientTarget=im2bw(GM);

                   % Finding the head and feet
[xh,yh,skip,SkipCounter]=LocationFunctionH(GradientTarget,...
    xh,height,width);                                 %%% 3.3 %%%

     %%%%% Feb 13, only need head height.
                   StandingHead=yh;

                   %%%%% relative change in head height
                   [yf]=BaseLocationFunction(Target,height);%5%

                   % Finding HeadDistanceStanding
                   [HeadDistance]=HeadDistanceFunction(yh,yf);%6 %
                   HeadDistanceStanding=HeadDistance;

                   SC=0;% Once Standing height is determined,
                          % no longer using standing stuff, so
                          % SC goes back to 0
                end

                %%%%%%% INITIAL FALLEN HEIGHT %%%%%%%

                % Determing height after potential fall
                FrameInput=k+Seconds*framerate;%Inputting frames
                                                %multiple seconds
                                                %later to compare

    [Target,fg,f,bg]=BSAcFunction(Length,height,width,mov,...
         k,j,f,Wait,AccCounter,AccFrame,FrameInput,SC); %2.4 %
                Target=im2double(Target);

                [GM,GD]=imgradient(Target);
                GradientTarget=im2bw(GM);

                xh=-.5; % Resetting xh=-.5 resets the LLimit and
                          % RLimit since it is checking multiple
                          % seconds later
```

```
 [xh,yh,skip,SkipCounter]=LocationFunctionH(GradientTarget,...
       xh,height,width);                              %%% 3.4 %%%
                 FallenHead=yh;
                 if skip==1
                    Wait=0;
                    continue
                 end

   %[yf]=BaseLocationFunction(GradientTarget,height); %5.2%

 [HeadDistance]=HeadDistanceFunction(StandingHead,FallenHead);%6.2%
                 HeadDistanceFallen=HeadDistance;

                 % Converting to the scaled distance
[HeadDistanceConverted]=PixelConversion(HeadDistanceStanding,... %7%
        HeadDistanceFallen,ActualHeight);
                 HDTable(AccCounter)=HeadDistanceConverted;

                 if HeadDistanceConverted > HeadDistanceThreshold-1
                    Wait=0;
                    continue
                 else

                     figure(2)
                     imshow(GradientTarget)

                    %%%%%%% SECOND HEIGHT CHECK %%%%%%%

                    Seconds=Seconds+3; % If initial head distance
                                         % implies fallen, then
                                         % check again 3 seconds
                                         % later
             FrameInput=k+Seconds*framerate; % Inputting frames
                                               % multiple seconds
                                               % later to compare


[Target,fg,f,bg]=BSAcFunction(Length,height,width,...

mov,k,j,f,Wait,AccCounter,AccFrame,FrameInput,SC);%2.5%
                    Target=im2double(Target);

                    [GM,GD]=imgradient(Target);
                    GradientTarget=im2bw(GM);

               xh=-.5;% Again, resetting the LLimit and the RLimit
                          %%% 3.5 %%%
[xh,yh,skip,SkipCounter]=LocationFunctionH(GradientTarget,...
    xh,height,width);
                    FallenHead=yh;

                    if skip==1
                       Wait=0;
                       continue
                    end
```

```
  %[yf]=BaseLocationFunction(Target,height);      %5.3%

  [HeadDistance]=HeadDistanceFunction(StandingHead,FallenHead);%6.3%
                    HeadDistanceFallen=HeadDistance;

  [HeadDistanceConverted]=PixelConversion(HeadDistanceStanding,...
            HeadDistanceFallen,ActualHeight);            %7.2%
                    HDTable(AccCounter)=HeadDistanceConverted;

                    if HeadDistanceConverted < HeadDistanceThreshold

                        figure(3)
                        imshow(GradientTarget)

                       %%%%%%% THIRD HEIGHT CHECK %%%%%%%

                  Seconds=Seconds+3;%If the height is still under
                                     % the threshold after 5
                                     % seconds, then check one more
                                     % time after 3 more seconds
                 FrameInput=k+Seconds*framerate; %Inputting frames
                                                  % multiple seconds
                                                  % later to compare

    [Target,fg,f,bg]=BSAcFunction(Length,height,width,mov,...
           k,j,f,Wait,AccCounter,AccFrame,FrameInput,SC); %2.6%
                       Target=im2double(Target);

                       [GM,GD]=imgradient(Target);
                       GradientTarget=im2bw(GM);

                       xh=-.5;
                       %3.6%
[xh,yh,skip,SkipCounter]=LocationFunctionH(GradientTarget,...
    xh,height,width);
                       FallenHead=yh;

                       if skip==1
                          Wait=0;
                          continue
                       end

           %[yf]=BaseLocationFunction(Target,height);

 [HeadDistance]=HeadDistanceFunction(StandingHead,FallenHead); %6.4%
                       HeadDistanceFallen=HeadDistance;

 [HeadDistanceConverted]=PixelConversion(HeadDistanceStanding,...
          HeadDistanceFallen,ActualHeight);    %7.3%
                       HDTable(AccCounter)=HeadDistanceConverted;

                    if HeadDistanceConverted < HeadDistanceThreshold
                          k
                          figure(4)
                          imshow(GradientTarget)
                          fprintf('Fallen? \n')
               Fallen=input('Press 1 if fallen, press 0 if not:');
                          if Fallen==1
                             xh=-.5;
                          fprintf('Golf Papa down. Oscar Mike \n')
                             break
                          else
                             xh=-.5;
                             Wait=0;
                         fprintf('All good on the home front \n')
                             continue
                          end
                       else
                           figure(3)
                           imshow(GradientTarget)
                       end

                    else
                       figure(4)
                       imshow(GradientTarget)
                       break
                    end
                 end
              end

           AccelerationFrame(AccCounter)=k;
           AccThreshold(AccCounter)=Acc;

        end
    end
end

if exist('frame')
    if exist('XPosition')
        if exist('YPosition')
            if exist('Acceleration')
% Creating a table to display frame number and x and y positions
frame=transpose(frame);
XPosition=transpose(XPosition);
YPosition=transpose(YPosition);
Acceleration=transpose(Acceleration);
if length(Acceleration)==length(YPosition)
    T=table(frame,XPosition,YPosition,Acceleration)
else
    T=table(frame,XPosition,YPosition)
end
            end
        end
    end
end

if AccCounter>0
    AccelerationFrame=transpose(AccelerationFrame);
    HeadDistance=transpose(HDTable);
    AccThreshold=transpose(AccThreshold);

    T2=table(AccFrame,AccThreshold,HeadDistance)
end
  toc
```

```
% Beth Schlegel
% 2014-11-01

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Mean-Shift Video Tracking %%%
%%% Modified version originally written by Sylvain Bernhardt %%%
%%% July 2008 %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Description
% Import a AVI video file from its 'path'.
% The output is its length, size (height,width) and the video
  % sequence read by Matlab from this AVI file.
% [lngth,h,w,mov]=Import_mov(path)

function [lngth,h,w,mov]=VideoImporting(path)
infomov=VideoReader(path);
lngth=infomov.NumberOfFrames;
h=infomov.Height;
w=infomov.Width;
mov(1:lngth)=struct('cdata',zeros(h,w,3,'uint8'),'colormap',[]);


% Read one frame at a time.
for k=1:lngth
    mov(k).cdata=read(infomov, k);
   %mov(k).alpha = read(infomov, k);
end

%% General Notes
```

```
% Beth Schlegel
% 2014-06-17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Background Subtraction Function %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
function[Target,fg,f,bg]=BSAcFunction(Length,height,width,mov,...
    k,j,f,Wait,AccCounter,AccFrame,FrameInput,SC);
%%% BSAc-BackgroundSubtractionAcceleration

if j==1
    f=1;
elseif rem(j,6)==0
    f=j;
end

if Wait==0
   bgFrame=k-2; % If AccThreshold has not been reached,
                 % background continues to update
   FrameInput=k;% Using the current frame as comparison until
                 % threshold is reached. Then using frames
                 % in advance
   if bgFrame<1
       bgFrame=1;
   end
elseif Wait==1
    if SC==0
       bgFrame=AccFrame-60;   % If AccThreshold has been reached,
                              % background stops updating and uses
                              % bg as frame where Threshold was
                              % first reached

    else
       bgFrame=AccFrame-325; % If checking if person is standing
                               % and deciding their height, use an
                               % earlier background reference frame

    end
    if bgFrame < 1
          bgFrame=1; % Making sure there are enough already
                      % processed frames
    end
end
MovLength=length(mov);
if FrameInput > MovLength
   FrameInput=MovLength;
end

% Setting Frame Variables
thresh=40;             % Threshold for maximum pixel difference

bg=mov(bgFrame).cdata;%reads in 1st frame as background frame
                       %initially and then updates using a
                       %new frame to account for furniture movement
```

```
fg = zeros(height,width); %Creates an empty array for the foreground

% Processing Frames
fr=zeros(height,width,3);
tic
    fr = mov(FrameInput).cdata;            % reads in frame
    fr_diff = abs(double(fr) - double(bg));

    for j=1:width
        for k=1:height
            if ((fr_diff(k,j) > thresh))   % Determine if the
                                             % difference at each
                                             % pixel is greater
                                             % than the threshold
                fg(k,j)=fr(k,j);%Set pixel equal to image pixel
                                 %of current frame
            else
                fg(k,j)= 0;%If not greater than the threshold,
                            %set the pixel equal to zero
            end
        end
    end

    %Target=fg;
   Target=wiener2(uint8(fg),[16 12]);       % filter

   %% General Notes
   %Added Adaptive BS with b=mov(k-2).cdata
```

```
% Beth Schlegel
% 2014-03-11 Original
% 2014-05-09 Update
% 2014-10-24 Update

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Function for finding head location for each frame %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Modification of the original Location Function
% 10-24 Modification. Adding a check to see if the found head
  % position is valid
% 10-29 Modification. Adding search limits
%%
function [xh,yh,Skip,SkipCounter]=LocationFunctionH(alpha,...
    xh,height,width)
%% Initial Parameters and Setting Search Limits
xhP=xh;     %Setting the last found xh as the xhprevious
             %to compare the current frame with the previous frame
Skip=0;             % Initializing Skip variable
SkipCounter=0;      % Initializing Skip Counter
invalid=0;          % Initializing invalid counter
a=1;           % Counter for the yh loop. Used to continue
                 %searching down if the the original found point
                 % is determined to be invalid
q=0;           % q is set to 1 if the found xh point is determined
                 % to be valid. Otherwise, q remains 0, and the
                 % function reruns through the loops beginning at
                 % row a+1 (the row below where it originally broke
                 % out of the loop when yh was first found)
LLimit=1;     % Search the entire bounds of the image unless
                % narrowed for an invalid point
RLimit=width;
y=0;          % Initially setting y=0 so that the LLimit and RLimit
                % will include the entire bounds of the image

% xh is initially set to -.5. Once it runs through this function,
  % a new value will get assigned. Only once the program has found
  % two points can it do a comparison so the comparison at the end
  % of the function will only run if n=1
if xh==-.5
    n=0;
else
    n=1;
end

%% Performing the Search and Determining xh
while q==0     % While the gap between the current and previous
                 % xh locations remain too far apart
    if y==1    % If an invalid point is found, the search field for
                 % the next point is narrowed
        LLimit=xhP-30;
        RLimit=xhP+30;

% Making sure the limits stay within the bounds of the image
        if LLimit<0
```

```
           LLimit=1;
        end

        if LLimit==0
            LLimit=1;
        end

        if RLimit>640
           RLimit=640;
        end
    end

    for i=a:1:height    % Initially set to 1. Set to the point
                          % after the initial break if the initial
                          % point is invalid and finds the next point
        if max(alpha(i,LLimit:RLimit))>0
            yh=i;
            if yh>i-1
                break
            end
        end
    end

    if ~exist('yh')
        xh=-.5;    % Resetting xh to -.5 to restart the
                    % search from the beginning
        yh=-.5;    % Setting it to -.5 so it has a value to return
        Skip=1;
        break      % Breaking out of while loop
    end

    YC=0;
    for k=LLimit:1:RLimit
       YC=YC+1;
        if max(alpha(i,k))>0
           xh1(YC)=k;

        else
           xh1(YC)=0;
        end

        if y==1
           ALI(YC)=xh1(YC);
        end

    end

    % Using only non-zero values of xh
    pp=length(xh1);
    j=0;
    for p=1:pp
        if xh1(p)>0
            j=j+1;
            xh(j)=xh1(p);
        end
```

```
end

if y==1
   if exist('ALI')==1
      pp=length(ALI);
      j=1;
      for p=1:pp
          if ALI(p)>0
             xhNew(j)=ALI(p);
             j=j+1;
          %else
         %   xhNew(j)=0;
          end
      end
   end
end

% Returning only the middle point of the xh1 range
LastXh1=length(xh);
xh=ceil(abs(xh(1)+xh(LastXh1))/2);

if y==1
    if exist('xhNew');
       LastxhNew=length(xhNew);
       xhNew=ceil(abs(xhNew(1)+xhNew(LastxhNew))/2);
       xh=xhNew;
    end
end

% Only comparing if there is a previous one to compare with
if n==1
    if abs(xh-xhP)>35      % Value subject to change
       q=0;
       y=1; % If y=1, the next search will narrow the x limits
       invalid=invalid+1;

       if invalid >1
          a=yh+1;
       end

       if invalid == 25
          Skip=1;
          SkipCounter=SkipCounter+1;
          q=1;           % Set q=1 to break out of while loop
          xh=-.5;        % Resetting xh to initial value
       end
    else
        q=1;
        y=0;
        invalid=0;  % Resetting the invalid counter if a
                      % valid point is found
    end
else
    q=1;            % If n=0, it means it was the first
                      % frame, so q needs to be set equal
                      % to 1 to break out of the while loop
    end
end

 %% General Notes

 % Still need to develop a way to validate the initial point.
   % Currently, it runs under the assumption that the first
   % point is valid, so the next point gets compared to it.This
   % will not work if the initial points ends up being an outlier.
   % 10-24-14.FIXED

 % Also, still need to add in what to do if person is not in frame
 % andtherefore no xh is found. FIXED 11-15
% Beth Schlegel
% 2014-06-17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Fall detection based on acceleration using no markers or sensors%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
function [Acc,Accx,Accy]=AccelerationFunction(XM1,XM2,XM3,YM1,...
    YM2,YM3,framerate,fs)

t=fs*(1/framerate);        % Time between measured frame points

vx1=abs((XM2-XM1))/t;      % Initial velocity in the x direction
vx2=abs((XM3-XM2))/t;%Velocity at the next point in the x direction

Accx=abs((vx2-vx1))/t;     % Acceleration in the x direction

vy1=abs((YM2-YM1))/t;      % Initial velocity in the y direction
vy2=abs((YM3-YM2))/t;%Velocity at the next point in the y direction

Accy=abs((vy2-vy1))/t;     % Acceleration in the y direction

Acc=ceil(sqrt(Accx^2+Accy^2));  % Acceleration magnitude

end

%% General Notes
```

```
% Beth Schlegel
% 2014-10-17

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function for finding the surface base (where the person's %%%
%%% feet are or where the floor is if they are not one and the same %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [yf]=BaseLocationFunction(Target,height)

for j=height:-1:1
    if max(Target(j,:))>0
        yf=j;
    else
        yf=0;
    end
    if yf>j-1
        break
    end
end
```

```
% Beth Schlegel
% 2015-01-24

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Finding the distance of the head from the floor %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [HeadDistance]=HeadDistanceFunction(yh,yf);

HeadDistance=yf-yh;
```

```
% Beth Schlegel
% 2015-01-24

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Pixel Conversion for the height of the head from the floor %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
function
[HeadDistanceConverted]=PixelConversion(HeadDistanceStanding,...
    HeadDistanceFallen,ActualHeight)

%HeadDistanceConverted=(AH*HDFallen)/HeadDistanceStanding;
PixelsPerInch=HeadDistanceStanding/ActualHeight;
HeadDistanceConverted=HeadDistanceFallen/PixelsPerInch;
```