VISION BASED ROBOTIC PERSON FOLLOWING USING AN IMPROVED IMAGE SEGMENTATION APPROACH

A Thesis
Presented to the Faculty of San Diego State University

In Partial Fulfillment of the Requirements for the Degree Master of Science in Computer Science

by
Brian Denver Blaine
Summer 2012
The Undersigned Faculty Committee Approves the

Thesis of Brian Denver Blaine:

Vision Based Robotic Person Following Using an Improved Image Segmentation Approach

Mahmoud Tarokh, Chair
Department of Computer Science

Roger Whitney
Department of Computer Science

Gordon Lee
Department of Electrical and Computer Engineering

5-2-2012
Approval Date
ABSTRACT OF THE THESIS

Vision Based Robotic Person Following Using an Improved Image Segmentation Approach

by

Brian Denver Blaine
Master of Science in Computer Science
San Diego State University, 2012

In image based robotic person following a robot attempts to autonomously follow a person based on visual characteristics of the person in an image. The robot's vision system must be able to detect and distinguish the person in a changing environment. Furthermore, the robot's control system must be able to properly follow the path of the person given the location found by the vision system. To distinguish a person from the environment region segmentation is performed on images acquired during person following to detect regions that may represent the person being followed. Regions are found based on their similarity in color to a training model of the person. The regions that are found are then considered for tracking the location and distance to the person for following. The detected regions may represent the entire person, parts of the person, or other objects in the scene. To perform person following well this system depends on evaluating the detected regions correctly to track the location and distance to the person to use for following. The approach presented in this thesis will consider all the regions detected to formulate a location and distance used for person following. The characteristics of each region with respect to the training region are evaluated and regions where the color closely matches the training region and have a larger size make a greater contribution to location and distance values. Performance of this approach is then compared to previous approaches used for person following on this project.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT ............................................................................................................................. iv</td>
</tr>
<tr>
<td>LIST OF FIGURES ................................................................................................................ vii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS ................................................................................................... viii</td>
</tr>
<tr>
<td>CHAPTER</td>
</tr>
<tr>
<td>1  INTRODUCTION .........................................................................................................1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>2  SYSTEM COMPONENTS ..........................................................................................13</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3  PERSON DETECTION ...............................................................................................16</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 2.1. Segway Robotic Mobile Platform ................................................................. 13
Figure 3.1. HSI color cone .......................................................................................... 17
Figure 4.1. Pan PID controller .................................................................................... 28
Figure 4.2. Tilt PID controller .................................................................................... 28
Figure 4.3. Speed PID controller ................................................................................ 30
Figure 5.1. Multiple regions detected example 1 of 4 ................................................. 33
Figure 5.2. Multiple regions detected example 2 of 4 ................................................. 33
Figure 5.3. Multiple regions detected example 3 of 4 ................................................. 33
Figure 5.4. Multiple regions detected example 4 of 4 ................................................. 34
Figure 5.5. Stationary test 1 ....................................................................................... 36
Figure 5.6. Stationary test 2 ....................................................................................... 36
Figure 5.7. Stationary test 3 ....................................................................................... 37
Figure 5.8. Stationary test 4 ....................................................................................... 38
Figure 5.9. Stationary test 5 ....................................................................................... 39
Figure 5.10. Region detections just after training phase ............................................. 40
Figure 5.11. Region detection in bright light ............................................................... 40
Figure 5.12. Region detection in dimmed lighting ....................................................... 40
Figure 5.13. Partial region detection ......................................................................... 41
Figure 5.14. Small partial region detected ................................................................. 41
Figure 5.15. Multiple regions detected 2 of 4 ............................................................ 42
Figure 5.16. Multiple regions detected 1 of 4 ............................................................ 43
Figure 5.17. Multiple regions detected 3 of 4 ............................................................ 43
Figure 5.18. Multiple regions detected 4 of 4 ............................................................ 43
Figure 5.19. Incorrect centroid detection 1 of 3 ......................................................... 44
Figure 5.20. Incorrect centroid detection 2 of 3 ......................................................... 45
Figure 5.21. Incorrect centroid detection 3 of 3 ......................................................... 45
ACKNOWLEDGEMENTS

I would like to express gratitude to Dr. Mahmoud Tarokh for his guidance and patience throughout the process of this thesis. His knowledge and enthusiasm in the coursework inspired me to further pursue study in computer vision and robotics with this thesis. I would also like to thank Dr. Roger Whitney for entertaining my conversations about programming languages and serving on my thesis committee during my time at SDSU. I would also like to acknowledge Dr. Gordon Lee for responding to my thesis committee “cold call” and serving on the thesis committee. Finally, I would like to thank my family and Audrey for their support through this process.
CHAPTER 1

INTRODUCTION

The field of robotics has a history of developing robots that can work both alongside and in place of humans in various environments as seen in [1, 2, 3]. A robot working with a person in these scenarios exhibits a reasonable level of independence to work in the presence of a person. The robot may perform tasks such as carrying supplies, operating as an assistant, or navigating hazardous terrain. In doing these tasks the robot must navigate the environment and operate autonomously. Robotic person following is an application of robots operating together with a person that can be used in various domains.

Person following is the ability of a robot to properly distinguish a person from the environment and follow the path the person takes. The robot must deal with many unknown and changing factors in the environment it is operating in to effectively follow the person. The lighting of the environment may change as the person moves, the terrain traveled may present obstacles or vary, and different lighting conditions will be encountered. These factors make person detection and following complex. The detection algorithms will have to be able to distinguish the person from the environment and from other people. When the person is properly detected, an accurate location of the person will also have to be found to follow the proper path. With the location of the person the following system will have to be able to follow the proper course and speed of the person. The system developed for this thesis addresses these challenges and demonstrates a typical application of robotic person following - to follow a person in an unknown environment with both terrain and lighting variations. The focus of this thesis is to investigate object detection algorithms and implement algorithms for person detection using previous versions of this system as a baseline for improvement.

1.1 REVIEW OF LITERATURE

There are numerous approaches to vision based person following that can be used to detect and classify objects within an image. Vision based approaches may be complemented
by additional sensors including global positioning satellite (GPS) units, lasers, and the use of multiple cameras. Sensors can be combined to provide some information that an approach that uses only image processing techniques cannot provide. For example, GPS information or feedback from a laser may provide an accurate distance to the person from the robot. A sample of research projects using these approaches is given here.

GPS can be used to track the location of objects. For person following, GPS can be used to track the location of the robot and the person. With the location of the robot and person the robot can follow the path of the person by using GPS information to find the path. Using only GPS information would not provide any information about the terrain which is also important. This is the approach used by Nguyen et al, 2004. In this approach the GPS coordinates of the person are transmitted to the robot and the robot follows the path to the person from its current location. Our work differs from Nguyen et al. 2004 in that we do not use GPS sensors since GPS based tracking requires an accurate and available GPS signal that may not be available.

Sidenbladh, Kragik,and Christensen developed an algorithm to follow a person in an indoor environment [4]. This algorithm locates the head of the person in the scene by segmenting pixels that are considered to be skin colors from non-skin colors based on a threshold from a training data set. This set of segmented skin colors is then filtered using spatial properties of the pixels and by noting the typical location of a person's face in the scene. i.e. the upper half of the image. With the initial position of the person's head found their system tracks the movement of the person using a Taylor approximation. The work presented here focuses on tracking the color characteristics of a person's clothing which may include the lower and upper body.

A vision system can be used in conjunction with other systems to implement person following. In one of [5] approaches a laser was used with a camera to implement person following. The laser in this implementation is the primary sensor and the camera is secondary. The laser scan yields blobs that may correspond to the person in the field of view. The camera yields regions from the image. Together, these are used as input to a probabilistic data association filter (PDAF). The PDAF produces a single target amongst potential many candidates and false alarms to determine a target for following. This approach demonstrated better performance than their initial vision based approach though it is noted that the target
can only go unseen for short durations. The system developed here does not use a laser; instead our system relies on the images from the camera only to detect, recognize, and find the approximate distance to the person being followed.

Munoz-Salinas, Aguirre, and Garcia-Silvente developed a person detection and tracking system using stereo vision and a color camera [6]. The stereo camera allows multiple persons to be tracked in a scene and provides useful color and depth information. Their system creates a background model of the environment using a height map. The background model is said to be more robust due to it's ability to handle illumination and intensity changes in a scene. The height map notes the maximum height of the points projected in it. The height map is used to map the background objects in a scene. Persons are considered foreground objects. Detecting a person is to extract the foreground objects from the background using the height map. Detection also uses color information about the persons detected with respect to the background model to help detection. They use an occupancy map to map objects detected in the foreground. Objects detected with similar dimensions to a person are considered candidates for a person. Persons that have already been detected are assigned to current objects in the occupancy map that are being tracked. Persons that have not already been detected are considered new and potential person(s) to follow. The system developed in this work does not use stereo vision or height maps to model the background.

Another vision based system was developed by Brookshire [7]. This system does not require any training of the person to be followed or the environment in which it operates. This is accomplished using a classifier system. The classifier system is based on Histogram of Oriented Gradient features developed by Dalal [8]. HOG descriptors defines the descriptors used for object detection. These descriptors state that local object characteristics in an image can be described by the distribution of intensity gradients or edge directions. The classifier system uses these descriptors and is trained off-line before any following is done to recognize scenes. There are two possible scenes that the classifier system is trained on; those that contain a person(positive detection) and those that do not have a person them(negative detection). When the system is running the classification data is used with the live images to detect whether or not a person is present in the scene. When a person in detected in the scene a tracking algorithm filters the detected person into a track that is used to follow the person.
The track contains information about the direction and speed of the person to be followed. The filter accounts for clutter in the scene and scenes in which a person is not detected. The filter information is provided to a following algorithm. The following algorithm takes a greedy approach to person following. That is, the algorithm will take the shortest most direct path between the current location of the system and the person. This makes it best suited for spacious outdoor environments where obstacles generally do not exist. This approach may not be suitable in indoor environments where obstacles may exist along the shortest path.

This work uses a similar hardware configuration to [9]. A vision based system is used in lieu of laser, GPS, or multi-sensor based systems. This work uses a vision based approach that uses color and shape for detection and following. This work is similar in nature to the first approach used by [9], the first approach done by [5], and other vision based approaches like [6] and [7].

1.2 OVERVIEW OF PREVIOUS INTELLIGENT MACHINES AND ROBOTS LAB WORK IN PERSON FOLLOWING

This thesis builds upon the work of other graduate students that have worked with professor Tarokh on this project. A chronological overview of their work is given for perspective and reasoning that led to this work. The overview also helps to understand the framework that led to the current design and implementation decisions.

1.2.1 The Work of Tarokh and Ferrari

Tarokh and Ferrari developed the initial version of this project [10]. This initial version paved the way for future Intelligent Machines and Robots lab work in robotic person following.

They developed the initial system which included the robotic vehicle, computers, transmitters, receivers, and camera equipment. Their initial hardware configuration used a robot with a camera mounted on it. The robot had a transmitter and receiver mounted to it that would transmit images captured by the camera to a remote laptop. The remote laptop would then process the images and transmit control signals back to the robot. The images were in gray scale in this initial version.
In this version two techniques were explored for person following. The first was image differencing and the second was target tracking. Both approaches share some image processing operations that segment the person from the image which is used in following.

The first operation on the gray scale image is to threshold the image. The images were thresholded with a value of either 140 or 150 which is a somewhat arbitrary middle value of the gray scale range of 0 to 255. Thresholding with these threshold values produced a binary image in which pixels above the threshold value were set to white and pixels below the threshold value were set to black. Thresholding the image segments the objects in the image based on the threshold value used.

Using the thresholded image, the image mass is found by counting the number of pixels that were either black or white in the image as given by:

\[ M = \sum_{x=0}^{R} \sum_{y=0}^{C} g(x,y) \]  

(1.1)

where R and C are the number of rows and columns, respectively and g(x,y) is the image function. The mass of a thresholded image is used to determine the distance from the robot to the person. The mass gives an estimate of distance to the person.

With the distance to the person found using the mass, the location of the person in the x and y directions must be found. This is done using the center of mass as given by:

\[ x_c = \frac{\sum_{x=0}^{R} \sum_{y=0}^{C} y g(x,y)}{M} \]  

(1.2)

\[ y_c = \frac{\sum_{x=0}^{R} \sum_{y=0}^{C} x g(x,y)}{M} \]  

(1.3)

The point \((x_c, y_c)\) is said to be the center of the target and is used to steer the robot towards the person.

Using the center of mass along with a sequence of images over time the following equations give the speed of motion in the X and Y directions:

\[ \frac{dx}{dt} = \frac{X_{cf} - X_{c2}}{T} \]  

(1.4)

\[ \frac{dx}{dt} = \frac{Y_{cf} - Y_{c2}}{T} \]  

(1.5)
Using a positive $dx/dt$ means the object that is moving from left to right and away from the camera. Conversely, a negative $dy/dt$ means that the object is moving from right to left and toward the camera.

Image differencing was the first technique to be used to track a person in motion. Image differencing takes two sequential images in a time series and computes their difference on a pixel by pixel basis. In image differencing, pixels in the output image are set to black if there is no difference between the two images. If there is a difference, the pixel value is set to value of the latest image in the sequence. This results in an image where the difference in the sequence of images shows the motion of the person which is used to track the motion of the person. An issue with this approach was that the camera had to be stationary to avoid introducing unwanted differences due to camera motion or camera shake. This was not desirable for a robot in motion and thus another approach was tried.

Target tracking was the next technique that was used to allow the system to work when the robot is moving. Target tracking attempted to track a property of the person such as color or shape from a sequence of images. The target to be tracked had to be known in advance which was done during an initial training phase using the person's shirt. This approach was used in a bare environment where the shirt would have a high contrast with respect to the environment. To accomplish this typically a black shirt was used which stood out well in test environments. The target tracking approach attempted to segment out the environment with thresholding leaving only the person's shirt which was used for tracking. With the shirt detected via thresholding the mass, center of mass, and direction are computed as noted in equations 1.1 to 1.5.

With the direction and distance to the person found using the above method fuzzy controllers were used to provide speed and steering signals to the robot. Fuzzy sets are used to represent control values and are based on the on the work of [11]. The fuzzy controllers developed handled the control outputs for robot steering and robot speed. The steering controller takes as input $X_c$, the center of mass of the person being followed and $dx/dt$, the speed of the person in the x-direction. The output for the steering controller is the rotation of the robot steering. The speed controller takes as input the mass of the image and $dy/dt$, the speed of the person in the y-direction. The mass of the image is used to denote how close or how far the person is from the vehicle. A larger mass means the person is closer and the
speed is maintained or reduced. A smaller mass and the person is farther away and the speed is maintained or increased. The output for the speed controller is the speed of the robot.

The work of [10] successfully laid the groundwork for person following. John Kuo and Mahmoud Tarokh continued this effort and added some enhancements. They switched from gray scale images to color images, introduced a pan and tilt camera, and introduced new fuzzy controllers for both the camera and robot follower.

1.2.2 The Work of Tarokh and Kuo

Tarokh and Kuo continued the work of Tarokh and Ferrari [10, 12]. They switched from gray scale to color images, added a pan and tilt camera along with other hardware upgrades, added fuzzy controllers for the camera's pan and tilt, and added some improvements to the target tracking algorithm. These changes showed improved results in tests that were conducted in indoor and outdoor environments. They also tested in different types of terrain.

The switch was made from gray scale images to color images in an effort to improve object definition and detection. A color image provides more information than a gray scale image which allows for a more accurate definition of the training colors as well as more accurate detection of the detected colors. The Hue Saturation Brightness (HSB) color model was chosen to use in the image processing algorithms. Some tolerance is allowed when comparing detected colors to training colors. This is to allow for subtle variations in color and lighting as the person moves through different lighting situations.

A pan and tilt camera was also introduced in this version. The pan and tilt camera can move in two directions. It can pan 100 degrees in either horizontal direction and can tilt 90 degrees up and 30 degrees down. The cameras range of motion provides more robust person following since the camera can pan or tilt to locate the person. During following the camera can articulate to keep the person in the center of the frame. If the person is lost, the pan and tilt functionality allows the camera to search to try and relocate the person.

This project introduced new fuzzy controllers to work with the pan and tilt camera and the robot. There is a fuzzy controller for the pan and another for the tilt functions of the camera. The goal of these controllers is to position the camera so that the person is in the center of the image. The other two new controllers are for the robot. These two controllers
control the speed and direction of the robot to try and maintain a constant distance and appropriate direction with respect to the person being followed.

Shape training and identification was also added in place of tracking a static black shirt as was done in [10]. This algorithm requires that the persons shirt be trained for both color and shape to be used when tracking and following the person. During person tracking the detected shape and color is scored based on circularity, compactness, and eccentricity. These measures are compared to the reference shape from the training phase and the region that most closely matches the reference shape is selected for following.

This version was tested in both indoor and outdoor environments on different types of terrain. The indoor tests were conducted in a bare environment on flat surface. The robot successfully navigated through doors, down hallways, and properly identified the target amongst multiple subjects. The outdoor tests were conducted in twilight to reduce the lighting variations that would be present in mid day when passing through bright sun and shadows. The terrain in the outdoor tests included a small hill, steep trails, and rocky paths. The outdoor tests showed good results for these types of terrain. This was attributed to the pan and tilt ability of the camera to keep the person in the frame over terrain and the target tracking algorithm to be able to distinguish the person from other objects in the environment. Tarokh and Kuo noted difficulties in tracking and following with changing illumination of the outdoor environment hence the reason the tests were conducted in the twilight. [12]

1.2.3 The Work of Tarokh, Merlotti, and Duddy

The most recent work on this project was done by [2] which continued the work done by [12]. In this version they made numerous changes and improvements to the previous version. A new approach was implemented to handle changes in illumination that could automatically control parameters of the camera which could affect the brightness of the images captured. The controllers to provide inputs for the pan, tilt, speed, and steering inputs were changed from controllers using fuzzy sets and fuzzy logic to PID controllers. Logic to control the robot and attempt to find the person when the person has not been found for some period of time was added. Lastly, there were some significant hardware changes and upgrades. These changes addressed some of the shortcomings of the previous version and is the hardware and software platform that this work began with.
To address the issue with changing illumination in different environments, experiments were conducted that found an ideal intensity for a range of lighting conditions. Using this ideal intensity, PID controllers were implemented that attempted to maintain the proper intensity using the ideal intensity and the continuous intensity error from each image. The PID controllers would adjust the camera's gain, shutter speed, and iris. The PID controllers to do this performed better than the camera's auto adjustment because of a localized “software sensor” that detects the light intensity of the person in the center of the image as opposed to adjusting based on the entire image like the auto adjustment does. This demonstrated that in an environment that was too light or too dark the intensity could be adjusted so that the image had the ideal intensity. The auto adjustment of the camera tended to under or over correct and produce an undesired intensity.

The control system was also modified to use PID controllers in place of the fuzzy controllers used in [10]. The PID controllers were implemented for the camera pan and tilt along with the speed and steering of the robot. The inputs to these controllers were based on the location and distance to the person based on from detected regions in the image. In this version a region scoring algorithm is used to score each of the detected regions. The region that scores the highest provides input data to the control functions for speed, direction, pan, and tilt. The score of each region is based on measures of the region itself, the detected region with respect to the training region, and the detected region with respect to previous detected regions. A tall and rectangular region that is large in size with respect to the other regions will increase the score of the region. The more a detected region resembles the characteristics of the training region the higher it will score. Lastly, a region's score can be increased if it's distance from the location of the previous winning regions is small. i.e. a similar region has been detected moving in the direction of the person.

The winning region approach used three shape measures to determine a regions score are circularity, compactness, and eccentricity. Each of these use the mass and center of mass of a region to determine a value for the shape measure. The mass of a region is defined as

\[ M = \sum_{j=0}^{p} \sum_{k=0}^{q} f(j,k) \]

where \( f(j,k) = 1 \) if pixel is white, 0 if pixel is black and returns 1 when the pixel is white and 0 otherwise. The center of mass \((X_c, X_y)\) is defined as
\[ X_c = \frac{1}{M} \sum_{j=0}^{p} \sum_{k=0}^{q} (j f(j,k)) \text{ and } Y_c = \frac{1}{M} \sum_{j=0}^{p} \sum_{k=0}^{q} (k f(j,k)) \] where \( p \) is the number of rows and \( q \) is the number of columns in the image.

Circularity is defined as \[ C = \left(1 + \frac{1}{n d_a} \sum_{i=1}^{n} (d_i - d_a)\right)^{-1} \] where \( d_i \) is the distance from a pixel on the region boundary to the center of mass and \( d_a \) is the average distance of the center to the boundary \( d_a = \frac{1}{n} \sum_{i=1}^{n} d_i \). An exact circular region has \( C = 1 \) and a non-circular region has \( C = 0 \).

Compactness is a measure of the squareness of a region. A perfectly square region has a value of 1.0 and a non-square region has a value of 0.0. Compactness is computed as \[ K = \frac{16 \pi M}{\beta^2} \] where \( M \) is the mass as defined in 5.1 and \( \beta \) is the number of pixels along the region's boundary. 

Eccentricity measures the flatness of a region. The flatness of a region is the ratio between the length of the the major and minor axes. Eccentricity is computed as \[ E = 1 - \frac{d_{\text{min}}}{d_{\text{max}}} \] where \( d_{\text{min}} \) and \( d_{\text{max}} \) are the minimum and maximum distances from the center of mass to the region boundary. An eccentric region will have a value of 1.0 where a non-eccentric region will have a value of 0.0.

Using the shape measures circularity, compactness, and eccentricity the similarity of a detected region to the trained reference shape is found by \[ \sigma = \frac{1}{3} \left( e^{-a_k \delta_k} + e^{-a_c \delta_c} + e^{-a_e \delta_e} \right) \] where \( \delta_k, \delta_c, \text{ and } \delta_e \) are the normalized differences between shape measures as given by 

\[ \sigma_k = \frac{K - K_j}{K_{\text{reference}}}, \sigma_c = \frac{C - C_j}{C_{\text{reference}}}, \sigma_e = \frac{E - E_j}{E_{\text{reference}}} \] and \( a_k, a_c, \text{ and } a_e \) are configurable weightings for each measure. The detected region with the highest \( \sigma \) value is the region closest to the trained region/reference region and is used in person following.

Additional modifications were made to the control system to handle the situation where the person was not found for some number of frames. This is one of the behaviors or states that the robot can be in that was added in this version. When this occurs, the robot switches to a search mode to attempt to locate the person. The search mode expands the area that is used to try and detect regions within the image, increases the range that the camera can
pan and tilt to locate the person, and lastly the camera's pan and tilt will search in a sinusoidal pattern to attempt to locate the person. This behavior continues until the person is located and then following continues.

Lastly, some significant hardware changes were made. The switch to a Segway® mobile platform was made. This platform was modified so that additional hardware could be mounted on it. This included a laptop which was capable of controlling all aspects of person following onboard. Speakers were added to announce messages along with a game pad controller to provide control of the robot when person following is not taking place.

The version showed performance improvements over the previous version in bright and dim environments as well as environments where the lighting conditions changed during following. A novel approach to try and locate the person when the person was lost during following also performed well. This version had issues with properly locating and then following the person when there were objects in the scene that had similar colors to the person's shirt that was being followed.

1.3 CONTRIBUTIONS OF THE THESIS

This thesis will build on the work of [1, 2, 10, 12]. This work attempts to improve on the person detection algorithms and calculations used to effect person following. Some of the previous work remains intact including the hardware setup, region growing algorithm, and control algorithms. Previous work used reference shapes and attempted to match how closely the detected regions matched the reference shapes. Another approach selected a single winning region from the detected regions and used it's centroid and effective mass. This work focuses on improving how the detected regions that are found are used for person following by taking into consideration all regions to calculate a location and distance to the person being followed.

This work also inherits and works within some boundaries inherent to a purely vision based approach. The training phase collects hue, saturation, and intensity values from region(s) detected that represent the persons shirt. In prior versions of this work, the system has performed best when the persons shirt is a bright color that contrasts with the background environment [10]. This work will adopt that limitation and focus on the detection algorithms by testing with with shirt colors that contrast the background environment. Tests with
similarly colored objects in the environment to the target color are also performed. The mobile robot also has hardware limitations. The robot has limited weight and power capacity which limits that amount of computing power that the on board computer will have. Our system uses a modest laptop that is underpowered by current hardware standards for computationally intensive tasks like image processing. To work around this limitation the code is optimized where applicable and only a portion of the image from the camera is used to reduce the number of pixels that are processed by the algorithms. Lastly, there is no effort made to learn or adapt from previous images acquired. Using learning algorithms that could adapt detection parameters based on previously acquired image data would be an area worth exploring to try and improve detection in changing environments.

In this implementation each of the detected regions make some contribution to the overall centroid and effective mass. The color match and size for each region is taken into account to determine the centroid of the detected regions. Each region will make some contribution to the overall centroid and effective mass. Regions with the closet color match and largest size will make the largest contribution. Conversely, regions whose color does not match the training colors and has a small size will make the smallest contribution to the centroid of detected regions and effective mass. Regions that are either to small or to large with respect to a typical range of sizes for a segment of a person detected are discarded and make no contribution to the overall centroid or mass. This approach will help eliminate the situation where an incorrect region is used for following. This is explained in more detail in section 2.2.

### 1.4 Description of Remaining Chapters

Chapter 2 will give a fundamental overview of image segmentation. The properties of what constitutes a properly segmented image will be given along with a look at some algorithms that segment images. Chapter 3 will discuss the approach used for person detection along with the algorithms used to implement it. Chapter 4 will give an overview of the software system as a whole including how our approach and algorithms fit into it. Chapter 5 will present the results of testing the system and provide discuss the results in comparison to previous versions of the system developed by [1, 10, 12]. Chapter 6 will summarize the work and offer suggestions for additional improvements and future work.
CHAPTER 2

SYSTEM COMPONENTS

2.1 HARDWARE COMPONENTS

The person following system consists of three main hardware components and other components that may be optionally used. The robotic platform is a mobile device that can drive itself when given the proper signals. A computer is mounted on the robotic platform to process the images received from the camera and generate control signals to drive the robot. Lastly, a camera is mounted on the robotic platform to capture images and transmit them back to the computer. These components provide the significant functionality for this thesis; others such as speakers to emit audio information and hand controls to operate the robot are present but were not used.

2.1.1 Robotic Platform

The platform used for this thesis was a Segway® Robotic Mobility Platform (RMP) as shown in Figure 2.1. The Segway® RMP provides an extensible robotic platform which provides a basic mobile system which can be controlled and allows for extensibility to add custom components to. An on board computer, a camera to obtain images, and other components are also mounted on the system.

Figure 2.1. Segway Robotic Mobile Platform.
2.1.2 Computer
A computer is mounted on the Segway® RMP for the image processing and control systems. A modest computer with a 2.0-GHz processor and 1 gigabyte of Random Access Memory (RAM) is mounted on the Segway®. An on board IEEE 1394 port connects to the camera to the computer. This computer is used for development and testing as well as the actual operation of the Segway® RMP. The computer also controls the pan and tilt for the camera as well as receiving and processing images from it.

2.1.3 Vision System
The camera used to obtain images is a DFK24AF from The Imaging Source and has a resolution of 640 × 480, can process 60 frames per second, and supports color images. It's shutter speed and gain are controllable through software and its iris is manually adjusted. The camera has a C++ library that is used to obtain and process images.

2.2 SOFTWARE COMPONENTS
The software for this system acquires images, processes images, and calculates control signals for the Segway® RMP. The camera and the computer communicate with a library provided by the manufacturer. The source code for this project is shown in the Appendix.

2.2.1 Programming Environment
The C++ language is used for this system including the Graphical User Interface (GUI), image processing, and the control systems. C++ was used in a previous version of this project and was also used in this version for its speed and interoperability with other devices on the system. Manufacturer provided device drivers were in C++ which made interacting with other systems easier. The Microsoft® Visual Studio® integrated development environment was used for development, dependency management, build, and also to debug and run the software.

2.2.2 Detection and Tracking Software Overview
Using the camera, the manufacturer's library to interface with the camera, and the image processing software that was developed for this project, person detection and tracking
is possible by a series of image processing, recognition, and control algorithms. Person
detection and tracking occur by combination of numerous small pieces of software that work
together to provide the desired behavior. An IC Imaging Control® library is used to interface
with the camera; an event notification is sent via callback that receives event notifications
when a new image is available. Using the hardware and software configurations described
here the images are processed by the software to detect the person. Chapter 3 discusses the
person detection algorithms developed for this project to detect the person.
CHAPTER 3

PERSON DETECTION

For the robot to follow the person, the person must first be detected so that their location and distance can be found. To detect the person the image processing system is trained to detect the color characteristics of the person in each image that is processed. Training involves learning the colors that represent the person which is typically their clothing and allowing for some variation due to changes in lighting while person following is taking place. These colors are captured and represented using the hue, saturation, and intensity (HSI) color model. With a color model of the person captured for training the image is then segmented into regions to attempt to locate the person from the rest of the image. Thresholding of the image is then done to segment pixels that are within a range of the trained HSI values. With the thresholded image a region growing algorithm collects and connects pixels to form regions in the image. A region in the image may represent the entire person, other objects in the scene, or a portion of a larger discontinuous region. The regions detected are potential representations of the trained images of the person to follow. Each region is compared with the colors of the training region to find regions that may represent the person. These regions are analyzed to try and determine the location of the person using the centroid of each region and their mass. To determine the centroid and mass for person following the algorithm developed considers the contribution of all the regions detected to determine a single centroid and mass. These values are then used as inputs to PID controllers. The PID controllers control for the camera's pan and tilt along with the robots steering and speed.

3.1 HUE SATURATION INTENSITY (HSI) COLOR MODEL

In the training and detection phases the HSI color model is used. The HSI color model is used in favor of the the Red, Green, Blue (RGB) color model for image processing. This is because the HSI color model allows for the hue and saturation of a color to be constant while varying the intensity. In the HSI color model hue and saturation represent the
dominant color perceived and intensity represents the amount of light or intensity in the hue and saturation. The HSI color model is often represented as a double cone as shown in Figure 3.1.

An HSI color is a point within this cone. A color within this cone is described by the individual hue, saturation, and intensity components. The hue component is represented as an angle between 0 and 360 degrees. An angle of 0 degrees represents red, 120 degrees is green, and 240 degrees is blue. The saturation represents how much white exists in the color and is represented as a value between 0.0 and 1.0. Lastly, the intensity or lightness represents how much light can pass through. The range for this value is from 0.0 to 1.0 where 0.0 represents black and 1.0 is white.

The HSI color model is well suited for this project. During the training phase the colors within the area being trained may be under various lighting conditions such as different light sources, sources of shade, and folds within the clothing. Thus, it is desirable to capture the hue and saturation independently since the intensity will change during person following as different lighting situations are encountered. The HSI model works well with
these lighting variations by allowing the intensity value to be independently changes from the hue and saturation.

3.2 TRAINING

In the training phase, the operator selects a rectangular region from the person to be trained with a graphical user interface (GUI) that shows video from the camera. The selection should cover the entire shirt and the general torso area to capture a representation of the person. The colors in the selected region are converted to the HSI model and are stored for reference as the training representation of the person. The intensity of the trained values in the HSI model are allowed some variation during following to detect the person when lighting variations occur during person following. The standard deviations for each hue and saturation pair \((h_j, s_j)\) are calculated for the training set. Tolerances \(\varepsilon_h\) and \(\varepsilon_s\) are added to each training pair \((h_j, s_j)\) which are proportional to the standard deviation. This is to allow for variance in hue and saturation during following. Allowing this variance enables similar shades of color to be detected, which is how different light influences are handled. Each point in the hue-saturation plane is replaced by an ellipse given by the equation:

\[
\frac{(h - h_j)^2}{\varepsilon_h^2} + \frac{(s - s_j)^2}{\varepsilon_s^2} = 1 \quad ; \quad j = 1, 2, \ldots, n
\]  

(3.1)

To capture the entire shirt and additional colors within the bounds of the ellipses in 3.1, region growing is performed from the selected rectangular region. The grown region may contain areas that did not fit within the color model of the ellipse given above. This could be due to various lighting factors, obstructions in the scene, or irregularities in the contour of the person's clothing. It is desirable to capture as much of the shirt as possible. To do this with the grown region an 8-directional chain code is used to find a singular region and encapsulate any holes in the detected region(s) that may exist. The region yielded by the 8-directional chain code is considered to be the training region and is referenced during person following as a color model of the person. This concludes the training phase.

During person following statistics from the training area are used as reference to evaluate the detected regions. Normalized histograms for hue and saturation are computed for the training area selected so that each detected region found during following can be compared to the training hue and saturation. The normalization of these histograms allows
the regions to be compared to the training region independently of the number of pixels in both the training selection and the detected region. This is referred to as the color match which is discussed in more detail in section 3.4.

**3.3 Thresholding**

Thresholding is the first phase in person detection algorithm. Thresholding will classify pixels based on if they match the training colors or not using the ellipses in 3.1 as a threshold value. Thresholding will yield a set of pixels that fall within our thresholding criteria. Using the ellipses allows for the threshold image produced to account for some variance.

```java
boolean isInEllipse(HSIData, hueStandardDeviation, saturationStandardDeviation, hueMean, saturationMean)
{
    boolean retVal = false;
    double value = (hsi.hue - hueMean)² / (hueStandardDeviation)²
    * (hsi.saturation - saturationMean)²
    / (saturationStandardDeviation)²
    if(value > 1.0)
    {retVal = true
    }
    return retVal
}
```

The thresholding algorithm uses the function `isInEllipse` to check if a pixel should be thresholded. This function takes as input a pixel, the HSI data of an image, and the mean and standard deviation of the hue and saturation. The function will return a boolean value to denote whether or nor the pixel met the threshold criteria.

This concludes the first phase in detection which we segment the colors we are interested in using thresholding. In the next phase, we find the regions of the thresholded image using a region growing algorithm.

**3.4 Region Growing**

Using the thresholded image, region growing is performed on the thresholded image to find regions. Region growing is an image segmentation technique that begins with an initial seed point to grow a region. Regions are grown based on the neighboring pixels of the initial seed pixel. Pixels that are similar to the initial seed point are added to the region until a
complete region is formed of similarly colored pixels. Region growing detects the regions in the image whose pixels are within the ellipses from equation 3.1. The algorithm for region growing is:

```cpp
ImageRegionsInfo& RegionGrower::growRegions(CookedImage& cookedImage, int left, int top, int right, int bottom) {
    m_imageRegionsInfo.reset();
    for (int y = top; y < bottom; y++) {
        Region* currentRegion = NULL;
        for (int x = left; x < right; x++) {
            PixelRegionInfo& pixelInfo = cookedImage.getPixel(x, y);
            // Skip black pixels. And skip yellow, which is the traversal circle.
            if (!pixelInfo.isMarked()) {
                currentRegion = NULL;
                continue;
            }
            if (y > top) {
                int startX = std::max(x - 1, left);
                int endX = std::min(right, x + 2);
                for (int i = startX; i < endX; i++) {
                    PixelRegionInfo& peer = cookedImage.getPixel(i, y - 1);
                    if (!peer.isMarked()) {
                        continue;
                    }
                    Region* hisRegion = peer.getOwner();
                    if (!currentRegion) {
                        currentRegion = hisRegion->getTopRegion();
                    } else {
                        Region* hisTop = hisRegion->getTopRegion();
                        currentRegion = currentRegion->getTopRegion();
                        if (currentRegion != hisTop) {
                            // join him to the current region
                            currentRegion->supercede(hisTop);
                        }
                    }
                }
            }
            if (currentRegion) {
                pixelInfo.setOwner(currentRegion);
                currentRegion->addPoint(Point(x, y));
            } else {
                currentRegion = m_imageRegionsInfo.addNewRegion(cookedImage, Point(x, y), pixelInfo);
            }
        }
    }
    return m_imageRegionsInfo;
}
```
This algorithm will yield a set of regions from the thresholded image. Each of the regions found must be analyzed to determine if it represents a region that is similar to the representation of the person. A region that is found could represent a significant part of the person in the scene or it could be an object in the background. If the region is significant, we want to give it more weight when used for detection and following. If the region is insignificant, we want to give it less weight. The weight of a region is based on it's color match, size, and location with respect to the training model of the person. Regions that are extremely small or extremely large with respect to the size of the training region are discarded since they are considered outliers. The color match of a region denotes how closely the detected region matches the colors of the training region. The larger a region is the more weight it will have. Lastly, regions that are in a reasonable location with respect to the last location the person was detected will have a higher weight.

3.5 Detection

Using the regions found from the region growing algorithm the location of the person must be found from the regions. The regions found will represent a part of the person and each region is used to determine the location and distance to the person. The location is found using a centroid of detected regions approach that considers the contribution of all the regions found based on characteristics of the region. The distance to the person is found using an effective mass of detected algorithms approach. The centroid of detected regions and effective mass algorithms use the size of the region, the color match of the region relative to the training region, and other characteristics of the region to determine a centroid and a distance. Additionally, the search area is limited to reduce the area that is searched to try and detect the person. This optimization reduces the number of pixels in each image that are looked at to detect regions and find the centroid of detected regions and effective mass.

3.5.1 Centroid of Detected Regions

Determining the centroid of detected regions is an important part of this algorithm and differs from how the location of the person was calculated in previous versions of this project. In previous versions a single region was chosen from the detected regions as the winner. This was based on characteristics of the regions such as shape, size, and color with respect to the training region. The closer the detected region matches the training region for
each characteristic the higher the score it will receive. This approach works well when the
chosen region represented a significant portion of the person’s shirt or there was only one
region detected. However, this is not always the case. The detected regions can potentially
consist of many regions of various sizes. If the wrong region is chosen as the detected region,
then this can lead to errors with following.

This approach attempts to address this by considering all regions when calculating the
centroid. The area of each region and its centroid are used to calculate the centroid of the
detected regions. If there are \( n \) detected regions, then each region will have an area \( A_i \) and a
centroid \((x_i, y_i)\) for \( I = 1, 2, ..., n \). The normalized area is defined as

\[
\alpha = \frac{A_i}{A_{\text{max}}}
\]

where \( A_{\text{max}} = \max(A_1, A_2, ..., A_m) \) and \( 0 \leq \alpha \leq 1 \). The area of a detected region is used to
determine its contribution to the centroid of the detected regions which is defined as

\[
x_c = \frac{\sum_{i=1}^{n} \alpha_i x_i}{\sum_{i=1}^{n} \alpha_i}, \quad y_c = \frac{\sum_{i=1}^{n} \alpha_i y_i}{\sum_{i=1}^{n} \alpha_i}
\]

The color of each detected region is also taken into consideration with respect to the colors
found in the training set to formulate a color match for the centroid of detected regions. To
do this, a normalized histogram for the trained regions hue \( F_t(h) \) and saturation \( G_t(h) \) is
found. The normalized histogram is used so that the color counts in the histogram are
independent of the number of pixels in both the trained area and the detected region. The
normalized histogram for the hue and saturation for each detected region \( i \) found as \( F_i(h) \) and
saturation \( G_i(h) \). The difference between the training region and the detected region's
histogram for hue is

\[
\Delta F_i = \sum_{h=\text{min}(h)}^{\text{max}(h)} |F_i(h) - F_t(h)|
\]

where \( \text{min}(h) \) and \( \text{max}(h) \) are the minimum and maximum hue in \( F_i(h) \) or \( F_t(h) \). The
difference between the trained region's histogram and the detected regions histograms are
formed by

\[
\Delta G_i = \sum_{s=\text{min}(s)}^{\text{max}(s)} |G_i(s) - G_t(s)|
\]
where $\min(h)$ and $\max(h)$ are the minimum and maximum saturation in $G_t(h)$ or $G_i(h)$.

The histogram equations are combined to formulate the relative color match between the trained and detected regions

$$\beta = \frac{\Delta F_i + w_0 \Delta G_i}{1 + w_0}$$

(3.6)

where $w_0$ is a weighting factor that can be used to emphasize or deemphasize the effect of hue relative to saturation on the color match. The exponential functions $e^{-\cdot}$ yield a value of $\beta_i$ between 0 and 1. When there is no color match $\beta_i$ will become smaller and when the color match is high $\beta_i$ will equal 1. Thus, $0 \leq \beta \leq 1$. To take into account the color match into the estimation of the centroid the centroids equations are modified to

$$x_c = \frac{\sum_{i=1}^{m} (\alpha_i + w\beta_i) x_i}{\sum_{i=1}^{m} (\alpha_i + w\beta_i)}$$

$$y_c = \frac{\sum_{i=1}^{m} (\alpha_i + w\beta_i) y_i}{\sum_{i=1}^{m} (\alpha_i + w\beta_i)}$$

(3.7)

Equation 3.7 is referred to as the area weighted centroid.

### 3.5.2 Effective Mass of Detected Regions

The centroid of detected regions gives a good approximation of the location of the person being followed. The distance between the person and robot must also be found so that the robot can adjust it's speed to keep a proper distance between itself and the person. This is found using the effective mass. The mass is used as an indication of distance from the person detected to the robot following the person. The greater the mass the closer the robot is to the person and the smaller the mass the farther away the robot is from the person. The robot uses this value to adjust it's speed to maintain a reasonable distance to the person. The effective mass of the detected regions is computed as:

$$M_{\text{eff}} = \frac{\sum_{i=1}^{m} (\alpha_i + w\beta_i) A_i}{\sum_{i=1}^{m} (\alpha_i + w\beta_i)}$$

(3.8)

Where, $A_i$ is the area of the region, $w$ is the weighting factor, $\alpha_i$ is the normalized area of the region, and $\beta_i$ is the color match of the region.
### 3.5.3 Limiting the Search Area

To optimize detection, the entire frame is not processed to try and detect the location of the person within it. Instead, the search area is limited to the following: (1) Near the center of the image, (2) Near the last known location of the person, (3) The area between 1 and 2, and (4) Biased by the direction of motion. Section 4.2.1 will describe how the camera attempts to keep the person in the center of the frame which makes restriction 1 reasonable. Restriction 2 is due to that fact that the person does not move significantly between frames due to the short interval between them. Restriction 3 accounts for the cases when restrictions 1 and 2 may be significantly different. The area between them is likely to include the person. Lastly, since the direction the person is moving is known we can restrict the search area to the last known direction the person was moving.

Suppose that \((0,0)\) is the coordinates of the origin in the current frame, used in item (1) above. Let \((x_{c1}, y_{c1})\) be the last known location (person’s centroid) with respect to the current coordinate frame, as in item (2). Also let \((x_{c2}, y_{c2})\) be the estimated location in the current coordinate frame based on the direction of motion, mentioned in item (4). Then (1)-(4) require checking hue and saturation of the pixels within the rectangle formed by the lines:

\[
\begin{align*}
  x &= \min(x_{c1}, x_{c2}) - a, \\
  y &= \max(y_{c1}, y_{c2}) - b
\end{align*}
\]

\[
\begin{align*}
  x &= \max(x_{c1}, x_{c2}) + a, \\
  y &= \min(y_{c1}, y_{c2}) + b
\end{align*}
\]

where \(a\) and \(b\) are some constants. The above rectangle is valid only if the points \((x_{c1}, y_{c1})\) and \((x_{c2}, y_{c2})\) are not too far away from the current coordinate origin \((0,0)\), i.e.

\[
\sqrt{x_{c1}^2 + y_{c1}^2} \leq d_{1\text{max}} \quad \text{and} \quad \sqrt{x_{c2}^2 + y_{c2}^2} \leq d_{2\text{max}}
\]

(3.10)

where \(d_{1\text{max}}\) is the maximum allowable distance between last know location and the current location of the centroid, and \(d_{2\text{max}}\) is the corresponding maximum distance based on direction of motion. Typically these maximum distances are one fourth of the diagonal length of the image. The too far away points could indicate noisy data and thus are invalid. In case inequalities (3) are not satisfied, the estimated values will be based on several previous values, using a simple extrapolation. Rectangles that are determined to be to far away from the origin are not used. Instead, the values from previous valid rectangle values are used in it's place.
The person detection algorithm uses the HSI color model for training and detection. The HSI color model allows for lighting variations to be considered by allowing for variation in the intensity. This model works well since the color of a person's clothing moving through different lighting will change from the camera's perspective. The initial training phase uses the person's clothing to gather training colors. The colors obtained are referenced in the remaining phases of person detection to identify the person from other objects in the image. For each image obtained during following, the algorithm attempts to segment the person from the image to determine their location. To segment the training colors from the image thresholding is first performed on the image to eliminate colors that do not represent the person in the image. Regions are then grown from the thresholded image with a region growing algorithm. The regions found may represent a portion of the person or in some cases the entire person. Each region is then considered when determining the location and distance to the person by allowing all regions to make some contribution to the overall centroid and effective mass based on their characteristics. Considering all regions eliminates the possibility that a significant region is ignored or an insignificant one is chosen in finding the location and distance of the person. The size, centroid, and color match of each region found contributes to a single centroid that is used to determine the location of the person. The distance to the person is found using the effective mass of the detected regions which measures the size of the regions found. A large mass and the person is closer while a smaller mass means that the person is farther away. The centroid of detected regions is used as a reference for the location of the person. The effective mass is used as a reference to the distance to the person from the robot. These values are then used as input to the control algorithms which control the direction and speed of the robot for person following. This chapter has covered how the person is detected in the image. With the person detected the robot must follow the path of the person. Chapter 4 discusses how the camera and robot work to follow the person once they are detected.
CHAPTER 4

ROBOT AND CAMERA CONTROL

Using the centroid of detected regions and effective mass, the robot must now follow the person. The robot attempts to maintain a reasonable distance to the person while also keeping the person centered in the camera. Using the centroid of detected regions, the cameras pan and tilt controls operate to keep the person in the center of the frame in the x and y directions. If the camera's pan controls are not sufficient to keep the person centered in the x-direction, then the robot's steering will be manipulated to account for larger directional changes. The effective mass is used as input to adjust the robot's velocity to maintain an appropriate distance from the person. The outputs pan, tilt, steering, and velocity are controlled by proportional integral derivative (PID) controllers.

4.1 PID CONTROLLER

The output values for camera pan, camera tilt, robot speed, and robot steering are controlled using a PID controller. The general continuous form of a PID controller is

\[ u = k_p e + k_i \int_0^t e \, dt + k_d \frac{de}{dt} \]  

(4.1)

where \( e \) is the error and \( u \) is the controller output. In practice the continuous form is inefficient and not used for a discretely timed implementation such as a function in a program. So a discrete implementation is used to increase performance and simplicity. The general formula for the discrete implementation is

\[ u(j) = u(j-1) + k_0 e(j) + k_1 e(j-1) + k_2 e(j-2) \]  

(4.2)

where \( k_0 = \left( k_p + \frac{1}{2} k_i \frac{T}{T} + \frac{k_d}{T} \right) \), \( k_1 = \left( k_p + \frac{1}{2} k_i \frac{T}{T} - \frac{2k_d}{T} \right) \), and \( k_2 = \frac{k_d}{T} \). This discrete form allows for simple implementation in code using only standard operators. In code, the discrete PID controller is implemented using the following code:
double calculatePID(double error) {
    integralError = integralError + 0.5 * (error + previousError);
    derivativeError = (error - previousError);
    previousError = error;
    return (error * kp) + (integralError * ki) + (derivativeError * kd);
}

4.2 Camera Control

The camera is mounted on a platform with two servo controllers attached to the camera. The pan and tilt for the camera is controlled by each of the servos. The pan and tilt controls attempt to keep the person in the center of the camera's viewfinder using the location of the person as found in equation 3.7. The camera's shutter speed and gain are also adjusted during following to maintain desirable lighting conditions for image processing.

4.2.1 Camera Pan and Tilt Control

The camera is controlled by a discrete PID controller as noted in (4.2). The pan and tilt controls each have their own controller which take as input the centroid of detected regions \((X_c, Y_c)\). The pan controller will provide output such that the person will be centered in the x direction based on the input \(X_c\). The tilt controller will provide output such that the person will be centered in the y direction based on the input \(Y_c\). The outputs from either controller are proportional to the amount in which \(X_c\) or \(Y_c\) is away from the center of the image (0,0).

The PID controller for the panning of the camera implements this algorithm with the values \(error = 0, K_p = 0.3, K_i = 0.0, Kd = 0.2\). These values were obtained empirically and provided good performance during testing. A sample plot of this PID controller is shown in Figure 4.1.
With this PID controller, the output signal lags the reference signal. The output signal does not overshoot the reference signal and eventually catches up to the reference signal. For the pan functionality of the camera, this is a desirable behavior. Since the camera may process frames and try to detect the person in the scene before it catches up to the person by panning it is acceptable for the pan to lag since it will eventually catch up. A partial detection may occur before the camera has panned completely which is OK. To overshoot the reference signal would be problematic since the camera may lose the person and have difficulty finding them after that.

The PID controller for the tilting of the camera implements the PID algorithm with the values $error = 0, K_p = 0.1, Ki = 0.0, Kd = 0.0$. A sample plot of this PID controller is shown in Figure 4.2.
The PID controller for the tilt controller lags the reference signal even more than the PID controller for panning. This is due to the small value of $K_p$ and $K_i$ and $K_d$ being equal to zero. The output signal does not overshoot the reference signal and takes a long time to catch up to it. This behavior is acceptable for the tilt control since we do not expect large or oscillating movements in the $y$ direction during following. It is expected that most movements of the person in the $y$ direction will be small and not enough for the person to leave the camera's field of view. If the robot were operating in terrain with elevation changes, then the values for this controller could be adjusted accordingly.

4.2.2 Camera Shutter Speed and Gain Control

To achieve proper image intensity controllers are implemented to control the shutter speed and gain of the camera. These controllers consider the gain $g$, shutter speed $s$, and iris radius $r$ of each image to achieve the proper intensity and an image that is in focus.

The desired intensity of the person's shirt $I_d$ is used with the current intensity $I$ to formulate the error for each image as $I_e = I_d - I$. The current intensity $I$ is measured from the detected person's shirt in each image during following. An initial value for the camera's iris is chosen based on whether the robot is being operated indoors or outdoors. In general, outdoors there would be more light so the $I_d$ would be smaller and indoors there is less light so $I_d$ would be larger to allow more light in. Whether indoors or outdoors each lighting situation is unique and the initial values would be chosen accordingly. The proportional derivative (PD) controller is a function of $I_e$ and $I$ to achieve the desired intensity $I$. The controller operates on the camera's iris and shutter speed to maintain images with the desired intensity that are also in focus.

4.3 Robot Control

The robot itself is also controlled by a discrete PID controller as noted in (4.2). The robot control uses the $X$ centroid $X_c$ and the effective $M_{eff}$. The robot's steering is affected with respect to $X_c$ to try and keep the person centered. This is similar to the way the pan controls operate, however the steering of the robot can account for greater directional change that can the pan control. This is necessary if the person makes sharp turns around a corner for example. The effective mass $M_{eff}$ is used as input for the controller that controls the robot's velocity. A large $M_{eff}$ will cause the controller to maintain its current speed or slow down and
a small $M_{off}$ will cause the robot to speed up. Both of these velocity adjustments are to keep the person at an optimum distance from the robot.

The speed PID controller as shown in Figure 4.3 uses only the proportional term with a value of 1.0. The affect on the output signal is that it will be the same as the error value. This is why we see no undershoot or overshoot on the output signal for speed. This is reasonable because we want the speed of the robot to be the same as the person it is following. If there is significant overshoot in the PID controller, then the robot may speed up

Figure 4.3. Speed PID controller.

to fast and become to close to the person. If there is significant undershoot in the PID controller, then the robot may move to slowly and fall to far behind the person.

The steering of the robot does not use a PID controller like the pan, tilt, and speed functions do. The steering is controlled by an algorithm that uses output values from the pan controller. The values from the pan controller that are used are the current pan position, the pan center, and the difference $\Delta$.

The algorithm to produce a steering value uses an average of the last $n$ $\Delta$ values. The output steering value will be within the bounds of an lower and upper limit to prevent it from turning to far in either direction.

```java
    double pan = panController.value
    double servoCenter = panController.servoCenter
    double delta = pan - servoCenter

    list panValues = panHistory.getValues
    panValues.add(pan)
    double sum = panValues.sum
```
averagePan = sum / maxIntegration

double turnScaled = 2.0 / (panUpperLimit - panLowerLimit) * averagePan

Using the person detection scheme described in Chapter 3 and the person following scheme as described in Chapter 4 a series of tests were conducted to analyze the performance of person detection and person following. The test scenarios and their results are discussed in Chapter 5.
CHAPTER 5

RESULTS

Using the person detection approach described in Chapter 3 and the person following approach described in Chapter 4 tests were conducted to analyze the performance of the system in different conditions. The results of these tests are compared to previous versions of this project. The approach presented here attempts to improve on previous approaches by considering all detected regions to form a single centroid of detected regions and effective mass used for person following. All regions will make some contribution to the centroid which represents the location of the person and mass which represents the distance to the person. This approach attempts to address some of the issues with the winning region approach. To compare the different approaches tests from this approach are shown and the performance is discussed in comparison to the previous approach.

5.1 WINNING REGION APPROACH TRIALS

For comparison, past tests from the winning region approach are revisited. In these tests the system was tested in an indoor environment. Sample frames are chosen from a test where multiple regions were detected and a single winning region was chosen per the algorithm. Though the algorithm performed well in many cases these examples show some instances where the winning region is not necessarily the best representation. In the images below the blue lines encompass a detected region and the white “x” denotes the centroid of the winning region.

In Figure 5.1-5.4 the winning region and it's centroid are generally not near the center of the person being followed. This results in an undesirable centroid choice which can later lead to problems in following. For following, the centroid of the winning region is used. If the centroid is to far off center of the person in the x direction, this can cause the robot to turn on a path that is away from the person. Similarly for the y value of the centroid; if the y value of the centroid represents a point that is to high or to low with from the center of the person, then the camera may tilt to far in the y direction. Either of these situations may result in the
Figure 5.1. Multiple regions detected example 1 of 4.

Figure 5.2. Multiple regions detected example 2 of 4.

Figure 5.3. Multiple regions detected example 3 of 4.
person being lost in following due to a centroid that was not in a good location. In sample frames such as these, the centroid of detected regions approach can choose a better centroid since all regions will be considered when calculating the overall centroid.

5.2 CENTROID OF DETECTED REGIONS TRIALS

The centroid of detected regions approach as described in section 3.4 was developed and tested in an indoor environment with different configurations. Tests were conducted indoors in a laboratory environment with overhead lighting and some natural light from the nearby windows. A sample of three tests are presented here for analysis using this test environment. The first is a basic test with a single person. The goal of this test is to verify that a reasonable centroid is found when multiple regions are detected. The second tests attempts to do the same with simulated lighting variations. Lighting of the scene is altered mid test using the lights and shutters in the room to simulate lighting variations. The third test inserts objects into the scene that are similarly colored to the person's clothing. This test ensures that multiple disparate regions are detected and forces the algorithm to find an overall centroid amongst them.

In these tests, each image shown will have various shapes drawn onto it. The regions detected are outlined in varying colors, the centroid of each region is shown as a green circle within the region, the centroid of detected regions is shown as a white x, and the search area is denoted by a white rectangle.
5.2.1 Test One – Basic Test

Stationary tests were performed initially to test the implementation of the algorithms. This test focuses on the image processing aspects of the system including thresholding, region detection, and the centroid of detected regions algorithm as described in section 3.4. The goal of this test is to verify that when multiple regions are detected each region will make some contribution to the overall centroid. This can be done by inspection of the centroids of each region along with the centroid of detected regions. The overall centroid will typically be near the largest region with the closest color match. The other regions will pull the centroid of detected regions in their direction by some amount also depending on their size and color match relative to the training region.

In the following tables, some sample images are shown from a sequence of stationary tests that were conducted. These tests show images where multiple regions were detected with different attributes. The attributes of each region are given to the right of the image in the table.

In Figure 5.5, there are three detected regions in the image. The largest region encompasses most of the shirt. There are also two extraneous smaller regions detected in the background. The largest region has a size of 5533 and a centroid of \((X_c, Y_c) = (148, 129)\). The two smaller regions have a size of 91 with a centroid of \((X_c, Y_c) = (196, 221)\) and size 167 with a centroid of \((X_c, Y_c) = (218, 216)\). The largest region's alpha as computed in equation 3.4 has an alpha of 1.0. The other two regions have an alpha of 0.016447 and 0.030183. Considering, the color match of each region with respect to the training region the color matched weighted centroid is \((X_c, Y_c) = (150, 132)\). This is a shift of 2 to 3 pixels in the x and y directions with two falsely detected regions. This shift is acceptable and results in a reasonable centroid to be used for guiding the direction of the robot. Had the two extraneous regions been larger in size or a comparable size to the largest region detected a significant shift in the color matched region weighted centroid would have occurred.

In Figure 5.6, there are now four detected regions in the image. The largest region detected is most of the shirt, the two small extraneous regions are again detected, and the fourth region is detected in the upper left part of the shirt. This region is separated by my hand that I moved through the image to attempt to create multiple regions by breaking up the continuous color of the shirt. The largest region in this image has a size of 4613 with a
Algorithm: Regions detected?: 3
Algorithm: Largest Region: 5533 points.
Algorithm: Normalized centroid: 148 129
Algorithm: Region centroid: 148 129
Algorithm: Region area: 5533
Algorithm: Region centroid: 196 221
Algorithm: Region area: 91
Algorithm: Region centroid: 218 216
Algorithm: Region area: 167
Algorithm: Hue Histogram Delta: 10812.000000
Algorithm: Saturation Histogram Delta: 13150.000000
Algorithm: colorMatch: 0.000000
Algorithm: Alpha: 1.000000
Algorithm: Hue Histogram Delta: 11069.000000
Algorithm: Saturation Histogram Delta: 11069.000000
Algorithm: colorMatch: 0.000000
Algorithm: Alpha: 0.016447
Algorithm: Hue Histogram Delta: 11069.000000
Algorithm: Saturation Histogram Delta: 11035.000000
Algorithm: colorMatch: 0.000000
Algorithm: Alpha: 0.030183
Algorithm: Color match weighed centroid: 150 132
Algorithm: Effective mass: 1930.333333

Figure 5.5. Stationary test 1.

Algorithm: Regions detected?: 4
Algorithm: Largest Region: 4613 points.
Algorithm: Normalized centroid: 149 135
Algorithm: Region centroid: 117 101
Algorithm: Region area: 247
Algorithm: Region centroid: 149 135
Algorithm: Region area: 4613
Algorithm: Region centroid: 196 221
Algorithm: Region area: 105
Algorithm: Region centroid: 218 216
Algorithm: Region area: 169
Algorithm: Hue Histogram Delta: 11069.000000
Algorithm: Saturation Histogram Delta: 11069.000000
Algorithm: colorMatch: 0.000000
Algorithm: Alpha: 0.053544
Algorithm: Hue Histogram Delta: 11079.000000
Algorithm: Saturation Histogram Delta: 10672.000000
Algorithm: colorMatch: 0.000000
Algorithm: Alpha: 1.000000
Algorithm: Hue Histogram Delta: 11069.000000
Algorithm: Saturation Histogram Delta: 11069.000000
Algorithm: colorMatch: 0.000000
Algorithm: Alpha: 0.036636
Algorithm: Color match weighed centroid: 150 137
Algorithm: Effective mass: 1283.500000

Figure 5.6. Stationary test 2.
centroid of 149, 135. The three smaller regions attributes are size 247 with a centroid of \((X_c, Y_c) = (117, 101)\), size 169 with centroid of \((X_c, Y_c) = (218, 216)\), and size 105 with a centroid of \((X_c, Y_c) = (196, 221)\). Considering the size and color match of the detected regions the color match region weighted centroid is shifted to \((X_c, Y_c) = (150, 137)\). In Figure 5.5, the largest region dominates the color matched region weighted centroid due to it's size despite the existence of two falsely detected regions.

In Figure 5.7, there are a total of four detected regions. There are two mid sized regions split by my arm moving through the image. The topmost region has a size of 2875 and it's centroid is at \((X_c, Y_c) = (151, 104)\) and the lower region has a size of 1911 and it's centroid is at \((X_c, Y_c) = (144, 163)\). The two smaller extraneous regions are detected in the lower right part of the image. Their attributes are size 105 and centroid at \((X_c, Y_c) = (196, 221)\) and size 173 with centroid \((X_c, Y_c) = (218, 216)\). The color match weighted centroid of the four regions has split the y difference between the two larger regions while considering the small regions on the lower right of the image to a negligible amount. The color match region weighted centroid of this image is \((X_c, Y_c) = (151, 132)\).
In Figure 5.8, there are two detected regions and a single smaller region. The largest region has a size of 4883 with a centroid of \((X_c, Y_c) = (149, 122)\) and the smaller region has a size of 156 with a centroid of \((X_c, Y_c) = (123, 173)\). The two falsely detected regions that were present in previous images are not present in this image. This is possibly due to a change in light or the control of the camera's parameters that did not result in the colors of them meeting the criteria to be considered. The color match region weighted centroid of this image is \((X_c, Y_c) = (148, 123)\) which is a small shift of 1 pixel in the x and y direction of the smaller region that was detected.

In Figure 5.9, there are three detected regions. The largest region encompasses the entire shirt and has a size of 5928 with a centroid of \((X_c, Y_c) = (146, 128)\). The size of this region is larger than previous ones since the shirt has not been obstructed in any way. The two extraneous regions on the lower right part of the image are still present. Their attributes are 105 with a centroid of \((X_c, Y_c) = (196, 221)\) and 182 with a centroid of \((X_c, Y_c) = (218, 216)\). Similar to the previous images there is a small shift towards these regions to account for their presence and the color match weighted centroid is \((X_c, Y_c) = (148, 123)\).

This basic stationary test shows when a single region is detected the centroid of detected regions is the same as the centroid of the single region. There are no other regions to consider and the single regions has a weight of 1.0 so the centroid calculations at that point are the same. This is the same behavior that exists with the winning region approach. In this
scenario our approach does not differ. However, when there are multiple regions detected the centroid of detected regions takes into account all the regions detected into determining an overall centroid. The centroid of detected regions in Figures 5.3 and 5.4 exhibit this by calculating a centroid that is a better representation then if another regions was used to represent the person.

5.2.2 Test Two - Lighting Variation Test

Lighting variation was simulated and tests similar to those in test one were conducted to investigate performance during lighting variations. Midway through a test the lighting was dimmed by turning off half of the lights in the room. This test exercised the ability to detect the person in a different lighting situation than the lighting used during training. Figures 5.10 to Figure 5.14 show images from the same test scenario under different lighting conditions.

Figure 5.10 shows the initial lighting and region detection after training was completed. The red shirt was used as the training region and was successfully detected.

Figure 5.11 shows another detection in a brighter scene with the subject standing under the light source. In this Figure the top of the shirt is brighter than the rest of the shirt and was not considered part of the region. Most of the remaining parts of the shirt were detected successfully in this brighter light.
In Figure 5.12 the lighting was dimmed by turning off some of the lights in the room. A good detection of the person's shirt has occurred in the dimmed lighting.

In Figure 5.13 a partial detection has occurred in a dimmed lighting situation. The part of the shirt that is facing towards the light matches the training colors plus tolerances.
well and is detected as a matched color. The centroid of this detected region is also in a reasonable location considering the partial detection.

In Figure 5.14 a very small region is detected in the dimmed lighting conditions. Most of the shirt is facing away from the lights that are on. In this case a very small region is detected at the bottom of the shirt. The remaining parts of the shirt are too dark with respect to the training region and are not detected.

This series of images show region detections and person detections of varying qualities under different lighting conditions. The detection algorithms allows for some variation in brightness as given by the \textit{isInEllipse} algorithm in section 3. The \textit{isInEllipse} algorithm included a deviation value from the mean colors found during training to account for lighting variations which allows person detection to occur in lighting variations like the ones shown.
5.2.3 Test Three – Similarly Colored Objects Test

The third test included objects that were similar in color to the training colors. In this case the training colors were the color of the shirt which was a shade of red. Other objects in the scene were similarly colored orange and red objects. The images in Figure 5.15 to 5.18 show a portion of the test where I am moving from left to right. In each of these images the shirt is detected along with one or two additional objects that are similarly colored. In each of the cases shown the shirt dominates the overall centroid due to it's size which makes it contribute more to the overall centroid. However, though the contributions of the other regions are undesired in this case their contribution is present in the overall centroid. In Figure 5.16, the overall centroid has shifted to the left to include the other object detected. The same thing has happened in Figures 5.17 and 5.18 where the overall centroid has shifted to include the other objects detected. In Figures 5.16, 5.17, and 5.18 the overall centroid has shifted to include other objects detected while maintaining a reasonable overall centroid that closely approximates what the centroid of the person would be if it were the only detection. In this situation some accuracy is compromised but if the regions detected were different parts of the person's shirt that were under different lighting conditions then this approach would approximate the centroid well.

Figure 5.15. Multiple regions detected 2 of 4.
Figure 5.16. Multiple regions detected 1 of 4.

Figure 5.17. Multiple regions detected 3 of 4.

Figure 5.18. Multiple regions detected 4 of 4.
5.2.4 Incorrect Centroid Detection

Despite considering all regions found in the scene in some situations an incorrect overall centroid may be found and used for following. One situation this may occur is when multiple regions are detected and the person is moving. The regions are found within the limited area that is being searched as described in section 4.4.3. When there is at least one region in the search area and the person is moving outside the search area the overall centroid may be pulled away from the person and towards the other regions. When the person completely moves out of the search area the region of the person is no longer detected and thus makes no contribution to the overall region. The remaining regions in the scene are the only contributors to the overall region calculation none of which are the person since they are outside of the limited search area. This situation may result in an incorrect centroid detection. The following sequence of images in Figures 5.19, 5.20, and 5.21 show an example of this.

![Image](image1)

There are three regions detected and the largest region is the shirt. The subject is moving to the right. The overall centroid is drawn up and to the left towards the two smaller detected regions.

Figure 5.19. Incorrect centroid detection 1 of 3.

This test showed a scenario where an incorrect centroid detection occurred in a limited number of frames. It is possible that during person following incorrect centroid detections will occur. A small number of incorrect centroids during person following will generally not lead to errors in person following because at some point a correct detection will occur and the robot will use that location for following.
Again, there are three regions detected and the largest region is the shirt of the subject though it is smaller than in previous frames due to limiting the search area. The subject has moved further to the right. The overall centroid has shifted further than before up and to the left due to the smaller size of the largest region.

Figure 5.20. Incorrect centroid detection 2 of 3.

There is only one region detected since both of the previous regions are outside of the search area. An incorrect centroid detection has occurred.

Figure 5.21. Incorrect centroid detection 3 of 3.

5.4 SUMMARY

This work investigated an approach to person following to consider all the regions detected during image segmentation to determine the location of a person and follow the person. This approach aimed to incrementally improve on approaches by [2, 10, 12] where a single region was chosen from the set of detected regions to be used for following.
Considering the contributions of all the detected regions where the size and color match of the region affects how much the region contributes to the centroid of detected regions aimed to find a reasonably good location and distance to the person while eliminating or reducing some cases with the winning region approach. To test if this approach was an improvement the algorithms were developed and implemented using the existing software and hardware setup to investigate performance. Many tests were conducted and a sample of them were presented here. The tests scenarios were: (1) a single person with static lighting and minimal movement by the person, (2) a single person with lighting variations and some movement, and (3) a single person with other objects in the scene and some movement to induce multiple regions found by segmentation. In the first scenario a small number of regions are detected and typically only a single region is found. With a small number of regions to consider there are less contributions to the overall centroid and typically the single region found or the largest region of the detected regions will dominate the overall centroid. In this scenario, the winning region approach and the centroid of detected regions approach will perform about the same. In the second scenario this approach was able to deal with minor lighting variations and the movement by the person created smaller regions to be found. When there are multiple regions detected and some of them represent the person this approach performs well since it will consider the regions that represent the person. The overall centroid may not be as accurate with the winning region approach is used and the correct region is chosen. This is an acceptable compromise since the overall centroid found would be comparable with the added benefit of eliminating a class of errors where the incorrect region could be chosen for following. Lastly, in the third scenario consider the case when there are multiple regions detected not all of which are the person such as in Figure 5.19. The winning region approach has a 1 in \(n\) chance of selecting the proper region to use for following where \(n\) is the number of regions detected. Selecting an incorrect region in this case will likely result in the robot going in the wrong direction. With the centroid of detected regions approach the overall centroid will not be as good as if the correct centroid were chosen by the winning region approach but more importantly it will be better than when the wrong region and it's centroid are chosen as the winner. In this situation considering all regions may compromise finding the precise location of the person but it will also eliminate using an incorrect region for person following. Thus, for these three common scenarios there
are compromises made which may have a minor affect on accuracy in some cases but in all cases enables person following to continue by avoiding choosing an incorrect location for following.

The results presented here conclude our work on this system. Chapter 6 discusses the achievements made with this work and looks ahead to suggest future direction for this project.
CHAPTER 6

ACHIEVEMENTS AND FUTURE WORK

This thesis implemented and tested changes to an existing system with the idea that the system could be improved by changing some of the core algorithms. The winning region approach to determine the location and distance of a person being followed was replaced in favor of this approach which considered the contribution of all regions for determining the location and distance to the person. This approach has demonstrated improvements to behavior of the system in most cases. In some situations the accuracy of the location and distance to the person might be not be quite as accurate by considering all detected regions but this is an acceptable trade off for eliminating the possibility of an incorrect region being used for following. Though improvements have been made there remains many areas in which future work can be done to continue improvement.

Though this system demonstrates improvements over previous systems and thus some degree of success, there are aspects of the system that can be improved and provide direction in future projects. Tarokh and Kuo suggested that the existing system be moved from a Java/C++ hybrid to a system entirely in C++ and thus native machine code for efficiency reasons [12]. This has been done since the writing of their paper and all work done on this project was done using C++. While this conversion has been beneficial in terms of efficiency, there have been significant trade offs made in terms of maintainability, expressiveness, and convenience. While these factors are somewhat vague and harder to quantify compared with a metric such as the speed of native code versus virtual machine code they are no less significant. Thus, a consideration for future work on this project is to move to a managed runtime environment and a more modern language. A managed runtime environment and a language with garbage collection will free the researcher from having to manually manage memory and allow them to focus on research. These two changes could alleviate a lot of the issues that are inherent with the existing setup and provide numerous benefits. At this time two good choices for a runtime environment would be the Microsoft CLR or the Java Hotspot™. The libraries used for the camera and system controllers are
available in CLR/.NET versions. These libraries are not available on the JVM. The C# language can run on .NET and is a reasonable first choice to convert from C++. Microsoft's F# language is also worth investigation. F# is “succinct, expressive and efficient functional and object-oriented language”. It is believed that the features of the CLR/.NET and either the C# or F# languages will translate into better code for person following and allow the time and focus of someone working on the project to be spent on implementing a system that performs better.

Currently the way to test the software is on the actual hardware with the robot running. The test results are inspected via print statements, output images, and by verifying that the robot moved along the correct path. Testing could be improved by adding a framework to automate and quantify the test process. The current testing process makes the testing the correctness of any code change now matter how small a time consuming process. This is just to test for correctness. To test for performance improvements is more difficult since it has to be done by inspection. There is no way to test how various parameters perform against one another or against a baseline. Thus, it is suggested that a simulation capability or test framework be implemented so that detection and tracking algorithms can be tested independently and comparatively against different versions. It would be ideal to give the same sequence of test images to two different configurations or implementations, run them, and infer results quantitatively. This feature would allow the system to be tested continuously and optimized over time to determine a good configuration and iteratively test algorithm changes.

Currently, the person detection algorithms perform best when the color of the shirt has some contrast with objects seen during detection in the foreground and background during following. This leaves us with a significant dependency on both the training phase being done correctly and the color of the persons shirt with respect to the color of other objects in the environment during following. It is thought that the shape detection algorithms can be improved to recognize changing shapes from frame to frame even when they do not appear to be the same shape as the person. Brookshire developed a system where there is no color or training dependency on the person being followed [7]. While, this approach differs from the way our system approach we could aim to obtain similar performance since ours benefits from the use of training information.
Our system currently relies on the effective mass of detected regions to denote the
distance from the robot to the person as given in equation 3.10. This approach is simple and
has proven effective thus far. It would be worthwhile to investigate and test a stereo vision
setup to find the distance from the person to the robot. Stereo vision uses multiple cameras to
acquire images from different angles and then calculates the depth from the combination of
the images from the different angles from each camera. The mass of detected regions and the
stereo vision approach could be compared in various scenarios with a known distance from
the robot to the person to see which is more accurate.

The training phase for this system is done once and at the start of person following.
This brief period captures and uses as reference a certain shape, size, and color of the person
for following. This brief period will not capture all of the lighting characteristics of the
person to use which would make detection and following not perform well. For example, if
the lighting during training is significantly different than the lighting during following then
the trained color model may not be a good representation of the colors found during
following. Similarly, the size of the person during training may be significantly different than
the size of the person detected due to the distance from the camera. This would make it
difficult to distinguish between a small person in the distance and a large object in the
foreground that have the same color characteristics but different sizes. Continuously learning
from characteristics acquired during following and updating training models to include this
information would improve person detection in varying conditions.
REFERENCES


APPENDIX

C++ SOURCE CODE
#pragma once

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"

class CRobotUIApp : public CWinApp {
public:
   CRobotUIApp();

   // Overrides
   virtual BOOL InitInstance();

   // Implementation
   DECLARE_MESSAGE_MAP();
};

extern CRobotUIApp theApp;

/* ---------------------------------------------------------------------------
   * Thesis: Vision Based Person Following Using an Improved
   * Image Segmentation Approach
   *
   * Spring 2012
   *
   * RobotUI.cpp: The graphical user interface for the application.
   *
   * Author: Brian Blaine, Padu Merlotti
   ** ------------------------------------------------------------------------*/
#include "stdafx.h"
#include "RobotUI.h"
#include "RobotUIDlg.h"
#define _CRTDBG_MAP_ALLOC

BEGIN_MESSAGE_MAP(CRobotUIApp, CWinApp)
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

CRobotUIApp::CRobotUIApp() {
}

CRobotUIApp theApp;

BOOL CRobotUIApp::InitInstance() {

    // If you have a trial version, the license key is "0" without quotation marks
    // Example: if(!DShowLib::InitLibrary( 0 ))
    //
    // If you have a licensed version, the license key is a string in quotation marks.
    // The license key has to be identical to the one entered during the
    // IC Imaging Control setup. Example: if( !DShowLib::InitLibrary( "XXXXXXX" ) )

    if (!DShowLib::InitLibrary("ISB3200016679")) {
        AfxMessageBox("The IC Imaging Control Class Library could not be initialized.\n(invalid license key?)");
        exit(1);
    }  

    // At the end of the program, the IC Imaging Control Class Library must be cleaned up
    // by a call to ExitLibrary().
    atexit(DShowLib::ExitLibrary);

    // InitCommonControls() is required on Windows XP, if an application
    // manifest specifies use of ComCtl32.dll version 6 or later to enable
    // visual styles. Otherwise, any window creation will fail.
    InitCommonControls();

    CWinApp::InitInstance();

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need
// Change the registry key under which our settings are stored
// TODO: You should modify this string to be something appropriate
// such as the name of your company or organization
SetRegistryKey(_T("RobotUI"));

CRobotUIDlg dlg;

m_pMainWnd = &dlg;

dlg.DoModal();

// Since the dialog has been closed, return FALSE so that we exit the
// application, rather than start the application's message pump.
return FALSE;

/* ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* +
* + Spring 2012
* +
* RobotUIDLG.h: Definition for GUI events and event handlers.
* +
* Author: Brian Blaine, Padu Merlotti
** ---------------------------------------------------------------------------*/
#pragma once
#include "afxwin.h"
#include "Listener.h"
#include "afxcmn.h"
#include "Scoring.h"

#ifndef PHIDGET_H
#include "Phidget.h"
#endif
#include "phidget21.h"
#define PHIDGET_H
#endif

#include "PersonSearch.h"

#ifndef GAMEPAD_H
#include "behGamepadControl.h"
#define GAMEPAD_H
#endif

using namespace ImageLib;

class CRobotUIDlg : public CDialog, public IScoreHandler, public IParameterProvider, public IGamepadHandler {

public:
  CRobotUIDlg(CWnd * pParent = NULL);
  ~CRobotUIDlg();

  bool selectDevice();

  afx_msg void OnBnClickedButtondevice();
  afx_msg void OnBnClickedButtonimagesettings();
  afx_msg void OnBnClickedButtonlivevideo();
  afx_msg void OnBnClickedReset();
  afx_msg LRESULT OnSetExpGain(WPARAM wp, LPARAM lp);

  void SetButtonStates(void);

  void ComputeSettings();

  void HandleTargetInfo(TargetInfo& scores);
  void HandleTargetLost();
  void HandleColorStats(int hueMean, int hueVar, int satMean, int satVar, int intMean, int intVar);
  void HandleLuminanceStats(bool gainAuto, bool expAuto, int gain, int exp);

  void OnColorCheck();
  void OnTraining();
  void OnStopped();
  void OnConfig();
  void ToggleLogging();
  bool GetLoggingState();

  bool GetControlLuminance();
  int GetTargetLuminance();
ImageLib::ThresholdParams GetParams() {
    return m_params;
}

DWORD GetTargetForecastTimeMS();
int GetIntensityAlpha();
bool SkewHue();
const char * SaveImagesToPath();
void GetLumControllerInfo(double & p, double & i, double & d);

protected:
    virtual void DoDataExchange(CDataExchange * pDX); // DDX/DDV support

void InitControl();
void OnTimer(UINT_PTR nIDEvent);

bool m_beatStatus;
double m_currentPan; // tracks the position of the person

protected:
    HICON m_hIcon;

    // Generated message map functions
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnClose();
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnStnClickedStaticvideo();
    afx_msg void OnColorSettingChange();
    afx_msg void OnBnClickedMisc();
DECLARE_MESSAGE_MAP()

private:
    DShowLib::Grabber m_cGrabber;

    DShowLib::FrameHandlerSink::tFHSPtr m_pSink;
    CListener m_cListener;

    void NormalizePoint(CPoint & pt);

    bool CheckServoPanEnabled();
    bool CheckServoTiltEnabled();
CBUTTON m_cButtonSettings;
CBUTTON m_cButtonLive;
CSTATIC m_cStaticVideoWindow;
CBUTTON m_cbxControlPan;
CBUTTON m_cbxControlTilt;

bool m_initComplete;
CPHIDGETSERVOHANDLE _servo;

BOOL m_controlLuminance;

ImageLib::ThresholdParams m_params;
CBUTTON m_cbxAutoShutter;
BOOL m_autoShutter;
CSliderCtrl m_shutterSlider;
CBUTTON m_cbxAutoGain;
BOOL m_autoGain;
CSliderCtrl m_gainSlider;
int m_targetIntensity;

CPersonSearch m_personSearch;
CRITICAL_SECTION m_cs;
double m_mass;
double m_x;
double m_y;
bool m_trackingValid;

public:
    CSpinButtonCtrl m_tgtIntSpin;

public:
    CListBox m_lbxTraceCats;

public:
    _afx_msg void OnLbnSelchangeTracingCats();

    void UpdatePIDControllers();
    void UpdateTrackingInfo();
    BOOL m_doRobotControl;
    double m_panP;
    double m_panI;
    double m_panD;
    double m_tiltP;
    double m_tiltI;
    double m_tiltD;
    double m_luminanceP;
    double m_luminanceI;
double m_luminanceD;
DWORD m_recaptureMS;
int m_intensityAlpha;
CEdit m_txtExp;
CEdit m_txtGain;
BOOL m_skewHue;
afx_msg void OnBnClickedChangeColor();
CString m_saveImagesPath;
BOOL m_saveImages;
afx_msg void OnBnClickedSaveImages();
};
/* ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* Spring 2012
* RobotUIDLG.cpp: Implementation for events and event handlers.
* Author: Brian Blaine
** ---------------------------------------------------------------------------*/

#include "stdafx.h"
#include "RobotUI.h"
#include "RobotUIDlg.h"
#include "\RobotUIDlg.h"
#include "Tracing.h"
#include "PerfCounter.h"
#include "Speech.h"
#include "ColorName.h"
#include "SegwayRmp.h"
#include "AbstractArbiter.h"
#include "SegwayArbiter.h"
#include "PanTiltArbiter.h"
#include "BehaviorList.h"
#include "behControlSpeed.h"
#include "behControlSteering.h"
#include "behControlPan.h"
#include "behControlTilt.h"
#include "behSearchPerson.h"
ifndef GAMEPAD_H
include "behGamepadControl.h"
define GAMEPAD_H
endif
#ifndef

CSegwayRmp * rmpLink;

CAbstractArbiter * segwayArbiter;
CPanTiltArbiter * panTiltArbiter;

CBehaviorList * behaviorList = CBehaviorList::Instance();

CbehControlTilt * behControlTilt = 0;
CbehControlPan * behControlPan = 0;
CbehControlSteering * behControlSteering = 0;
CbehControlSpeed * behControlSpeed = 0;
CbehGamepadControl * behGamepadControl = 0;
CbehSearchPerson * behSearchPerson = 0;

// this is a simple helper that automates setting color preferences

struct ColorSettings {
    const char * speakText;
    int satTolerance;
    int targetIntensity;
    ColorSettings(const char * text, int sat, int bright)
        : speakText(text), satTolerance(sat), targetIntensity(bright) {}
};

static ColorSettings s_settings[] = {
    ColorSettings("bravo", 15, 50),
    ColorSettings("romeo", 8, 80),
    ColorSettings("yankee", 15, 80)
};

static int s_next_setting = 0;
const int NUM_SETTINGS = 3;

const int TIMER_ID_HEARTBEAT = 400;
const int HEARTBEAT_UPDATE_RATE = 25;

using namespace DShowLib;

#define SET_EXPGAIN (WM_USER + 2)
// wparam = EXP_AUTO << 8 | gain_auto
// lparam = EXP_VAL << 16 | GAIN_VAL
#define SERVO_PAN    0
#define SERVO_TILT  1

#define TILT_MIN  125
#define TILT_MAX  169
#define PAN_MIN   55
#define PAN_MAX   180

#define AUTO_TRAINING_SIZE 20  //size of the automatic training area

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog {
public:
    CAboutDlg();

    // Dialog Data
    enum { IDD = IDD_ABOUTBOX };  

protected:
    virtual void DoDataExchange(CDataExchange * pDX);  // DDX/DDV support

protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD) {
}

void CAboutDlg::DoDataExchange(CDataExchange * pDX) {
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

CRobotUIDlg::CRobotUIDlg(CWnd * pParent /*=NULL*/) : CDIalog(CRobotUIDlg::IDD, pParent)
, m_initComplete(false)
, m_panP(0.06)
, m_panI(0.0)
, m_panD(0.2)
, m_tiltP(0.06)
, m_tiltI(0.0)
, m_tiltD(0.45)
, m_currentPan(0.0)
, m_controlLuminance(TRUE)
, m_autoShutter(FALSE)
, m_autoGain(FALSE)
, m_targetIntensity(40)
, m_doRobotControl(TRUE)
, m_mass(0.0)
, m_x(0.0)
, m_y(0.0)
, m_trackingValid(false)
, m_luminanceP(0.8)
, m_luminanceI(0)
, m_luminanceD(0.1)
, m_recaptureMS(1500)
, m_intensityAlpha(1)
, m_skewHue(FALSE)
, m_saveImagesPath(_T("e:\logs"))
, m_saveImages(FALSE)
{
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
  InitializeCriticalSection(&m_cs);
}

CRobotUIDlg::~CRobotUIDlg() {
  DeleteCriticalSection(&m_cs);
}

void CRobotUIDlg::DoDataExchange(CDataExchange * pDX) {
  CDialog::DoDataExchange(pDX);
  DDX_Control(pDX, IDC_BUTTONIMAGESETTINGS, m_cButtonSettings);
  DDX_Control(pDX, IDC_BUTTONLIVEVIDEO, m_cButtonLive);
  DDX_Control(pDX, IDC_STATICVIDEO, m_cStaticVideoWindow);
  DDX_Control(pDX, IDC_CONTROL_PAN, m_cbxControlPan);
  DDX_Control(pDX, IDC_CONTROL_TILT, m_cbxControlTilt);
  DDX_Check(pDX, IDC_CONTROL_LUMINANCE, m_controlLuminance);
  DDX_Check(pDX, IDC_THRESH_HUE, m_params.useHue);
  DDX_Check(pDX, IDC_THRESH_SAT, m_params.useSat);
  DDX_Check(pDX, IDC_THRESH_INT, m_params.useIntensity);
  DDX_Text(pDX, IDC_THRESH_HUE_TOL, m_params.hueTolerance);
  DDX_Text(pDX, IDC_THRESH_SAT_TOL, m_params.satTolerance);
  DDX_Text(pDX, IDC_THRESH_INT_TOL, m_params.intTolerance);
  DDX_Text(pDX, IDC_PAN_P, m_panP);
  DDX_Text(pDX, IDC_PAN_I, m_panI);
DDX_Text(pDX, IDC_PAN_D, m_panD);
DDX_Text(pDX, IDC_TILT_P, m_tiltP);
DDX_Text(pDX, IDC_TILT_I, m_tiltI);
DDX_Text(pDX, IDC_TILT_D, m_tiltD);
DDX_Control(pDX, IDC_TILT_D, m_tiltD);
DDX_Control(pDX, IDC_AUTO_SHUTTER, m_cbxAutoShutter);
DDX_Control(pDX, IDC_AUTO_GAIN, m_cbxAutoGain);
DDX_Control(pDX, IDC_SLIDER_SHUTTER, m_shutterSlider);
DDX_Control(pDX, IDC_SLIDER_GAIN, m_gainSlider);
DDX_Text(pDX, IDC_TARGET_INT, m_targetIntensity);
DDV_MinMaxInt(pDX, m_targetIntensity, 0, 255);
DDX_Control(pDX, IDC_SKEW_HUE, m_skewHue);
DDX_Control(pDX, IDC_SAVE_IMAGES_PATH, m_saveImagesPath);
DDX_Check(pDX, IDC_SAVE_IMAGES, m_saveImages);
}

// CRobotUIDlg message handlers
BEGIN_MESSAGE_MAP(CRobotUIDlg, CDialog)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
//}}AFX_MSG_MAP
ON_BN_CLICKED(IDC_RESET, OnBnClickedReset)
ON_BN_CLICKED(IDC_BUTTONDEVICE, OnBnClickedButtondevice)
ON_BN_CLICKED(IDC_BUTTONIMAGESETTINGS, OnBnClickedButtonimagesettings)
ON_BN_CLICKED(IDC_BUTTONLIVEVIDEO, OnBnClickedButtonlivevideo)
ON_WM_CLOSE()
ON_STN_CLICKED(IDC_STATICVIDEO, &CRobotUIDlg::OnStnClickedStaticvideo)
ON_WM_MOUSEMOVE()
ON_WM_LBUTTONDOWN()
ON_MESSAGE(SET_EXPGAIN, OnSetExpGain)
ON_BN_CLICKED(IDC_THRESH_HUE, &CRobotUIDlg::OnBnClickedMisc)
ON_BN_CLICKED(IDC_THRESH_SAT, &CRobotUIDlg::OnBnClickedMisc)
ON_BN_CLICKED(IDC_THRESH_INT, &CRobotUIDlg::OnBnClickedMisc)
BOOL CRobotUIDlg::OnInitDialog() {
    CDialog::OnInitDialog();

    // INIT the servo controllers
    //CPhidgetServo_create(&_servo);
    //CPhidget_open((CPhidgetHandle)_servo, -1);

    m_tgtIntSpin.SetRange(0, 255);

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu * pSysMenu = GetSystemMenu(FALSE);

    if (pSysMenu != NULL) {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty()) {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);    // Set big icon
SetIcon(m_hIcon, FALSE);   // Set small icon

// Resize the video window to IMAGE_WIDTH*IMAGE_HEIGHT pixels.
m_cStaticVideoWindow.SetWindowPos(NULL, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, SWP_NOMOVE | SWP_NOZORDER);

// Add the CListener object to the CGrabber object.
m_cGrabber.addListener(&m_cListener);

m_cListener.SetParent(this);

m_cListener.setScoreProcessor(this);

m_cListener.setParamProvider(this);

// Pass the video window to the listener, so it can draw in it.
m_cListener.setViewCWnd(&m_cStaticVideoWindow);

// Set the sink
m_pSink = FrameHandlerSink::create(DShowLib::eRGB24, 3);

m_pSink->setSnapMode(false);    // Automatically copy every frame to the sink and call CListener::frameReady().

m_cGrabber.setSinkType(m_pSink);

// Try to load the previously used video capture device

if(m_cGrabber.openDev( "DFx 21AF04" ))
if (m_cGrabber.loadDeviceStateFromFile("device.xml")) {
    // Display the device's name in the caption bar of the application.
    SetWindowText("RobotUI " + CString(m_cGrabber.getDev().c_str()));
    m_cGrabber.startLive(false);    // The live video will be displayed by the CListener object.
} else {
    if (!selectDevice()) {
        AfxMessageBox("No device was selected.");
        return FALSE;
    }
}
// Initialize the VCDProp object to access the properties of our ICImagingControl object
CSimplePropertyAccess& props(m_cListener.getProperties());

props.init(m_cGrabber.getAvailableVCDProperties());

// now, get the gain & shutter properties
int minGain, maxGain, minExp, maxExp;

m_cListener.GetGainExpProperties(minGain, maxGain, minExp, maxExp,
    m_autoGain, m_autoShutter);

m_gainSlider.SetRangeMin(minGain);

m_gainSlider.SetRangeMax(maxGain);

m_shutterSlider.SetRangeMin(minExp);

m_shutterSlider.SetRangeMax(maxExp);

m_initComplete = true;

SetButtonStates();

int i = 0;

while (TraceLib::TraceNames[i]) {
    m_lbxTraceCats.AddString(TraceLib::TraceNames[i]);
    ++i;
}

m_cbxControlPan.SetCheck(BST_CHECKED);

m_cbxControlTilt.SetCheck(BST_CHECKED);

InitControl();

return TRUE;  // return TRUE unless you set the focus to a control.
}

void CRobotUIDlg::OnSysCommand(UINT nID, LPARAM lParam) {
    if ((nID & 0xFFF0) == IDM_ABOUTBOX) {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    } else {
        CDialog::OnSysCommand(nID, lParam);
    }
}
/* If you add a minimize button to your dialog, you will need the code below to draw the icon. For MFC applications using the document/view model, this is automatically done for you by the framework. */

void CRobotUIDlg::OnPaint()
{
if (IsIconic())
{
    CPaintDC dc(this);  // device context for painting.

    // Center icon in client rectangle.
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
}
else {
    CDialog::OnPaint();
}
}
m_cGrabber.showDevicePage(this->m_hWnd);

// If we have selected a valid device, save it to the file "device.xml", so
// the application can load it automatically when it is started the next time.
if (m_cGrabber.isDevValid()) {
  m_cGrabber.saveDeviceStateToFile("device.xml");
}

// Now display the device's name in the caption bar of the application.
SetWindowText("RobotUI " + CString(m_cGrabber.getDev().c_str()));
SetButtonStates();
}

// Show the image settings dialog of IC Imaging Control.
void CRobotUIDlg::OnBnClickedButtonImagesettings() {
  if (m_cGrabber.isDevValid()) {
    m_cGrabber.showVCDPropertyPage(this->m_hWnd);
    m_cGrabber.saveDeviceStateToFile("device.xml");
  }
}

// This method sets the states of the dialog's buttons.
void CRobotUIDlg::SetButtonStates(void) {
  bool bDevValid = m_cGrabber.isDevValid();
  bool bIsLive = m_cGrabber.isLive();

  m_cButtonSettings.EnableWindow(bDevValid);
  m_cButtonLive.EnableWindow(bDevValid);

  if (!bDevValid) {
    m_cButtonLive.SetWindowText("Live Start");
  }

  if (bIsLive) {
    m_cButtonLive.SetWindowText("Live Stop");
  } else {
    m_cButtonLive.SetWindowText("Live Start");
  }
}
// This method starts and stops the live video.
void CRobotUIDlg::OnBnClickedButtonlivevideo() {
    if (m_cGrabber.isDevValid()) {
        if (m_cGrabber.isLive()) {
            m_cGrabber.stopLive();
        } else {
            // Call startLive with "false", because the live display is done
            // by the CListener object.
            m_cGrabber.startLive(false);
        }
    }
    SetButtonStates();
}

void CRobotUIDlg::OnClose() {
    m_cGrabber.stopLive();

    Sleep(1000);
    CDialog::OnClose();
}

void CRobotUIDlg::NormalizePoint(CPoint& pt) {
    if (pt.x < 0) {
        pt.x = 0;
    } else if (pt.x >= IMAGE_WIDTH) {
        pt.x = IMAGE_WIDTH - 1;
    }
    if (pt.y < 0) {
        pt.y = 0;
    } else if (pt.y >= IMAGE_HEIGHT) {
        pt.y = IMAGE_HEIGHT - 1;
    }
}

void CRobotUIDlg::OnStnClickedStaticvideo() {
    if (GetCursorPos(&m_cListener.m_selectionStart)) {
        SetCapture();
        m_cListener.SetSelectionMode(true);
        m_cStaticVideoWindow.ScreenToClient(&m_cListener.m_selectionStart);
        NormalizePoint(m_cListener.m_selectionStart);
        m_cListener.m_lastMove = m_cListener.m_selectionStart;
    }
}
void CRobotUIDlg::OnMouseMove(UINT nFlags, CPoint point) {
    if (m_cListener.GetSelectionMode()) {
        // these are relative to my client position - turn them to the static control's
        // coordinates, which are the same as the image's
        ClientToScreen(&point);
        m_cStaticVideoWindow.ScreenToClient(&point);
        m_cListener.m_lastMove = point;
        NormalizePoint(m_cListener.m_lastMove);
    }
}

void CRobotUIDlg::OnLButtonUp(UINT nFlags, CPoint point) {
    if (!m_cListener.GetSelectionMode()) {
        return;
    }

    ReleaseCapture();

    m_cListener.SetSelectionMode(false);
    ClientToScreen(&point);
    m_cStaticVideoWindow.ScreenToClient(&point);
    m_cListener.m_lastMove = point;
    NormalizePoint(m_cListener.m_lastMove);

    m_cListener.TrainColorsToSelection();
}

void CRobotUIDlg::HandleTargetLost() {
    EnterCriticalSection(&m_cs);

    if (m_trackingValid) {
        DO_TRACE(TraceLib::TrackingState, "Target lost!");
    }

    m_trackingValid = false;

    LeaveCriticalSection(&m_cs);
}

void CRobotUIDlg::HandleTargetInfo(TargetInfo& scores) {
    EnterCriticalSection(&m_cs);

    // cache values
m_trackingValid = true;
m_mass = scores.mass;
m_x = scores.x;
m_y = scores.y;

LeaveCriticalSection(&m_cs);
DO_TRACE(TraceLib::TrackingState, "tracking:%d,x=%f,y=%f,m=%f", m_trackingValid, m_x, m_y, m_mass);
}

void CRobotUIDlg::HandleColorStats(int hueMean, int hueVar,
                                    int satMean, int satVar,
                                    int intMean, int intVar) {
    char buf[128];
sprintf(buf, "%d, %d", hueMean, hueVar);
SetDlgItemText(IDC_HUE_STATS, buf);
sprintf(buf, "%d, %d", satMean, satVar);
SetDlgItemText(IDC_SAT_STATS, buf);
sprintf(buf, "%d, %d", intMean, intVar);
SetDlgItemText(IDC_INT_STATS, buf);
}

void CRobotUIDlg::HandleLuminanceStats(bool gainAuto, bool expAuto,
                                        int gain, int exp) {
    WPARAM wp = 0;
    LPARAM lp = 0;

    if (gainAuto) {
        wp |= TRUE;
    }

    if (expAuto) {
        wp = wp | (TRUE << 8);
    }

    // lparam = EXP_AUTO << 8 | gain_auto
    // lparam = EXP_VAL << 16 | GAIN_VAL
    lp = (unsigned int)gain;

    lp = lp | (unsigned int)exp << 16;

    PostMessage(SETP_EXPGAIN, wp, lp);
}

// wparam = EXP_AUTO << 8 | gain_auto
// lparam = EXP_VAL << 16 | GAIL_VAL
LRESULT CRobotUIDlg::OnSetExpGain(WPARAM wp, LPARAM lp) {
    m_autoGain = (wp >> 8) ? TRUE : FALSE;
    m_autoShutter = (wp & 0x00FF) ? TRUE : FALSE;
    int shutter = (int)(lp >> 16);
    int gain = (int)(lp & 0x0000FFFF);

    m_shutterSlider.SetPos(shutter);
    m_gainSlider.SetPos(lp & 0x0000FFFF);
    CString tmp;

    if (!m_autoShutter) {
        tmp.Format("%d", shutter);
    }

    m_txtExp.SetWindowText(tmp);

    if (!m_autoGain) {
        tmp.Format("%d", gain);
    } else {
        tmp.Empty();
    }

    m_txtGain.SetWindowText(tmp);

    // UpdateData(FALSE);

    return 0;
}

bool CRobotUIDlg::CheckServoPanEnabled() {
    if (m_cbxControlPan.GetCheck() == BST_CHECKED) {
        return true;
    }

    return false;
}

bool CRobotUIDlg::CheckServoTiltEnabled() {
    if (m_cbxControlTilt.GetCheck() == BST_CHECKED) {
        return true;
    }

    return false;
}
void CRobotUIDlg::OnBnClickedReset() {
    UpdateData();
    m_cListener.ResetTraining();
    m_trackingValid = false;
    UpdateTrackingInfo();
    behaviorList->Reset();
}

bool CRobotUIDlg::GetControlLuminance() {
    return m_controlLuminance ? true : false;
}

DWORD CRobotUIDlg::GetTargetForecastTimeMS() {
    return m_recaptureMS;
}

int CRobotUIDlg::GetIntensityAlpha() {
    return m_intensityAlpha;
}

bool CRobotUIDlg::SkewHue() {
    return m_skewHue ? true : false;
}

const char * CRobotUIDlg::SaveImagesToPath() {
    if (!m_saveImages || m_saveImagesPath.IsEmpty()) {
        return NULL;
    }

    return m_saveImagesPath;
}

int CRobotUIDlg::GetTargetLuminance() {
    return m_targetIntensity;
}

void CRobotUIDlg::GetLumControllerInfo(double & p, double & i, double & d) {
    p = m_luminanceP;
    i = m_luminanceI;
    d = m_luminanceD;
}
void CRobotUIDlg::OnBnClickedMisc() {
    if (m_initComplete) {
        UpdateData();
        UpdatePIDControllers();
    }
}

bool CRobotUIDlg::selectDevice() {
    m_cGrabber.showDevicePage(m_hWnd);
    return m_cGrabber.isDevValid();
}

void CRobotUIDlg::OnLbnSelchangeTracingCats() {
    INT items[TraceLib::TraceCategoryLAST];
    int retval = m_lbxTraceCats.GetSelItems(TraceLib::TraceCategoryLAST, items);

    if (LB_ERR != retval) {
        for (int i = 0; i < TraceLib::TraceCategoryLAST; i++)
            SET_TRACE_ENABLED((TraceLib::TraceCategory)i, false);

        for (int i = 0; i < retval; i++)
            SET_TRACE_ENABLED((TraceLib::TraceCategory)items[i], true);
    }
}

#define TraceLib::TraceCategoryLAST

/// Control code

void CRobotUIDlg::InitControl() {
    //allocate the machine link
    rmpLink = new CSegwayRmp();
    rmpLink->Open();

    segwayArbiter = new CSegwayArbiter(rmpLink, behControlPan);
    panTiltArbiter = new CPanTiltArbiter();

    //add all behaviors to the behavior list
    behControlPan = new CbehControlPan(50, 0, m_personSearch, panTiltArbiter, 30, PAN_MAX, PAN_MIN);
    behControlTilt = new CbehControlTilt(40, 0, m_personSearch, panTiltArbiter, 22, TILT_MAX, TILT_MIN);
behControlSpeed = new CbehControlSpeed(15, m_personSearch, (CSegwayArbiter*)segwayArbiter);
behControlSteering = new CbehControlSteering(15, (CbehControlPan*)behControlPan);
behGamepadControl = new CbehGamepadControl(30);
behSearchPerson = new CbehSearchPerson(30,
m_personSearch,
panTiltArbiter,
(CbehControlPan*)behControlPan,
(CbehControlTilt*)behControlTilt,
behControlSpeed,
behControlSteering);

behGamepadControl->SetListener(this);

behaviorList->AddBehavior(behGamepadControl);
behaviorList->AddBehavior(behControlPan);
behaviorList->AddBehavior(behControlTilt);
behaviorList->AddBehavior(behControlSpeed);
behaviorList->AddBehavior(behControlSteering);
behaviorList->AddBehavior(behSearchPerson);

UpdatePIDControllers();

behaviorList->Reset();

//set heartbeat timer
SetTimer(TIMER_ID_HEARTBEAT, HEARTBEAT_UPDATE_RATE, 0);
}

void CRobotUIDlg::UpdatePIDControllers() {
    // read values into PID controllers
    behControlPan->m_pid->SetKp(m_panP);
    behControlPan->m_pid->SetKi(m_panI);
    behControlPan->m_pid->SetKd(m_panD);
    behControlTilt->m_pid->SetKp(m_tiltP);
    behControlTilt->m_pid->SetKi(m_tiltI);
    behControlTilt->m_pid->SetKd(m_tiltD);
}

void CRobotUIDlg::UpdateTrackingInfo() {
    EnterCriticalSection(&m_cs);

    m_personSearch.m_trackingInfo.IsTargetLocked = m_trackingValid;
    m_personSearch.m_trackingInfo.x_pos = m_x;
    m_personSearch.m_trackingInfo.y_pos = m_y;

m_personSearch.m_trackingInfo.mass = m_mass;

LeaveCriticalSection(&m_cs);
}

void CRobotUIDlg::OnTimer(UINT_PTR nIDEvent) {
    if (nIDEvent != TIMER_ID_HEARTBEAT) {
        // we only care for the heartbeat timer
        return;
    }

    // in every heartbeat we exercise sensors, execute behaviors and execute arbiters
    //
    // exercise sensors
    UpdateTrackingInfo();

    behaviorList->ExecuteAll();

    PerfCounter perfControl;

    // execute arbiters
    // execute behaviors
    panTiltArbiter->m_panEnabled = CheckServoPanEnabled();

    panTiltArbiter->m_tiltEnabled = CheckServoTiltEnabled();

    panTiltArbiter->Arbitrate();

    if (!m_doRobotControl) {
        return;
    }

    segwayArbiter->Arbitrate();

    perfControl.recordEnd("Segway");

    // visual indicator
    char tbuffer[64];

    if (m_beatStatus) {
        strcpy_s(tbuffer, "/");
    } else {
        strcpy_s(tbuffer, \\");
    }

    m_beatStatus = !m_beatStatus;
void CRobotUIDlg::OnColorCheck() {
    CColorName colorName("main_colors.csv",
        m_cListener.m_centerColor.hue * (240 / 360),
        (double)m_cListener.m_centerColor.saturation / 240.0,
        (double)m_cListener.m_centerColor.intensity / 240.0);
    SPEAK("<pitch middle='+5'>This color, is likely to be:<pitch middle='-5'><rate speed='-5'>
    %s<rate speed='+5'>", colorName.GetColorName()
        .c_str());
}

void CRobotUIDlg::OnTraining() {
    CPoint scrCenter(IMAGE_WIDTH / 2, IMAGE_HEIGHT / 2);
    int half = AUTO_TRAINING_SIZE / 2;

    m_cListener.SetSelectionMode(true);
    m_cListener.m_selectionStart.SetPoint(scrCenter.x - half, scrCenter.y - half);

    m_cListener.SetSelectionMode(false);
    m_cListener.m_lastMove.SetPoint(scrCenter.x + half, scrCenter.y + half);

    m_cListener.TrainColorsToSelection();
}

void CRobotUIDlg::OnStopped() {
    OnBnClickedReset();
}

void CRobotUIDlg::OnConfig() {
    OnBnClickedChangeColor();
}

void CRobotUIDlg::OnBnClickedChangeColor() {
    m_params.satTolerance = s_settings[s_next_setting].satTolerance;
    m_targetIntensity = s_settings[s_next_setting].targetIntensity;
    UpdateData(FALSE);
    /*SetDlgItemInt(IDC_THRESH_SAT_TOL, m_params.satTolerance);
     SetDlgItemInt(IDC_TARGET_INT, m_targetIntensity);*/
SPEAK("<pitch middle='+5'>Configuration<pitch middle='-5'><rate speed='5'> %s<rate speed='+5'>", s_settings[s_next_setting].speakText);

if (++s_next_setting >= NUM_SETTINGS) {
    s_next_setting = 0;
}

void CRobotUIDlg::OnBnClickedSaveImages() {
    m_saveImages = IsDlgButtonChecked(IDC_SAVE_IMAGES) ? TRUE : FALSE;
    GetDlgItemText(IDC_SAVE_IMAGES_PATH, m_saveImagesPath);
}

void CRobotUIDlg::ToggleLogging() {
    CheckDlgButton(IDC_SAVE_IMAGES, GetLoggingState() ? 0 : 1);
    OnBnClickedSaveImages();
}

bool CRobotUIDlg::GetLoggingState() {
    return IsDlgButtonChecked(IDC_SAVE_IMAGES) ? true : false;
}
/* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* Spring 2012
* TargetTracker.h: This class maintains attributes of the target that is being
* tracked. Various info about the target is stored, the state of tracking is
* maintained, and utility functions are defined here.
* Author: Brian Blaine, Padu Merlotti
** -------------------------------------------------------------------------*/
#ifndef TARGET_TRACKER_H
#define TARGET_TRACKER_H

#include <list>
#include "Thresholder.h"
#include "Scoring.h"
#include "Scorers.h"
#include "ParameterProvider.h"
#include "RegionGrower.h"
#include "ImageRegionsInfo.h"

using namespace std;

struct TargetInfo {
    double x;       // -1..1, relative to center
    double y;       // -1..1, relative to center
    double mass;    // -1..1, relative to training mass
    double mass_pixels;     // current mass in pixels
    int mass_training;      // training mass in pixels
    int width;            // in pixels
    int height;           // in pixels
    int x_centroid;       // in pixels
    int y_centroid;       // in pixels
    int weightedXCentroid;
    int weightedYCentroid;
    int meanTrainingHue;
    int meanTrainingSat;
    int meanTrainingInt;
    int hueSkew;
}
std::map<WORD, int> hueHistogramTraining;
std::map<WORD, int> saturationHistogramTraining;
};

struct HueSaturationStats {
    double hue_standard_deviation;
    double saturation_standard_deviation;
    double hue_mean;
    double saturation_mean;
};

class TargetTracker : public ImageLib::IScorerLimitProvider {

public:
    TargetTracker(IParameterProvider * params);
    void Reset();
    void Train(CRect& rectSelected, ImageLib::IImage * image);
    void ProcessFrame(ImageLib::IImage * image);
    void DrawArtifacts(ImageLib::IImage * image);
    bool GetTargetInfo(TargetInfo& info);
    void ComputeProjectedTargetInfo();
    void RecalcHueSkew(int measuredHue);
    void ResetHueSkew();
    void LogColorsIfNeeded(FILE * outFile, bool force);
    // returns true if adjusting the lighting is OK
    bool OKToAdjustLuminance();
    bool IsTrained();
    int fileIndex;

private:
    enum TrackingStateType {
        TargetLost,
        TargetProjecting,
        TargetTracking,
        TargetNone
    };
    void changeToState(TrackingStateType newState);
    void updateTargetInfo(ImageLib::Region * winner, double effectiveMass, ImageLib::Point colorMatchedRegionWeightedCentroid);
    void restrictThresholdingAroundTargetArea();
    void drawRestrictionRect(ImageLib::IImage * image);
    void resetThresholdArea();
void resetLastInfo(bool resetLocation);

double GetMaxDistanceAllowed();

static double clamp(double val, double min, double max);

TrackingStateType m_targetState;
DWORD m_targetLastSeen;
ImageLib::Thresholder m_thresholder;
bool m_colorsChanged;
ImageLib::ScorerManager m_scoreManager;
ImageLib::Averager m_xyAverages;
ImageLib::Averager m_massAverages;
ImageLib::DerivativeManager m_derivatives;

ImageLib::RegionGrower m_regionGrower;
IParameterProvider * m_paramProvider;

CRect m_rectForThresholding;
double m_dx;
double m_dy;
double m_uncertainty;   // 1.0 = certain, > 1.0 more uncertain of loc

double m_allowedXYDistance;
int m_hueSkewHistory[20];
int m_numHueEntries;

TargetInfo m_lastInfo;

ImageLib::HSIDataNormalized trainingHSINormalized;
ImageLib::HSIDataNormalized currentHSINormalized;

// Used to track previous hue values seen so we can discern between 0 and 360 degrees hue in later calculations.
list<int> hueHistory;

HueSaturationStats getSelectionStats(CRect& rectSelected, ImageLib::IImage * image);
HueSaturationStats getDetectedRegionStats(CRect& rectSelected, ImageLib::IImage * image,
std::vector<ImageLib::Region *> regions);
}
* tracked. Various info about the target is stored, the state of tracking is
* maintained, and utility functions are defined here.
*  
* Author: Brian Blaine, Padu Merlotti
** -------------------------------------------------------------------------*/
#include "stdafx.h"
#include "PerfCounter.h"
#include "TargetTracker.h"
#include "Tracing.h"
#include "Image.h"
#include "Scorers.h"
#include "ColorData.h"
#include "MassScorer.h"
#include "XScorer.h"
#include "YScorer.h"
#include "RectangularityScorer.h"
#include "TallnessScorer.h"
#include "RegionSizeScorer.h"
#include "gsl\gsl_statistics.h"
#include "FreemanChainCode.h"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace ImageLib;
using namespace std;

#define THRESH_FUDGE_H 45
#define THRESH_FUDGE_W 45
#define DEFAULT_WIDTH IMAGE_WIDTH
#define DEFAULT_HEIGHT IMAGE_HEIGHT
#define MAX_MASS (IMAGE_WIDTH * IMAGE_HEIGHT / 3)
#define UNCERTAINTY_INCREMENT 0.1
#define JUMP_FACTOR_LIMITED 0.4
#define JUMP_FACTOR_UNLIMITED 2.0
#define MASS_LUMINANCE_MIN 1000
#define CONVERGE_TO_CENTER_SCALE 0.95

TargetTracker::TargetTracker(IParameterProvider * params)
  : m_paramProvider(params)
  , m_xyAverages(3)
  , m_massAverages(10)
  , m_dx(0.0)
  , m_dy(0.0)
  , m_targetState(TargetNone)
  , m_allowedXYDistance(JUMP_FACTOR_UNLIMITED)
  , m_numHueEntries(0)
bool TargetTracker::IsTrained() {
    return m_thresholder.hasColors();
}

void TargetTracker::changeToState(TrackingStateType newState) {
    static char * stateNames[] = {
        "TargetLost",
        "TargetProjecting",
        "TargetTracking",
        "TargetNone"
    };

    if (newState != m_targetState) {
        DO_TRACE(TraceLib::TrackingState, "Changing from state %s to %s",
                  stateNames[m_targetState], stateNames[newState]);
        m_targetState = newState;
    }
}

void TargetTracker::ProcessFrame(ImageLib::IImage * image){

    Region * largestRegion;
    resetLastInfo(false);

    if (!m_thresholder.hasColors()) {
        return;
    }

    //Clear the regions grown from the last iteration.
    m_regionGrower.clear();

    ImageLib::ThresholdParams params;
    if (m_paramProvider) {
        params = m_paramProvider->GetParams();
    }

    if (!m_paramProvider->SkewHue()) {
        m_lastInfo.hueSkew = 0;
    }
// Threshold the image

DO_TRACE(TraceLib::Thresholding, "Params - hue tolerance %d, intensity tolerance %d, saturation tolerance %d", params.hueTolerance, params.intTolerance, params.satTolerance);

PerfCounter perfThreshold;
PerfCounter perfRg;

m_thresholder.threshold(image, m_lastInfo.hueSkew, params, m_rectForThresholding.left, m_rectForThresholding.top,
                      m_rectForThresholding.right,
                      m_rectForThresholding.bottom);

// Save the thresholded image
std::stringstream sstm;
sstm << "threshold" << fileIndex << ".bmp";
string fileName = sstm.str();

// End save the thresholded image

int effectiveMass = 0;
Point colorMatchWeightedCentroid;

DO_TRACE(TraceLib::Algorithm, "Regions detected?: %d", detectedRegionsInfo.getAllRegions().size());

if (detectedRegionsInfo.getAllRegions().empty() == false) {
    Point normalizedCentroid = m_regionGrower.getInfo().getWeightedCentroid(detectedRegionsInfo);
    DO_TRACE(TraceLib::Algorithm, "Normalized centroid: %d %d", normalizedCentroid.x, normalizedCentroid.y);
std::vector<Region *> regions = m_regionGrower.getInfo().getRegionsBetween(100, 9000);

for (std::vector<Region *>::iterator regions_iter = regions.begin(); regions.end() !=
regions_iter; ++regions_iter) {
    image->drawCircle((*regions_iter)->getCentroid().x, (*regions_iter)->getCentroid().y,
1.0, RGBData(0, 255, 0));
    DO_TRACE(TraceLib::Algorithm, "Region centroid: %d %d", (*regions_iter)-
>getCentroid().x, (*regions_iter)->getCentroid().y);
    DO_TRACE(TraceLib::Algorithm, "Region area: %d ", (*regions_iter)-
>getNumberOfPoints());
}

double weight = 0.8;
Point colorMatchWeightedCentroid =
    m_regionGrower.getInfo().getColorMatchWeightedCentroid(detectedRegionsInfo,
m_lastInfo.hueHistogramTraining,
m_lastInfo.saturationHistogramTraining,
weight);

m_lastInfo.colorMatchRegionWeightedCentroid = colorMatchWeightedCentroid;

image->drawCircle(colorMatchWeightedCentroid.x, colorMatchWeightedCentroid.y, 5.0,
RGBData(133, 14, 207));
    DO_TRACE(TraceLib::Algorithm, "Color match weighed centroid: %d %d", 
colorMatchWeightedCentroid.x, colorMatchWeightedCentroid.y);

effectiveMass =
    m_regionGrower.getInfo().getEffectiveMass(detectedRegionsInfo,
m_lastInfo.hueHistogramTraining,
m_lastInfo.saturationHistogramTraining,
weight);
    DO_TRACE(TraceLib::Algorithm, "Effective mass: %f", effectiveMass);

largestRegion = m_regionGrower.getInfo().getLargestRegion();

    DO_TRACE(TraceLib::Algorithm, "Updating target info with centroid (%d, %d) and mass %f.", 
colorMatchWeightedCentroid.x, 
colorMatchWeightedCentroid.y, 
effectiveMass);

m_scoreManager.computeScores(m_thresholder.getResult(), m_regionGrower.getInfo());
updateTargetInfo(largestRegion, effectiveMass, colorMatchWeightedCentroid);
} else {
    ComputeProjectedTargetInfo();
}
void TargetTracker::updateTargetInfo(ImageLib::Region *largestRegion, int effectiveMass, ImageLib::Point colorMatchedRegionWeightedCentroid){

    m_allowedXYDistance = JUMP_FACTOR_LIMITED;
    changeToState(TargetTracking);
    m_targetLastSeen = GetTickCount();

    ScoreValuesType scores = m_scoreManager.getScores();
    m_xyAverages.computeAndReplace(scores);
    m_massAverages.computeAndReplace(scores);
    m_derivatives.computeAndAddDerivatives(scores);

    m_uncertainty = 1.0;

    m_dx = scores["dx"];  
    m_dy = scores["dy"];
    m_lastInfo.x = scores["x"];  
    m_lastInfo.y = scores["y"];  
    m_lastInfo.mass = scores["mass"];  
    m_lastInfo.width = largestRegion->getWidth();
    m_lastInfo.height = largestRegion->getHeight();

    m_lastInfo.mass_pixels = effectiveMass;
    m_lastInfo.x_centroid = colorMatchedRegionWeightedCentroid.x;
    m_lastInfo.y_centroid = colorMatchedRegionWeightedCentroid.y;
}

void TargetTracker::resetLastInfo(bool resetLocation){
    if (resetLocation) {
        m_lastInfo.x = m_lastInfo.y = 0;

        m_lastInfo.x_centroid = IMAGE_WIDTH / 2;
        m_lastInfo.y_centroid = IMAGE_HEIGHT / 2;
    }
    m_lastInfo.width = DEFAULT_WIDTH;
    m_lastInfo.height = DEFAULT_HEIGHT;
}

void TargetTracker::ComputeProjectedTargetInfo(){
    if (TargetProjecting != m_targetState && TargetTracking != m_targetState) {
        resetLastInfo(true);
    }
m_allowedXYDistance = JUMP_FACTOR_UNLIMITED;
return;
}

DWORD reaquireTimeout = m_paramProvider->GetTargetForecastTimeMS();
DWORD dwNow = GetTickCount();
if (dwNow - m_targetLastSeen > reaquireTimeout / 2) {
    // halfway to losing him - allow jumps
    m_allowedXYDistance = JUMP_FACTOR_UNLIMITED;
}

if (dwNow - m_targetLastSeen > reaquireTimeout) {
    // we lost him - give up
    changeToState(TargetLost);
    m_derivatives.clear();
    m_xyAverages.clear();
    m_massAverages.clear();
    m_allowedXYDistance = JUMP_FACTOR_UNLIMITED;
    resetLastInfo(true);
} else {
    changeToState(TargetProjecting);
    // we know the robot has been acting on our outputs, trying to center the
    // target in the camera. So, the right thing to do is to feed out a position
    // that assumes it is converging to 0, 0 in about 1/2 second. At 15 frames/sec
    // that should happen by multiplying by CONVERGE_TO_CENTER_SCALE every time
    m_lastInfo.x *= CONVERGE_TO_CENTER_SCALE;
    m_lastInfo.y *= CONVERGE_TO_CENTER_SCALE;
    m_lastInfo.x_centroid = (short)((m_lastInfo.x + 1) * IMAGE_WIDTH / 2);
    m_lastInfo.y_centroid = (short)((m_lastInfo.y + 1) * IMAGE_HEIGHT / 2);
    m_uncertainty = clamp(m_uncertainty + UNCERTAINTY_INCREMENT, 1.0, 3.0);
    // leave mass alone
    // TODO
    // add the new values to the averager?
}
// tell the scorers that the reference values have changed
m_scoreManager.updateScorer("x", m_lastInfo.x);
    m_scoreManager.updateScorer("y", m_lastInfo.y);
}

double TargetTracker::clamp(double val, double min, double max){
    if (val < min) {
        return min;
    }
    if (val > max) {
        return max;
    }
    return val;
void TargetTracker::RecalcHueSkew(int measuredHue) {
    if (TargetTracking == m_targetState) {
        // cases
        // train: 350, measured = 10 -> skew = -20
        // train: 10, measured = 350 -> skew = 20
        // train: 350, measured = 330 -> skew = 20
        // train: 330, measured = 350 -> skew = -20
        int skewMagnitude = abs(m_lastInfo.meanTrainingHue - measuredHue);
        int direction = 1;
        if (skewMagnitude > 120) {
            if (m_lastInfo.meanTrainingHue > measuredHue) {
                direction = -1;
            }
            skewMagnitude = 240 - skewMagnitude;
        } else if (m_lastInfo.meanTrainingHue < measuredHue) {
            direction = -1;
        }
    } /* ifdef _DEBUG */
    const int NUM_ENTRIES = sizeof(m_hueSkewHistory) / sizeof(m_hueSkewHistory[0]);
    if (0 == m_numHueEntries) {
        // initialize
        for (int i = 0; i < NUM_ENTRIES; i++)
            m_hueSkewHistory[m_numHueEntries++] = 0;
    }
    m_hueSkewHistory[(m_numHueEntries++) % NUM_ENTRIES] = direction * skewMagnitude;
    m_lastInfo.hueSkew = 0;
    for (int i = 0; i < NUM_ENTRIES; i++)
        m_lastInfo.hueSkew += m_hueSkewHistory[i];
    m_lastInfo.hueSkew /= NUM_ENTRIES;
}

void TargetTracker::ResetHueSkew() {
    m_lastInfo.hueSkew = 0;
    m_numHueEntries = 0;
}
void TargetTracker::Reset()
{
    m_thresholder.clearTraining();
    ResetHueSkew();
    resetThresholdArea();
    changeToState(TargetNone);
    //m_xyAverages.clear();
    //m_massAverages.clear();
    //m_derivatives.clear();
    //m_scoreManager.clear();
    m_allowedXYDistance = JUMP_FACTOR_UNLIMITED;
    resetLastInfo(true);
}

bool TargetTracker::GetTargetInfo(TargetInfo& info)
{
    if (TargetProjecting == m_targetState || TargetTracking == m_targetState) {
        info = m_lastInfo;
        return true;
    }
    return false;
}

void TargetTracker::Train(CRect& rectSelected, ImageLib::IImage * image)
{
    Reset();

    if (rectSelected.IsRectEmpty()) {
        return;
    }

    PerfCounter perfImageTrain;

    HueSaturationStats hueSaturationStats = getSelectionStats(rectSelected, image);

    m_thresholder.saturationStandardDeviation =
        hueSaturationStats.saturation_standard_deviation;
    m_thresholder.hueStandardDeviation = hueSaturationStats.hue_standard_deviation;
    m_thresholder.hueMean = hueSaturationStats.hue_mean;
    m_thresholder.saturationMean = hueSaturationStats.saturation_mean;

    m_thresholder.train(rectSelected.left, rectSelected.top, rectSelected.right,
                        rectSelected.bottom, image);

    perfImageTrain.recordEnd("TrainingColors");

    if (!m_thresholder.hasColors()) {

changeToState(TargetNone);
return;
}

m_colorsChanged = true;
resetThresholdArea();

ImageLib::ThresholdParams params;

if (m_paramProvider) {
    params = m_paramProvider->GetParams();
}

//std::vector<HSIData> hsiDatas = m_thresher.getTargetHSIEllipseColors();
//DO_TRACE(TraceLib::ChainCode, "HSI datas[0]: %d %d %d", hsiDatas.at(0).hue, hsiDatas.at(0).hue, hsiDatas.at(0).hue);

//FreemanChainCode freemanChainCode(rectSelected.top, rectSelected.bottom, rectSelected.left, rectSelected.right, image, hsiDatas);
//std::vector<int> chain = freemanChainCode.getEightChainCode();

// threshold the image
PerfCounter perfThreshold;

m_thresher.threshold(image,
    0,
    params,
    m_rectForThresholding.left, m_rectForThresholding.top,
    m_rectForThresholding.right, m_rectForThresholding.bottom);

perfThreshold.recordEnd("Threshold");

PerfCounter perfRg;

m_regionGrower.growRegions(
    m_thresher.getResult(), m_rectForThresholding.left, m_rectForThresholding.top,
    m_rectForThresholding.right,
    m_rectForThresholding.bottom);

perfRg.recordEnd("Regions");

ImageRegionsInfo detectedRegionsInfo = m_regionGrower.getInfo();

Point normalizedCentroid =
m_regionGrower.getInfo().getWeightedCentroid(detectedRegionsInfo);
double hueHistogramDelta =
detectedRegionsInfo.getHistogramsDelta(m_lastInfo.hueHistogramTraining).hueHistogramDelta;

double saturationHistogramDelta =
detectedRegionsInfo.getHistogramsDelta(m_lastInfo.saturationHistogramTraining).saturationHistogramDelta;

double weight = 0.8;

Point colorMatchWeightedCentroid =
   m_regionGrower.getInfo().getColorMatchWeightedCentroid(detectedRegionsInfo,
m_lastInfo.hueHistogramTraining,

m_lastInfo.saturationHistogramTraining,
   weight);

int effectiveMass = (int) m_regionGrower.getInfo().getEffectiveMass(detectedRegionsInfo,
m_lastInfo.hueHistogramTraining,

m_lastInfo.saturationHistogramTraining,
   weight);

///

//Get stats for the regions that were detected.
/*saturationStandardDeviation = getSaturationStandardDeviation(rectSelected, image, regions);
hueStandardDeviation = getHueStandardDeviation(rectSelected, image, regions);
hueMean = getHueMean(rectSelected, image, regions);
saturationMean = getSaturationMean(rectSelected, image, regions);*/

POINT ptCenter = rectSelected.CenterPoint();
Region * region = m_thresholder.getResult().getPixel(ptCenter.x, ptCenter.y).getOwner();

if (!region) {
    DO_TRACE(TraceLib::Train, "WARNING: None matched training?\n");
    changeToState(TargetNone);
    return;
}

//Occasional segfault here. Not sure what it is.
m_scoreManager.addScorer("x", new XScorer(this), 100.0, m_thresholder.getResult(),
detectedRegionsInfo);
m_scoreManager.addScorer("y", new YScorer(this), 100.0, m_thresholder.getResult(),
detectedRegionsInfo);
m_xyAverages.addItem("mass");
m_derivatives.addDerivative("dx", "x");
m_derivatives.addDerivative("dy", "y");
m_derivatives.addDerivative("dmass", "mass");

Region * largestRegion = detectedRegionsInfo.getLargestRegion();

if (!largestRegion) {
    DO_TRACE(TraceLib::Train, "Warning: No regions!\n");
    Reset();
    return;
}

m_lastInfo.mass_training = effectiveMass;
m_lastInfo.hueHistogramTraining = largestRegion->getHistograms().hue;
m_lastInfo.saturationHistogramTraining = largestRegion->getHistograms().saturation;

map<WORD, int>::iterator iterator;
for (iterator = m_lastInfo.hueHistogramTraining.begin(); iterator != m_lastInfo.hueHistogramTraining.end(); iterator++) {
    DO_TRACE(TraceLib::Train, "hue %d has a value of %d", iterator->first, iterator->second);
}

for (iterator = m_lastInfo.saturationHistogramTraining.begin(); iterator != m_lastInfo.saturationHistogramTraining.end(); iterator++) {
    DO_TRACE(TraceLib::Train, "saturation %d has a value of %d", iterator->first, iterator->second);
}

updateTargetInfo(largestRegion, effectiveMass, colorMatchWeightedCentroid);

restrictThresholdingAroundTargetArea();

/* When we restrict thresholding, we must consider that an off-center target will
cause the robot to try to center it. Therefore, it is most likely to be closer
to the center, where it is, or somewhere in between (if he's moving fast across
the field of view). So, take two rectangles - where he is, and him centered, then
enclose them both with a rectangle.
*/
void TargetTracker::restrictThresholdingAroundTargetArea(){
    int x = (int)((m_lastInfo.x + 1) * IMAGE_WIDTH / 2);
    int y = (int)((m_lastInfo.y + 1) * IMAGE_HEIGHT / 2);
    int w = m_lastInfo.width;
    int h = m_lastInfo.height;
CRect whole(0, 0, IMAGE_WIDTH - 1, IMAGE_HEIGHT - 1);
CRect himNow(x - w / 2, y - h / 2, x + w / 2 + 1, y + h / 2 + 1);
CRect himCentered(IMAGE_WIDTH / 2 - w / 2, IMAGE_HEIGHT / 2 - h / 2, IMAGE_WIDTH / 2 + w / 2, IMAGE_HEIGHT / 2 + h / 2);
m_rectForThresholding.UnionRect(&himNow, &himCentered);
// add a general fudge factor
m_rectForThresholding.InflateRect((int)(THRESH_FUDGE_W * m_uncertainty),
(int)(THRESH_FUDGE_H * m_uncertainty));

// constrain to the image bounds
CRect copy = m_rectForThresholding;
m_rectForThresholding.IntersectRect(copy, whole);
}

void TargetTracker::drawRestrictionRect(ImageLib::IImage * image){
image->drawRectangle(m_rectForThresholding.left, m_rectForThresholding.top,
    m_rectForThresholding.right, m_rectForThresholding.bottom,
    RGBData::white());
}

void TargetTracker::resetThresholdArea(){
    m_rectForThresholding = CRect(0, 0, IMAGE_WIDTH, IMAGE_HEIGHT);
}

void TargetTracker::DrawArtifacts(ImageLib::IImage * image){
    switch (m_targetState) {
    case TargetLost:
        // just draw the restraining rect
        drawRestrictionRect(image);
        break;
    case TargetProjecting: // draw center in red
        image->drawCenterMark(m_lastInfo.x_centroid, m_lastInfo.y_centroid, RGBData(255, 0, 0));
        drawRestrictionRect(image);
        break;
    case TargetTracking:
        m_regionGrower.drawRegions(image, true);
        image->drawCenterMark(m_lastInfo.x_centroid, m_lastInfo.y_centroid, RGBData(255, 255, 255));
        drawRestrictionRect(image);
        break;
    case TargetNone:
        // nothing
        break;
    }
bool TargetTracker::OKToAdjustLuminance()
{
    if (TargetTracking == m_targetState) {
        bool retval = m_lastInfo.mass_pixels >= MASS_LUMINANCE_MIN;
        if (!retval) {
            DO_TRACE(TraceLib::CameraControl, "Mass too small for luminance (%d)",
                     m_lastInfo.mass_pixels);
        }
        return retval;
    }
    return TargetProjecting != m_targetState;
}

double TargetTracker::GetMaxDistanceAllowed()
{
    return m_allowedXYDistance;
}

void TargetTracker::LogColorsIfNeeded(FILE * outFile, bool force){
    if (m_colorsChanged || force) {
        m_thresholder.logColors(outFile);
        m_colorsChanged = false;
    }
}

/*
 * Gets the standard deviation and mean for saturation and hue from the regions that fall
 * within the selection training rectangle.
 * *
 * 1) Iterate through all the regions.
 * 2) Iterate through all the points of a region.
 * 3) Check to see if the point is within selection rectangle. If so, add it to array.
 * 4) Calculate standard deviation and mean on array.
 */
HueSaturationStats TargetTracker::getDetectedRegionStats(CRect &rectSelected,
                  ImageLib::IImage * image, std::vector<Region *> regions){
    HueSaturationStats results;

    //Allocate the max possible space not all of which is used.
    double * saturations = new double[(rectSelected.bottom - rectSelected.top) *
                                       (rectSelected.right - rectSelected.left)];
    double * hues = new double[(rectSelected.bottom - rectSelected.top) * (rectSelected.right -
                                rectSelected.left)];
int index = 0;

for (std::vector<Region *>::iterator regions_iter = regions.begin(); regions.end() !=
regions_iter; ++regions_iter) {
    std::vector<Point> points = (*regions_iter)->getPoints();

    for (std::vector<Point>::iterator points_iter = points.begin(); points.end() !=
points_iter; ++points_iter) {
        int x = (*points_iter).x;
        int y = (*points_iter).y;

        if (x > rectSelected.left && x < rectSelected.right && y > rectSelected.top && y <
rectSelected.bottom) {
            saturations[index] = (double) image->getRGB(x, y).toHSI().saturation;
            hues[index] = (double) image->getRGB(x, y).toHSI().hue;
            index++;
        }
    }
}

int size = index;

results.saturation_mean = gsl_stats_mean(saturations, 1, size);
results.hue_mean = gsl_stats_mean(hues, 1, size);
results.hue_standard_deviation = gsl_stats_sd(hues, 1, size);
results.saturation_standard_deviation = gsl_stats_sd(saturations, 1, size);

delete [] hues;
delete [] saturations;

return results;
}

HueSaturationStats TargetTracker::getSelectionStats(CRect &rectSelected, ImageLib::IImage *
image){
    HueSaturationStats results;

    double * hues = new double[(rectSelected.bottom - rectSelected.top) * (rectSelected.right -
rectSelected.left)];
    double * saturations = new double[(rectSelected.bottom - rectSelected.top) *
(rectSelected.right - rectSelected.left)];

    int index = 0;

    int x, y;
for (x = rectSelected.left; x < rectSelected.right; x++) {
    for (y = rectSelected.top; y < rectSelected.bottom; y++) {
        hues[index] = (double) image->getRGB(x, y).toHSI().hue;
        saturations[index] = (double) image->getRGB(x, y).toHSI().saturation;
        index++;
    }
}

int size = index;

results.hue_standard_deviation = gsl_stats_sd(hues, 1, size);
results.hue_mean = gsl_stats_mean(hues, 1, size);
results.saturation_standard_deviation = gsl_stats_sd(saturations, 1, size);
results.saturation_mean = gsl_stats_mean(saturations, 1, size);

DO_TRACE(TraceLib::Train, "HSD HM SSD SM %f %f %f %f", results.hue_standard_deviation, results.hue_mean, results.saturation_standard_deviation, results.saturation_mean);

delete [] hues;
delete [] saturations;

return results;
/* -----------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
*
* Spring 2012
*
* Author: Brian Blaine, Padu Merlotti
*
* Listener.h: interface for the CListener class.
*
* The CListener class is derived from GrabberListener. It overwrites
* the "frameReady()" method. In the frameReady method, the method
* "saveImage()" is called.
* "saveImage()" saves the specified buffer to a BMP file and calls a "Sleep(250)"
* to simulate time consuming image processing. "saveImage()" is also called
* by the main() function of this example to save all buffers that have
* not been processed in the frameReady method.
*
* This class also overwrites the overlayCallback method to draw a
* frame counter.
*
* The CListener object is registered with the parameter
* eFRAMEREADY|eOVERLAYCALLBACK.
***************************************************************************/

#if !defined(AFX_LISTENER_H__3E017E1D_6B0A_472C_9F9C_0C5F9A8DFB23__INCLUDED_)
#define AFX_LISTENER_H__3E017E1D_6B0A_472C_9F9C_0C5F9A8DFB23__INCLUDED_
#endif // _MSC_VER > 1000

#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>
#include "tisudshl.h"
#include "ImageResizeAdapter.h"
#include "ParameterProvider.h"
#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>
#include "tisudshl.h"
#include "ImageResizeAdapter.h"
#include "ParameterProvider.h"
#include "SimplePropertyAccess.h"
#include "TargetTracker.h"
#include "PIDController.h"
#include "ColorStats.h"

#define MESSAGEDEVICELOST WM_USER + 90

class IScoreHandler {

public:
    virtual void HandleTargetInfo(TargetInfo& scores) = 0;
    virtual void HandleTargetLost() = 0;
    virtual void HandleColorStats(int hueMean, int hueVar, int satMean, int satVar, int intMean, int intVar) = 0;
    virtual void HandleLuminanceStats(bool gainAuto, bool expAuto, int gain, int exp) = 0;
};

using namespace DShowLib;

class CListener : public GrabberListener {

public:
    void SetViewCWnd(CWnd *pView);
    CListener();
    virtual ~CListener();
    virtual void deviceLost(Grabber& param);
    void SetParent(CWnd * pParent);
    void SetScoreProcessor(IScoreHandler * handler);
    void SetParamProvider(IParameterProvider * provider);
    virtual void frameReady(Grabber& param, smart_ptr<MemBuffer> pBuffer, DWORD FrameNumber);

    void GetGainExpProperties(int& minGain, int& maxGain, int& minExp, int& maxExp, BOOL& gainAuto, BOOL& expAuto);

    void ResetTraining();

    void SetSelectionMode(bool selecting);
    bool GetSelectionMode() const {
        return m_selecting;
    }

    CPoint m_selectionStart;
    CPoint m_lastMove;
    HSIData m_centerColor;
    BITMAPINFO m_imageHdr;

    void TrainColorsToSelection();
CSimplePropertyAccess& getProperties() {
    return m_VCDProp;
}

protected:
  CWnd * m_pParent;
  CWnd * m_pDrawCWnd;
  SIZE m_WindowSize;
  bool m_selecting;
  // An instance of the helper class for camera parameter control
  CSimplePropertyAccess m_VCDProp;
  
  void DrawResult();
  void StoreCenterColor();
  void DoImageProcessing();
  bool ControlLuminance(int x, int y);
  void addIntensities(int xCenter, int yCenter, int size);
  
  void GetSelectionLRTB(int& l, int& r, int& t, int& b);
  
  void SaveBitmapToFile(BYTE * pBitmapBits, LONG lWidth, LONG lHeight, WORD wBitsPerPixel, LPCTSTR lpszFileName);
  void LogToPath(const char * path);
  void WriteStats();
  void ClearLoggingState();

  struct LoggingStats {
    int meanIntensity;
    int meanHue;
    int meanSat;
    int mass;
    int x;
    int y;
    int newGain;
    int newExp;
    bool okToAdjustLuminance;
    bool adjustedLuminance;
    bool tracking;
  };

  LoggingStats m_stats;

  TargetTracker * m_tracker;

  IScoreHandler * m_scoreHandler;
  ImageLib::Point m_centroid;
IParameterProvider * m_paramProvider;
void InitCameraSettings();
bool m_trainColorsPending;
bool m_resetTrainPending;
CImageResizeAdapter m_resultImage;

// store the bounds for the gain & exposure settings
long m_expMin;
long m_expMax;
long m_gainMin;
long m_gainMax;
bool m_settingsInitialized;
bool m_gainExpAuto;
double m_gainToExpControlRatio;

DWORD m_lastCameraControlTime;

// the controller for the luminance
CPIDController m_cameraPid;
// ColorStatsIntensityOnlyATM m_intensityStats;
ColorStatsFull m_intensityStats;
);
#endif // !defined(AFX_LISTENER_H__3E017E1D_6B0A_472C_9F9C_0C5F9A8DFB23__INCLUDED_)

/* ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* Spring 2012
* listener.cpp: This class implements the CListener class from the Imaging Source
* library. This library receives images from the camera. */
Author: Brian Blaine, Padu Merlotti
** -------------------------------------------------------------------------*/
//////////////////////////////////////////////////////////////////////
// CListener: Listener.cpp: implementation of the CListener class.  
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Listener.h"
#include "StreamImageAdapter.h"
#include "PerfCounter.h"
#include "ColorStats.h"
#include <direct.h>

#define EXP_ADJ_TARGET 100
#define EXP_ADJ_MIN 45
#define EXP_ADJ_MAX 1000
#define CAMERA_CONTROL_INTERVAL_MS 100

/* The sampling area for luminace control is a grid in the middle 
 and 4 smaller grids around it*/
#define LUM_LARGE_GRID_SIZE 8
#define LUM_SMALL_GRID_SIZE 2

/* 5 between the grids */
#define LUM_SMALL_GRID_OFFSET (5 + LUM_LARGE_GRID_SIZE / 2 + LUM_SMALL_GRID_SIZE / 2)

// The amount of error that determines if we bother to control luminance
#define LUM_CONTROL_THRESHOLD 0

//The intentity readings above which we regard as noise
#define INTENSITY_LIMIT (int)(0.8 * 240)

using namespace ImageLib;

CListener::CListener()
: m_resultImage(IMAGE_WIDTH, IMAGE_HEIGHT)
, m_resetTrainPending(false)
, m_settingsInitialized(false)
, m_gainExpAuto(false)
, m_expMin(EXP_ADJ_MIN)
, m_expMax(EXP_ADJ_MAX)
, m_lastCameraControlTime(GetTickCount())
, m_tracker(NULL)
, m_intensityStats(0, 0) {
  m_pParent = NULL;
m_scoreHandler = NULL;
m_paramProvider = NULL;
m_pDrawCWnd = NULL;
m_selecting = false;
m_trainColorsPending = false;
m_imageHdr.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
m_imageHdr.bmiHeader.biWidth = IMAGE_WIDTH;
m_imageHdr.bmiHeader.biHeight = -IMAGE_HEIGHT;
m_imageHdr.bmiHeader.biPlanes = 1;
m_imageHdr.bmiHeader.biBitCount = (unsigned short) 24;
m_imageHdr.bmiHeader.biCompression = BI_RGB;
m_imageHdr.bmiHeader.biSizeImage = 0;
m_imageHdr.bmiHeader.biXPelsPerMeter = 0;
m_imageHdr.bmiHeader.biYPelsPerMeter = 0;
m_imageHdr.bmiHeader.biClrUsed = 0;
m_imageHdr.bmiHeader.biClrImportant = 0;
}

CListener::~CListener() {
    if (m_tracker) {
        delete m_tracker;
    }
}

void CListener::SetParent(CWnd *pParent) {
    m_pParent = pParent;
}

void CListener::GetGainExpProperties(int &minGain, int &maxGain,
    int &minExp, int &maxExp,
    BOOL &gainAuto, BOOL &expAuto) {
    InitCameraSettings();
    minGain = (int)m_gainMin;
    maxGain = (int)m_gainMax;
    minExp = (int)m_expMin;
    maxExp = (int)m_expMax;
    gainAuto = getProperties().getAuto(VCDID_Gain);
    expAuto = getProperties().getAuto(VCDID_Exposure);
}

void CListener::InitCameraSettings() {
    if (m_settingsInitialized) {
        return;
    }

    m_settingsInitialized = true;
m_expMin = getProperties().getRangeMin(VCDID_Exposure);

m_expMax = getProperties().getRangeMax(VCDID_Exposure);

m_gainMin = getProperties().getRangeMin(VCDID_Gain);

m_gainMax = getProperties().getRangeMax(VCDID_Gain);

// constrain the exposure further
if (m_expMin < EXP_ADJ_MIN) {
    m_expMin = EXP_ADJ_MIN;
}

if (m_expMax > EXP_ADJ_MAX) {
    m_expMax = EXP_ADJ_MAX;
}

if (m_expMax < m_expMin) {
    m_expMax = m_expMin;
}

if (m_expMax - m_expMin > 0) {
    m_gainToExpControlRatio = (double)(m_gainMax - m_gainMin) / (double)(m_expMax - m_expMin);
} else {
    m_gainToExpControlRatio = 1.0;
}

// by default, for them to be auto
getProperties().setAuto(VCDID_Exposure, true);

getProperties().setAuto(VCDID_Gain, true);

m_gainExpAuto = true;

void CListener::SetScoreProcessor(IScoreHandler * handler) {
    m_scoreHandler = handler;
}

void CListener::SetParamProvider(IParameterProvider * provider) {
    m_paramProvider = provider;

    if (NULL != m_tracker) {
        delete m_tracker;
    }
}
m_tracker = new TargetTracker(provider);

void CListener::ResetTraining() {
    m_resetTrainPending = true;
}

// Notify parent, device is lost.
void CListener::deviceLost(Grabber& param) {
    if (m_pParent != NULL) {
        m_pParent->PostMessage(MESSAGEDEVICELOST, 0, 0);
    }
}

// Set the CWnd that will be used to render the resulting image.
void CListener::SetViewCWnd(CWnd *pView) {
    m_pDrawCWnd = pView;
    RECT r;
    m_pDrawCWnd->GetClientRect(&r);
    m_WindowSize.cx = r.right;
    m_WindowSize.cy = r.bottom;
}

// Callback handler.
static void drawRect(IImage * image, int xCenter, int yCenter, int w, int h) {
    image->drawRectangle(xCenter - w / 2, yCenter - h / 2, xCenter + w / 2, yCenter + h / 2, RGBData::black());
}

/* This method is called after the video capture device has copied a new frame into a memory buffer. */
void CListener::frameReady(Grabber& param, smart_ptr<MemBuffer> pBuffer, DWORD FrameNumber) {
    pBuffer->lock();
    PerfCounter perfFrame;
    // Get the bitmap info header from the membuffer. It contains the bits per pixel, // width and height.
    smart_ptr<BITMAPINFOHEADER> pInf = pBuffer->getBitmapInfoHeader();
    // Now retrieve a pointer to the image. For organization of the image data
    BYTE * pImageData = pBuffer->getPtr();
    CStreamImageAdapter image(pImageData, pInf->biWidth, pInf->biHeight);
    m_resultImage.convert(&image);
TargetInfo info;

if (m_tracker->GetTargetInfo(info)) {
    m_centroid.x = info.x_centroid;
    m_centroid.y = info.y_centroid;

    if (m_scoreHandler) {
        m_scoreHandler->HandleTargetInfo(info);
    }

    m_stats.tracking = true;

    m_stats.mass = info.mass_pixels;
} else if (!m_selecting) {  
    m_centroid.x = IMAGE_WIDTH / 2;
    m_centroid.y = IMAGE_HEIGHT / 2;

    if (m_scoreHandler) {
        m_scoreHandler->HandleTargetLost();
    }

    m_stats.tracking = false;

    m_stats.mass = 0;
}

m_stats.x = m_centroid.x;

m_stats.y = m_centroid.y;

m_stats.okToAdjustLuminance = m_tracker->OKToAdjustLuminance();

if (m_selecting || m_stats.okToAdjustLuminance) {
    m_stats.adjustedLuminance = ControlLuminance(m_centroid.x, m_centroid.y);
} else {
    m_stats.adjustedLuminance = false;
}

pBuffer->unlock();

m_tracker->DrawArtifacts(&m_resultImage);
// draw the luminosity mask
// main square
drawRect(&m_resultImage, m_centroid.x, m_centroid.y, LUM_LARGE_GRID_SIZE,
 LUM_LARGE_GRID_SIZE);
// then, the 4 outliers
drawRect(&m_resultImage, m_centroid.x - LUM_SMALL_GRID_OFFSET, m_centroid.y,
 LUM_SMALL_GRID_SIZE, LUM_SMALL_GRID_SIZE);
drawRect(&m_resultImage, m_centroid.x + LUM_SMALL_GRID_OFFSET, m_centroid.y,
 LUM_SMALL_GRID_SIZE, LUM_SMALL_GRID_SIZE);
drawRect(&m_resultImage, m_centroid.x, m_centroid.y - LUM_SMALL_GRID_OFFSET,
 LUM_SMALL_GRID_SIZE, LUM_SMALL_GRID_SIZE);
drawRect(&m_resultImage, m_centroid.x, m_centroid.y + LUM_SMALL_GRID_OFFSET,
 LUM_SMALL_GRID_SIZE, LUM_SMALL_GRID_SIZE);

DrawResult();
const char * path = m_paramProvider->SaveImagesToPath();

if (path) {
    LogToPath(path);
} else {
    ClearLoggingState();
}

perfFrame.recordEnd("Process Frame");

static int s_name = 0;
static unsigned int s_counter = 0;
static FILE * s_outFile = NULL;
static char s_dirName[MAX_PATH];
static DWORD s_logStart = 0;

void CListener::ClearLoggingState() {
    if (s_outFile) {
        s_name = 0;
        s_counter = 0;

        fprintf(s_outFile, "</entries>\n");

        fclose(s_outFile);
        s_outFile = NULL;
    }
}

void CListener::LogToPath(const char * path) {

}
// save very third frame
if (path) {
    /*if (s_counter++ % 3)
        return; // skip all but every third frame
    */

    DO_TRACE(TraceLib::Algorithm, "Begin Image %d", s_name);

    bool newFile = (NULL == s_outFile);

    char buf[MAX_PATH];

    if (NULL == s_outFile) {
        // create the target file

        struct tm *newTime;
        time_t tNow;
        // Get time in seconds
        time(&tNow);
        newTime = localtime(&tNow);
        sprintf_s(s_dirName, sizeof(s_dirName), "%s\%d_%02d%02d%02d",
                  path, newTime->tm_yday, newTime->tm_hour, newTime->tm_min, newTime->tm_sec);
        _mkdir(s_dirName);
        sprintf_s(buf, sizeof(buf), "%s\log.xml", s_dirName);
        s_outFile = fopen(buf, "wt");
        if (s_outFile) {
            fprintf(s_outFile, "<?xml version="1.0" encoding="utf-8"?>
<entries>
");
        }
        s_logStart = GetTickCount();
    }

    if (s_outFile) {
        char baseName[MAX_PATH];
        m_tracker->LogColorsIfNeeded(s_outFile, newFile);
        sprintf_s(baseName, sizeof(baseName), "%08d.bmp", s_name);
        sprintf_s(buf, sizeof(buf), "%s\%s", s_dirName, baseName);
        SaveBitmapToFile(m_resultImage.getData(), IMAGE_WIDTH, IMAGE_HEIGHT, 24, buf);
        fprintf(s_outFile, "<entry time="%u">
<imageFile>%s</imageFile>
",
                GetTickCount() - s_logStart, baseName);
        WriteStats();
        fprintf(s_outFile, "</entry>\n");
    }
}

#define LOG_INT(x) fprintf(s_outFile, "<%s%d</%s>\n", # x, m_stats.x, # x)
#define LOG_BOOL(x) fprintf(s_outFile, "<%s>%s</%s>
", # x, m_stats.x ? "true" : "false", # x)

void CListener::WriteStats() {
    LOG_INT(meanIntensity);
    LOG_INT(meanHue);
    LOG_INT(meanSat);
    LOG_INT(mass);
    LOG_INT(x);
    LOG_INT(y);
    LOG_INT(newGain);
    LOG_INT(newExp);
    LOG_BOOL(okToAdjustLuminance);
    LOG_BOOL(adjustedLuminance);
    LOG_BOOL(tracking);
}

// Draw the image buffer into the DrawCWnd.
void CListener::SaveBitmapToFile(BYTE * pBitmapBits, LONG lWidth, LONG lHeight, WORD wBitsPerPixel, LPCTSTR lpszFileName) {
    BITMAPINFOHEADER bmpInfoHeader = {0};
    // Set the size
    bmpInfoHeader.biSize = sizeof(BITMAPINFOHEADER);
    // Bit count
    bmpInfoHeader.biBitCount = wBitsPerPixel;
    // Use all colors
    bmpInfoHeader.biClrImportant = 0;
    // Use as many colors according to bits per pixel
    bmpInfoHeader.biClrUsed = 0;
    // Store as un Compressed
    bmpInfoHeader.biCompression = BI_RGB;
    // Set the height in pixels
    bmpInfoHeader.biHeight = -lHeight;
    // Width of the Image in pixels
    bmpInfoHeader.biWidth = lWidth;
    // Default number of planes
    bmpInfoHeader.biPlanes = 1;
    // Calculate the image size in bytes
    bmpInfoHeader.biSizeImage = lWidth * lHeight * (wBitsPerPixel / 8);

    BITMAPFILEHEADER bfh = {0};
    // This value should be values of BM letters i.e 0x4D42
    // 0x4D = M 0x42 = B storing in reverse order to match with endian
    bfh.bfType = 0x4D42;
    /* or
    bfh.bfType = ‘B’+(‘M’ << 8);
    // <<8 used to shift ‘M’ to end */
/ Offset to the RGBQUAD
bfh.bfOffBits = sizeof(BITMAPINFOHEADER) + sizeof(BITMAPFILEHEADER);
// Total size of image including size of headers
bfh.bfSize = bfh.bfOffBits + bmpInfoHeader.biSizeImage;
// Create the file in disk to write
HANDLE hFile = CreateFile(lpszFileName, GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

if (!hFile) {   // return if error opening file
  return;
}

DWORD dwWritten = 0;

// Write the File header
WriteFile(hFile, &bfh, sizeof(bfh), &dwWritten, NULL);
// Write the bitmap info header
WriteFile(hFile, &bmpInfoHeader, sizeof(bmpInfoHeader), &dwWritten, NULL);
// Write the RGB Data
WriteFile(hFile, pBitmapBits, bmpInfoHeader.biSizeImage, &dwWritten, NULL);
// Close the file handle
CloseHandle(hFile);

void CListener::DrawResult() {
  if (m_pDrawCWnd != NULL) {
    CDC *pDC = m_pDrawCWnd->GetDC();
    pDC->SetStretchBltMode(COLORONCOLOR);
    //const char* path = m_paramProvider->SaveImagesToPath();
    static int name = 0;
    static unsigned int counter = 0;
    int nLines = StretchDIBits(pDC->GetSafeHdc(),      // Handle to the device
0,
0,
IMAGE_WIDTH,
IMAGE_HEIGHT,
0,
0,
IMAGE_WIDTH,
IMAGE_HEIGHT,
m_resultImage.getData(),     // Modified address of array with
DIB bits.
&m_imageHdr,
DIB_RGB_COLORS,     // RGB or palette indices.
SRCCOPY
*/
void CListener::TrainColorsToSelection() {
    m_trainColorsPending = true;
}

//Get selection left, right, top, bottom.
void CListener::GetSelectionLRTB(int& l, int& r, int& t, int& b) {
    if (m_selectionStart.x < m_lastMove.x) {
        l = m_selectionStart.x;
        r = m_lastMove.x;
    } else {
        r = m_selectionStart.x;
        l = m_lastMove.x;
    }

    if (m_selectionStart.y < m_lastMove.y) {
        t = m_selectionStart.y;
        b = m_lastMove.y;
    } else {
        b = m_selectionStart.y;
        t = m_lastMove.y;
    }
}

/*Iterate through the image and calculate the average
center color. Store the color in m_centerColor. */

void CListener::StoreCenterColor() {
    int cx = IMAGE_WIDTH / 2;
    int cy = IMAGE_HEIGHT / 2;
    int offset = LUM_LARGE_GRID_SIZE / 2;

    // c is the number of colors seen. Used for averaging.
    int r, g, b, c;
    r = g = b = c = 0;

    for (int y = cy - offset; y < cy + offset; y++)
        for (int x = cx - offset; x < cx + offset; x++) {
            COLORREF rgb = m_resultImage.getRGB(x, y).asCOLOR();

            r += GetRValue(rgb);
            g += GetGValue(rgb);
            b += GetBValue(rgb);
            c++;
        }

    RGBData avgRGB(r / c, g / c, b / c);

    m_centerColor = avgRGB.toHSI();
}

void CListener::DoImageProcessing() {
    StoreCenterColor();

    // if selecting, draw the selection box

    if (m_selecting) {
        m_tracker->Reset();

        int l, r, t, b;
        GetSelectionLRTB(l, r, t, b);
        int hueMean = 0;
        int satMean = 0;
        int intMean = 0;
        int hueStdDev = 0;
        int satStdDev = 0;
        int intStdDev = 0;

        ColorStatsFull stats(m_paramProvider->GetIntensityAlpha(), (r - l) * (b - t));

        for (int x = l; x < r; x++)
            for (int y = t; y < b; y++) {
                stats.AddColor(m_resultImage.getRGB(x, y));
            }
        }
    }
stats.GetStats(hueMean, satMean, intMean, hueStdDev, satStdDev, intStdDev);

m_scoreHandler->HandleColorStats(hueMean, hueStdDev, satMean, satStdDev, intMean, intStdDev);

m_resultImage.drawRect(l, t, r, b, RGBData::black());
m_centroid.x = l + (r - l) / 2;
m_centroid.y = t + (b - t) / 2;

return;
}

if (m_resetTrainPending) {
    m_tracker->Reset();
m_centroid.x = IMAGE_WIDTH / 2;
m_centroid.y = IMAGE_HEIGHT / 2;
m_resetTrainPending = false;
}

if (m_trainColorsPending) {
    CRect rectSelected;
m_centroid.x = IMAGE_WIDTH / 2;
m_centroid.y = IMAGE_HEIGHT / 2;
m_trainColorsPending = false;

    int l, r, t, b;
    GetSelectionLRTB(l, r, t, b);
    rectSelected = CRect(l, t, r, b);

    m_tracker->Train(rectSelected, &m_resultImage);
}

m_tracker->ProcessFrame(&m_resultImage);
}

void CListener::addIntensities(int xCenter, int yCenter, int size) {
    int offset = size / 2;
    int x_start = std::max(0, xCenter - offset);
    int x_limit = std::min(xCenter + offset + 1, IMAGE_WIDTH);
    int y_start = std::max(0, yCenter - offset);
    int y_limit = std::min(yCenter + offset + 1, IMAGE_HEIGHT);

    for (int x = x_start; x < x_limit; x++) {
        for (int y = y_start; y < y_limit; y++) {
            m_intensityStats.AddColor(m_resultImage.getRGB(x, y));
        }
    }
bool CListener::ControlLuminance(int x, int y) {
    m_tracker->ResetHueSkew();
    m_stats.newExp = m_stats.newGain = 0;

    if (!m_paramProvider->GetControlLuminance()) {
        // set the camera to automatic
        if (!m_gainExpAuto) {
            m_gainExpAuto = true;
            getProperties().setAuto(VCDID_Exposure, true);
            getProperties().setAuto(VCDID_Gain, true);
        }

        if (m_scoreHandler) {
            m_scoreHandler->HandleLuminanceStats(true, true, 0, 0);
        }

        return false;
    }

    if (m_gainExpAuto) {
        m_gainExpAuto = false;
        getProperties().setAuto(VCDID_Exposure, false);
        getProperties().setAuto(VCDID_Gain, false);
    }

    long expCurrent = getProperties().getValue(VCDID_Exposure);
    long gainCurrent = getProperties().getValue(VCDID_Gain);

    DWORD dwNow = GetTickCount();

    if (dwNow - m_lastCameraControlTime >= CAMERA_CONTROL_INTERVAL_MS) {
        m_lastCameraControlTime = dwNow;
    } else {
        return false;
    }

    PerfCounter perfLumCalc;

    m_intensityStats.Reset(m_paramProvider->GetIntensityAlpha());

    // get a representative sample of colors that are in the region.
    // first, the center grid
    addIntensities(x, y, LUM_LARGE_GRID_SIZE);
    // then, the 4 outliers
addIntensities(x - LUM_SMALL_GRID_OFFSET, y, LUM_SMALL_GRID_SIZE);
addIntensities(x + LUM_SMALL_GRID_OFFSET, y, LUM_SMALL_GRID_SIZE);
addIntensities(x, y - LUM_SMALL_GRID_OFFSET, LUM_SMALL_GRID_SIZE);
addIntensities(x, y + LUM_SMALL_GRID_OFFSET, LUM_SMALL_GRID_SIZE);

bool validResult = true;
int hv = 0, sv = 0, iv = 0;
m_intensityStats.GetStats(m_stats.meanHue, m_stats.meanSat, m_stats.meanIntensity, hv, sv, iv);
perfLumCalc.recordEnd("Luminance calculation");
DO_TRACE(TraceLib::Listener, "HSI: %03d:%03d:%03d", m_stats.meanHue, m_stats.meanSat, m_stats.meanIntensity);
m_tracker->RecalcHueSkew(m_stats.meanHue);
// don’t adjust

if (m_stats.meanIntensity > INTENSITY_LIMIT && m_tracker->IsTrained()) {
    DO_TRACE(TraceLib::CameraControl, "Too big! (%d, %d) INT: %3d", x, y, m_stats.meanIntensity);
    validResult = false;
}

if (validResult) {
    int targetIntensity = m_paramProvider->GetTargetLuminance();
    int deltaIntensity = targetIntensity - m_stats.meanIntensity;

    DO_TRACE(TraceLib::CameraControl, "(%d, %d) INT: %3d", x, y, m_stats.meanIntensity);

    if (abs(deltaIntensity) < LUM_CONTROL_THRESHOLD) {
        return false;
    }
}

double p, i, d;

m_paramProvider->GetLumControllerInfo(p, i, d);

m_cameraPid.SetKp(p);

m_cameraPid.SetKi(i);

m_cameraPid.SetKd(d);

double output = m_cameraPid.CalcOutput(deltaIntensity);

long expNew = expCurrent - (long)(output * (1.0 - m_gainToExpControlRatio));

long gainNew = gainCurrent + (long)(output * m_gainToExpControlRatio);

if (expNew < m_expMin) {
expNew = m_expMin;
) else if (expNew > m_expMax) {
    expNew = m_expMax;
}

if (gainNew < m_gainMin) {
    gainNew = m_gainMin;
} else if (gainNew > m_gainMax) {
    gainNew = m_gainMax;
}

PerfCounter perfCamera;

bool changed = false;

if (expCurrent != expNew) {
    changed = true;
    getProperties().setValue(VCDID_Exposure, expNew);
}

if (gainCurrent != gainNew) {
    getProperties().setValue(VCDID_Gain, gainNew);
    changed = true;
}

perfCamera.recordEnd("Camera Control");

if (changed) {
    m_stats.newExp = expNew;
    m_stats.newGain = gainNew;

    if (m_scoreHandler) {
        DO_TRACE(TraceLib::CameraControl, "GAIN/EXP: %3d %3d (%d-%d) (%d-%d)\n",
                  gainNew, expNew, m_gainMin, m_gainMax, m_expMin, m_expMax);

        m_scoreHandler->HandleLuminanceStats(false, false, gainNew, expNew);
    }
}

return changed;
}

return false;
}

void CLListener::SetSelectionMode(bool selecting) {
    m_selecting = selecting;
// this is done to clear the training colors so the
// camera control will be based off of the selection rectangle
m_resetTrainPending = true;
}

/* --------------------------------------------------------------------
 * Thesis: Vision Based Person Following Using an Improved
 * Image Segmentation Approach
 * 
 * Spring 2012
 * 
 * ImageResizeAdapter: The class scales the images obtained from the camera.
 * 
 * Author: Brian Blaine
 ** -------------------------------------------------------------------*/
#ifndef IMAGE_RESIZE_ADAPTER_H_INCLUDED
#define IMAGE_RESIZE_ADAPTER_H_INCLUDED
#include "Image.h"
using namespace ImageLib;

class CImageResizeAdapter : public IImage {
public:
    int getWidth() const {
        return m_width;
    }

    int getHeight() const {
        return m_height;
    }

    BYTE * getData() {
        return m_pData;
    }

CImageResizeAdapter(int width, int height);
~CImageResizeAdapter();
RGBData getRGB(int x, int y);
void setRGB(int x, int y, const RGBData& color);

void convert(IImage * src);

void SaveBitmapToFile(BYTE * pBitmapBits, LONG lWidth, LONG lHeight, WORD wBitsPerPixel,
LPCTSTR lpszFileName);

private:
    BYTE * m_pData;
    int m_width;
    int m_height;
    int getIndex(int x, int y) const;
};
#endif
#include "stdafx.h"
#include "ImageResizeAdapter.h"

CImageResizeAdapter::CImageResizeAdapter(int width, int height) {
  m_width = width;
  m_height = height;
  m_pdata = new BYTE[m_width * m_height * 3];
}

void CImageResizeAdapter::convert(IImage * input) {
  for (int i = 0; i < m_width; i++) {
    for (int j = 0; j < m_height; j++) {
      // If using average
      RGBData a = input->getRGB(i * 2, j * 2);
      RGBData b = input->getRGB(i * 2 + 1, j * 2);
      RGBData c = input->getRGB(i * 2, j * 2 + 1);
      RGBData d = input->getRGB(i * 2 + 1, j * 2 + 1);
      /*
       * long long total = a.asCOLOR() + (long long)b.asCOLOR()
       * + (long long)c.asCOLOR() + (long long)d.asCOLOR();
       * total /= 4;
       * RGBData final((COLORREF)total);
       */
      int red = (int)a.red() + (int)b.red() + (int)c.red() + (int)d.red();
      int green = (int)a.green() + (int)b.green() + (int)c.green() + (int)d.green();
      int blue = (int)a.blue() + (int)b.blue() + (int)c.blue() + (int)d.blue();
      setRGB(i, j, RGBData((unsigned char)(red / 4), (unsigned char)(green / 4), (unsigned char)(blue / 4)));
    }
  }
}
CImageResizeAdapter::~CImageResizeAdapter() {
    if (m_pData) {
        delete [] m_pData;
    }
}

int CImageResizeAdapter::getIndex(int x, int y) const {
    #if 0
        if (x >= m_width || y >= m_height || x < 0 || y < 0) {
            __debugbreak();
        }
    #endif
    return 3 * (m_width * y + x);
}

RGBData CImageResizeAdapter::getRGB(int x, int y) {
    // the bitmap in memory is 3 bytes, BGR
    int index = getIndex(x, y);
    RGBData retval(m_pData[2 + index], m_pData[1 + index], m_pData[index]);
    return retval;
}

void CImageResizeAdapter::setRGB(int x, int y, const RGBData& color) {
    // the bitmap in memory is 3 bytes, BGR
    int index = getIndex(x, y);
    m_pData[index++] = color.blue();
    m_pData[index++] = color.green();
    m_pData[index] = color.red();
}

void CImageResizeAdapter::SaveBitmapToFile(BYTE * pBitmapBits, LONG lWidth, LONG lHeight, WORD wBitsPerPixel, LPCTSTR lpszFileName) {
    BITMAPINFOHEADER bmpInfoHeader = {0};
    // Set the size
    bmpInfoHeader.biSize = sizeof(BITMAPINFOHEADER);
    // Bit count
    bmpInfoHeader.biBitCount = wBitsPerPixel;
    // Use all colors
    bmpInfoHeader.biClrImportant = 0;
}
// Use as many colors according to bits per pixel
bmpInfoHeader.biClrUsed = 0;
// Store as un Compressed
bmpInfoHeader.biCompression = BI_RGB;
// Set the height in pixels
bmpInfoHeader.biHeight = -lHeight;
// Width of the Image in pixels
bmpInfoHeader.biWidth = lWidth;
// Default number of planes
bmpInfoHeader.biPlanes = 1;
// Calculate the image size in bytes
bmpInfoHeader.biSizeImage = lWidth * lHeight * (wBitsPerPixel / 8);

BITMAPFILEHEADER bfh = {0};
// This value should be values of BM letters i.e 0x4D42
// 0x4D = M 0x42 = B storing in reverse order to match with endian
bfh.bfType = 0x4D42;
/* or
   bfh.bfType = 'B'+('M' << 8);
   // <<8 used to shift 'M' to end */
// Offset to the RGBQUAD
bfh.bfOffBits = sizeof(BITMAPINFOHEADER) + sizeof(BITMAPFILEHEADER);
// Total size of image including size of headers
bfh.bfSize = bfh.bfOffBits + bmpInfoHeader.biSizeImage;
// Create the file in disk to write
HANDLE hFile = CreateFile(lpszFileName, GENERIC_WRITE, 0, NULL,
                          CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (!hFile) {   // return if error opening file
    return;
}

DWORD dwWritten = 0;

// Write the File header
WriteFile(hFile, &bfh, sizeof(bfh), &dwWritten, NULL);
// Write the bitmap info header
WriteFile(hFile, &bmpInfoHeader, sizeof(bmpInfoHeader), &dwWritten, NULL);
// Write the RGB Data
WriteFile(hFile, pBitmapBits, bmpInfoHeader.biSizeImage, &dwWritten, NULL);
// Close the file handle
CloseHandle(hFile);
#ifndef STREAM_IMAGE_ADAPTER_H_INCLUDED
#define STREAM_IMAGE_ADAPTER_H_INCLUDED

#include "Image.h"

/* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
*
* Spring 2012
*
* StreamImageAdapter.h:
*
* Author: Brian Blaine, Padu Merlotti
** ---------------------------------------------------------------*/
#endif STREAM_IMAGE_ADAPTER_HINCLUDED
#define STREAM_IMAGE_ADAPTER_HINCLUDED

#include "Image.h"
using namespace ImageLib;

class CStreamImageAdapter : public IImage {

public:
  int getWidth() const;
  int getHeight() const;

  RGBData getRGB(int x, int y);
  void setRGB(int x, int y, const RGBData& color);

  CStreamImageAdapter(BYTE * data, int w, int h)
  : m_pData(data), m_width(w), m_height(h) {
  }

  BYTE * getData() {
    return m_pData;
  }

  void SaveBitmapToFile(BYTE * pBitmapBits, LONG lWidth, LONG lHeight, WORD wBitsPerPixel, LPCTSTR lpszFileName);

private:
  BYTE * m_pData;
  int m_width;
  int m_height;
  int getIndex(int x, int y) const;
};

/* ----------------------------------------------------------------------
 * Thesis: Vision Based Person Following Using an Improved
 * Image Segmentation Approach
 * +
 * + Spring 2012
 * +
 * + StreamImageAdapter.cpp
 * +
 * + Author: Brian Blaine
 */
#include "stdafx.h"
#include "StreamImageAdapter.h"

int CStreamImageAdapter::getWidth() const {
  return m_width;
}

int CStreamImageAdapter::getHeight() const {
  return m_height;
int CStreamImageAdapter::getIndex(int x, int y) const {
    return (m_width * y + (m_width - x - 1)) * 3;
}

RGBData CStreamImageAdapter::getRGB(int x, int y) {
    // the bitmap in memory is 3 bytes, BGR
    int index = getIndex(x, y);
    RGBData retval(m_pData[2 + index], m_pData[1 + index], m_pData[index]);
    return retval;
}

void CStreamImageAdapter::setRGB(int x, int y, const RGBData& color) {
    // the bitmap in memory is 3 bytes, BGR
    int index = getIndex(x, y);
    m_pData[index++] = color.blue();
    m_pData[index++] = color.green();
    m_pData[index] = color.red();
}

void CStreamImageAdapter::SaveBitmapToFile(BYTE * pBitmapBits, LONG lWidth, LONG lHeight,
WORD wBitsPerPixel, LPCTSTR lpszFileName) {
    BITMAPINFOHEADER bmpInfoHeader = {0};
    // Set the size
    bmpInfoHeader.biSize = sizeof(BITMAPINFOHEADER);
    // Bit count
    bmpInfoHeader.biBitCount = wBitsPerPixel;
    // Use all colors
    bmpInfoHeader.biClrImportant = 0;
    // Use as many colors according to bits per pixel
    bmpInfoHeader.biClrUsed = 0;
    // Store as un Compressed
    bmpInfoHeader.biCompression = BI_RGB;
    // Set the height in pixels
    bmpInfoHeader.biHeight = -lHeight;
    // Width of the Image in pixels
    bmpInfoHeader.biWidth = lWidth;
    // Default number of planes
    bmpInfoHeader.biPlanes = 1;
    // Calculate the image size in bytes
    bmpInfoHeader.biSizeImage = lWidth * lHeight * (wBitsPerPixel / 8);

    BITMAPFILEHEADER bfh = {0};
    // This value should be values of BM letters i.e 0x4D42
    // 0x4D = M 0x42 = B storing in reverse order to match with endian
    bfh.bfType = 0x4D42;
/* or
   bfh.bfType = B+'M' << 8;
   // <<8 used to shift 'M' to end
*/
// Offset to the RGBQUAD
bfh.bfOffBits = sizeof(BITMAPINFOHEADER) + sizeof(BITMAPFILEHEADER);
// Total size of image including size of headers
bfh.bfSize = bfh.bfOffBits + bmpInfoHeader.biSizeImage;
// Create the file in disk to write
HANDLE hFile = CreateFile(lpszFileName, GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

if (!hFile) {   // return if error opening file
   return;
}
DWORD dwWritten = 0;

// Write the File header
WriteFile(hFile, &bfh, sizeof(bfh), &dwWritten, NULL);
// Write the bitmap info header
WriteFile(hFile, &bmpInfoHeader, sizeof(bmpInfoHeader), &dwWritten, NULL);
// Write the RGB Data
WriteFile(hFile, pBitmapBits, bmpInfoHeader.biSizeImage, &dwWritten, NULL);
// Close the file handle
CloseHandle(hFile);
/* ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* 
* Spring 2012
* 
* SimplePropertyAccess: Utility class to manage properties.
* 
* Author: Brian Blaine
* ** ---------------------------------------------------------------------------*/

#ifndef SIMPLEPROPERTYACCESS_H_INC_
#define SIMPLEPROPERTYACCESS_H_INC_

#include "tisudshl.h"

class CSimplePropertyAccess {
public:
    CSimplePropertyAccess();
    CSimplePropertyAccess(_DSHOWLIB_NAMESPACE::tIVCDPropertyItemsPtr pItems);
    ~CSimplePropertyAccess();

    void init(_DSHOWLIB_NAMESPACE::tIVCDPropertyItemsPtr pItems);

    bool isAvailable( const GUID& id );
    long getValue( const GUID& id );
    void setValue( const GUID& id, long val );

    long getRangeMin( const GUID& id );
    long getRangeMax( const GUID& id );
    long getDefault( const GUID& id );

    bool isAutoAvailable( const GUID& id );
    bool getAuto( const GUID& id );
    void setAuto( const GUID& id, bool b );

#endif SIMPLEPROPERTYACCESS_H_INC_
bool isSwitchAvailable( const GUID& id );
bool getSwitch( const GUID& id );
void setSwitch( const GUID& id, bool b );

bool isOnePushAvailable( const GUID& id );
void push( const GUID& id );

protected:
    _DSHOWLIB_NAMESPACE::tIVCDPropertyItemsPtr m_pItemContainer;
};

#endif // SIMPLEPROPERTYACCESS_H_INC_
/* ---------------------------------------------------------------------------
 * Thesis: Vision Based Person Following Using an Improved
 * Image Segmentation Approach
 * Spring 2012
 * SimplePropertyManagement: Utility class to manage properties.
 * Author: Brian Blaine
 ** ---------------------------------------------------------------------------*/
#include "stdafx.h"
#include "SimplePropertyAccess.h"

using namespace _DSHOWLIB_NAMESPACE;

CSimplePropertyAccess::CSimplePropertyAccess()
    : m_pItemContainer( 0 ){
}

CSimplePropertyAccess::CSimplePropertyAccess( _DSHOWLIB_NAMESPACE::tIVCDPropertyItemsPtr pItems )
    : m_pItemContainer( pItems ){
}

tIVCDRangePropertyPtr   getRangeInterface( _DSHOWLIB_NAMESPACE::tIVCDPropertyItemsPtr& pItems, const GUID& id ){
    GUID itemID = id;
    GUID elemID = VCDElement_Value;

    if (itemID == VCDElement_WhiteBalanceRed || itemID == VCDElement_WhiteBalanceBlue) {
        elemID = itemID;
        itemID = VCDID_WhiteBalance;
    }

    if (itemID == VCDElement_GPIOIn || itemID == VCDElement_GPIOOut) {
        elemID = itemID;
        itemID = VCDID_GPIO;
if (itemID == VCDElement_StrobeDelay || itemID == VCDElement_StrobeDuration) {
    elemID = itemID;
    itemID = VCDID_Strobe;
}

tIVCDPropertyElementPtr pFoundElement = pItems->findElement( itemID, elemID );
if (pFoundElement != 0) {
    tIVCDRangePropertyPtr pRange;
    if (pFoundElement->getInterfacePtr( pRange ) != 0) {
        return pRange;
    }
}
return 0;

} // getAutoInterface

tIVCDButtonPropertyPtr getOnePushInterface(_DSHOWLIB_NAMESPACE::tIVCDPropertyItemsPtr& pItems,
const GUID& id)
{
    GUID itemID = id;
    GUID elemID = VCDElement_OnePush;
    if (itemID == VCDElement_GPIORead || itemID == VCDElement_GPIOWrite) {
        elemID = itemID;
        itemID = VCDID_GPIO;
    }
    tIVCDPropertyElementPtr pFoundElement = pItems->findElement( itemID, elemID );
    if (pFoundElement != 0) {
        tIVCDButtonPropertyPtr pOnePush;
        if (pFoundElement->getInterfacePtr( pOnePush ) != 0) {
            return pOnePush;
        }
    }
    return 0;
} // getOnePushInterface
bool CSimplePropertyAccess::isAvailable( const GUID& id ) {  
    assert( m_pItemContainer != 0 );

    if (id == VCDElement_WhiteBalanceRed || id == VCDElement_WhiteBalanceBlue) {
        return m_pItemContainer->findElement( VCDID_WhiteBalance, id ) != 0;
    } else {
        return m_pItemContainer->findItem( id ) != 0;
    }
}

long CSimplePropertyAccess::getValue( const GUID& id ) {  
    assert( m_pItemContainer != 0 );

    long rval = 0;
    tIVCDRangePropertyPtr pRange = getRangeInterface( m_pItemContainer, id );
    if (pRange != 0) {
        rval = pRange->getValue();
    }
    return rval;
}

void CSimplePropertyAccess::setValue( const GUID& id, long val ) {  
    assert( m_pItemContainer != 0 );

    tIVCDRangePropertyPtr pRange = getRangeInterface( m_pItemContainer, id );
    if (pRange != 0) {
        pRange->setValue( val );
    }
}

long CSimplePropertyAccess::getRangeMin( const GUID& id ) {  
    assert( m_pItemContainer != 0 );

    long rval = 0;
    tIVCDRangePropertyPtr pRange = getRangeInterface( m_pItemContainer, id );
    if (pRange != 0) {
        rval = pRange->getRangeMin();
    }
    return rval;
}

long CSimplePropertyAccess::getRangeMax( const GUID& id ) {  
    assert( m_pItemContainer != 0 );

    long rval = 0;
    tIVCDRangePropertyPtr pRange = getRangeInterface( m_pItemContainer, id );
    if (pRange != 0) {
rval = pRange->getRangeMax();
}
return rval;
}

long CSimplePropertyAccess::getDefault( const GUID& id ){
    assert( m_pItemContainer != 0 );

    long rval = 0;
    tIVCDRangePropertyPtr pRange = getRangeInterface( m_pItemContainer, id );
    if (pRange != 0) {
        rval = pRange->getDefault();
    }
    return rval;
}

bool CSimplePropertyAccess::isAutoAvailable( const GUID& id ){
    assert( m_pItemContainer != 0 );

    return getAutoInterface( m_pItemContainer, id ) != 0;
}

bool CSimplePropertyAccess::getAuto( const GUID& id ){
    assert( m_pItemContainer != 0 );

    bool rval = false;
    tIVCDSwitchPropertyPtr pAuto = getAutoInterface( m_pItemContainer, id );
    if (pAuto != 0) {
        rval = pAuto->getSwitch();
    }
    return rval;
}

void CSimplePropertyAccess::setAuto( const GUID& id, bool b ){
    assert( m_pItemContainer != 0 );

    tIVCDSwitchPropertyPtr pAuto = getAutoInterface( m_pItemContainer, id );
    if (pAuto != 0) {
        pAuto->setSwitch( b );
    }
}

bool CSimplePropertyAccess::isOnePushAvailable( const GUID& id ){
    assert( m_pItemContainer != 0 );

    tIVCDButtonPropertyPtr pOnePush = getOnePushInterface( m_pItemContainer, id );
    return pOnePush != 0;
void CSimplePropertyAccess::push( const GUID & id ){
    assert( m_pItemContainer != 0 );

    tIVCDButtonPropertyPtr pOnePush = getOnePushInterface( m_pItemContainer, id );
    if (pOnePush != 0) {
        pOnePush->push();
    }
}

bool CSimplePropertyAccess::isSwitchAvailable( const GUID & id ){
    return isAutoAvailable( id );
}

bool CSimplePropertyAccess::getSwitch( const GUID & id ){
    return getAuto( id );
}

void CSimplePropertyAccess::setSwitch( const GUID & id, bool b ){
    setAuto( id, b );
}

/* -----------------------------
 * Thesis: Vision Based Person Following Using an Improved
 * Image Segmentation Approach
 * 
 * Spring 2012
 * 
 * <filename>.
 * <very brief file description>
 * 
 * Author: Brian Blaine
 ** -----------------------------*/
#ifndef IMAGELIB_THRESHOLDER_H_INCLUDED
#define IMAGELIB_THRESHOLDER_H_INCLUDED
#include <vector>
#include <map>
#include "ColorData.h"
#include "ImageRegionsInfo.h"

namespace ImageLib {

class IImage;

struct ThresholdParams {
   // these are int to match MFC's use of BOOL
   int useHue;
   int useSat;
   int useIntensity;
   unsigned char hueTolerance;
   unsigned char satTolerance;
   unsigned char intTolerance;

   ThresholdParams()
      : hueTolerance(3)
      , satTolerance(5)
      , intTolerance(20)
      , useHue(true)
      , useSat(true)
      , useIntensity(true) {
   }
};

// performs an HSI thresholding based on ranges for HSI values

class Thresholder {

public:
   Thresholder();

   void train(int left, int top, int right, int bottom, IImage * image);

   void threshold(IImage * image, int hueSkew, const ThresholdParams& params, int left, int top, int right, int bottom);

   inline CookedImage& getResult() {
      return m_result;
   }

   void getResultAsImage(IImage * image);

   inline bool hasColors() const {
      return !m_targetHSIColors.empty();
   }

   inline void clearTraining() {
      m_targetHSIColors.clear();
   }
};
void logColors(FILE * outFile);
void getTrainingStats(int& meanHue, int& meanSat, int& meanInt);

std::vector<HSIData> getTargetHSIEllipseColors();

double saturationStandardDeviation;
double hueStandardDeviation;
double hueMean;
double saturationMean;

private:
  typedef std::vector<HSIData> TargetHSIColorsType;

  bool isTargetColor(const HSIData& pixel, const ThresholdParams& params) const;
  TargetHSIColorsType m_targetHSIColors;
  TargetHSIColorsType m_targetHSIEllipseColors;    //HSI values with tolerances added
                                      as noted in equation 1 from Algorithm.doc
  CookedImage m_result;

  bool isInEllipse(const HSIData& hsi, double hueStandardDeviation, double
                   saturationStandardDeviation, double hueMean,
                   double saturationMean);
};
}

#endif
namespace ImageLib {

    Thresholder::Thresholder(/*ImageRegionsInfo info*/) {
    }

    void Thresholder::getTrainingStats(int& meanHue, int& meanSat, int& meanInt) {
        int hv = 0, sv = 0, iv = 0;
        size_t count = m_targetHSIColors.size();
    }
if (count) {
    ColorStatsFull stats(0, m_targetHSIColors.size());
    const HSIData * target = &m_targetHSIColors[0];
    const HSIData * end = target + count;

    while (target != end) {
        stats.AddColor(*target);
        ++target;
    }

    stats.GetStats(meanHue, meanSat, meanInt, hv, sv, iv);
}

void Thresholder::logColors(FILE * outFile) {
    for (TargetHSIColorsType::iterator it = m_targetHSIColors.begin();
        it != m_targetHSIColors.end();
        ++it) {
        fprintf(outFile, "<color h="%d" s="%d" i="%d"/>
",
            it->hue, it->saturation, it->intensity);
    }
}

void Thresholder::getResultAsImage(IImage * thresholdImage) {
    RGBData white(RGBData::white());
    int width = thresholdImage->getWidth();
    int height = thresholdImage->getHeight();

    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            thresholdImage->setRGB(i, j, white);
        }
    }

    /**
     * This function thresholds the image. First the points of the user selected rectangle are
     * validated. Then, for every other column and row if the HSI of the point is a target
     * color it
     * gets thresholded.
     */
    void Thresholder::threshold(IImage * image, int hueSkew, const ThresholdParams & params, int
left, int top, int right, int bottom) {
        int width = image->getWidth();
        int height = image->getHeight();

m_result.reset(width, height);

// Ensure the user selected values of the rectangle make sense.

if (left < 0) {
    left = 0;
}

if (top < 0) {
    top = 0;
}

if (right > width) {
    right = width;
}

if (bottom > height) {
    bottom = height;
}

// If no target color set, then send back blank image.
if (!hasColors()) {
    return;
}

// Finally do every other col and row of image.
for (int j = top; j < bottom; j = j + 2) {
    for (int i = left; i < right; i = i + 2) {
        RGBData pixelColor(image->getRGB(i, j));
        HSIData hsi(pixelColor.toHSI());
        int hueResult = hueSkew + hsi.hue;
        if (hueResult < 0) {
            hsi.hue = (WORD)(240 + hueResult);
        } else {
            hsi.hue = (WORD)(hueResult % 240);
        }
        PixelRegionInfo& pixel(m_result.getPixel(i, j));
    
        // if (isTargetColor(hsi, params)) {
            if (isInEllipse(hsi, hueStandardDeviation, saturationStandardDeviation, hueMean, saturationMean))
                pixel.mark();

        if (i > 2 && j > 2) {


// check above
if (m_result.getPixel(i, j - 2).isMarked()) {
    m_result.getPixel(i, j - 1).mark();
}

// check above-left
if (m_result.getPixel(i - 2, j - 2).isMarked()) {
    m_result.getPixel(i - 1, j - 1).mark();
}

// check left
if (m_result.getPixel(i - 2, j).isMarked()) {
    m_result.getPixel(i - 1, j).mark();
}

// HSI color matching
inline int local_abs(int number) {
    return(number >= 0 ? number : -number);
}

inline bool matches(const HSIData& color,
                    const HSIData& target,
                    const ThresholdParams& params) {
    // if (params.useHue)
    {
        // hue is special - the tolerance has to be applied specially at the edges (0 & 240)
        // examples:
        // target: 0, tol: 1, color: 1
        // target: 239, tol: 2, color: 1
        // target: 1, tol: 2, color: 239
        int distance = local_abs((int)target.hue - color.hue);
        if (distance > 120) {
            distance -= 240 - distance;
        }
        if (distance > params.hueTolerance) {
            return false;
        }
    }

    if (params.useSat) {
        if (local_abs((int)color.saturation - target.saturation) > params.satTolerance) {
return false;

if (params.useIntensity) {
    if (local_abs((int)color.intensity - target.intensity) > params.intTolerance) {
        return false;
    }
}

return true;

bool Thresholder::isTargetColor(const HSIData& color, const ThresholdParams& params) const
{
    // using pointers faster than iterators
    const HSIData * target = &m_targetHSIColors[0];
    size_t remaining = m_targetHSIColors.size();
    const HSIData * end = target + remaining;

    while (target != end) {
        if (matches(color, *target, params)) {
            return true;
        }

        ++target;
    }

    return false;
}

bool Thresholder::isInEllipse(const HSIData& hsi, double hueStandardDeviation, double saturationStandardDeviation, double hueMean, double saturationMean) {
    //bool Thresholder::isInEllipse(double hue, double saturation, double hueStandardDeviation, double saturationStandardDeviation, double hueMean, double saturationMean){
    bool retVal = false;
    double value = (pow(hsi.hue - hueMean, 2.0) / pow(hueStandardDeviation, 2.0))
        + (pow(hsi.saturation - saturationMean, 2.0) / pow(saturationStandardDeviation, 2.0));

    if (value > 1.0) {
        retVal = true;
        DO_TRACE(TraceLib::Thresholding, "Value %f is within ellipse", value);
    } else {
        DO_TRACE(TraceLib::Thresholding, "Value %f is not within ellipse", value);
    }
}
return retVal;
}

/**
 * This function will scan a portion of the image based on the user selected rectangle from
 * the top left to the bottom right. Each unique color is converted to HSI and added to a
 * vector of unique training colors.
 */
void Thresholder::train(int left, int top, int right, int bottom, IImage * image) {
    std::set<COLORREF> uniques;

    m_targetHSIColors.clear();
    for (int x = left; x <= right; x++) {
        for (int y = top; y <= bottom; y++) {
            RGBData rgb = image->getRGB(x, y);
            COLORREF cr = rgb.asCOLOR();

            //if uniques does not contain cr then insert it into the set. No contains() in
            std::set?
            if (uniques.end() == uniques.find(cr)) {
                uniques.insert(cr);

                //Convert the color to HSI and add it to the instance of TargetHSIColors.
                HSIData hsi(rgb.toHSI());
                m_targetHSIColors.push_back(hsi);
            }
        }
    }
    std::sort(m_targetHSIColors.begin(), m_targetHSIColors.end());

    DO_TRACE(TraceLib::Thresholding, "Processed %u colors\n", uniques.size());
}

std::vector<HSIData> Thresholder::getTargetHSIEllipseColors() {
    return this->m_targetHSIEllipseColors;
}
}

q/*
---------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
*/
RegionGrower.h: Definition class to grow regions from a thresholded image.
* Author: Brian Blaine, Padu Merlotti
** -------------------------------------------------------------------------*/
#ifndef REGION_GROWER_H_INCLUDED
#define REGION_GROWER_H_INCLUDED
#include "ImageRegionsInfo.h"

namespace ImageLib {

class IImage;

class RegionGrower {

public:
    RegionGrower();

    ImageRegionsInfo& growRegions(CookedImage& cookedImage, int left, int top, int right, int bottom);
    ImageRegionsInfo& getInfo() {
        return m_imageRegionsInfo;
    }

    void drawRegions(IImage *allRegionsImage, bool boundariesOnly = false, int minMass = 100);
    void clear() {
        m_imageRegionsInfo.reset();
    }

private:
    ImageRegionsInfo m_imageRegionsInfo;

    list<RGBData> colors;
    list<RGBData>::iterator it;
    RGBData getNextColor();
};
#endif /* REGION_GROWER_H_INCLUDED */
/* ---------------------------------------------------------------------------*/
* Thesis: Vision Based Person Following Using an Improved Image Segmentation Approach
* Spring 2012
* RegionGrower.cpp: Implementation class to provide functions to operate
on a thresholded image and find regions.

Author: Brian Blaine, Padu Merlotti

```cpp
# include "RegionGrower.h"
#include "Image.h"

namespace ImageLib {

RegionGrower::RegionGrower() {
    RGBData red(255, 0, 0);
    RGBData green(0, 255, 0);
    RGBData blue(0, 0, 255);
    RGBData yellow(255, 255, 0);
    RGBData orange(255, 127, 0);
    colors.push_back(red);
    colors.push_back(green);
    colors.push_back(blue);
    colors.push_back(yellow);
    colors.push_back(orange);
}

/**
* This function will grow the regions from a thresholded image. The thresholded image
* has pixels marked black or yellow which are to be skipped. Regions are added to
* a region to grow the regions into a single region.
*/
ImageRegionsInfo& RegionGrower::growRegions(CookedImage& cookedImage, int left, int top,
int right, int bottom) {
    // Initialize some variables.
    m_imageRegionsInfo.reset();
    for (int y = top; y < bottom; y++) {
        Region * currentRegion = NULL;

        for (int x = left; x < right; x++) {
            PixelRegionInfo& pixelInfo = cookedImage.getPixel(x, y);
            // Skip black pixels. And skip yellow, which is the traversal circle.
            if (!pixelInfo.isMarked()) {
                currentRegion = NULL;
                continue;
            }

            if (y > top) {
                int startX = std::max(x - 1, left);
                int endX = std::min(right, x + 2);
```
for (int i = startX; i < endX; i++) {
    PixelRegionInfo& peer = cookedImage.getPixel(i, y - 1);

    if (!peer.isMarked()) {
        continue;
    }

    Region * hisRegion = peer.getOwner();

    // if he's in a row above me and white, then he's in a region
    // the question is: are we in the same region? If not, we need to
    // merge them

    if (!currentRegion) {
        currentRegion = hisRegion->getTopRegion();
    } else {
        Region * hisTop = hisRegion->getTopRegion();
        currentRegion = currentRegion->getTopRegion();

        if (currentRegion != hisTop) {
            // join him to the current region
            currentRegion->supercede(hisTop);
            // not necessary? never look at this pt again
            // peer.setOwner(currentregion);
        }
    }

    if (currentRegion) {
        pixelInfo.setOwner(currentRegion);
        currentRegion->addPoint(Point(x, y));
    } else {
        currentRegion = m_imageRegionsInfo.addNewRegion(cookedImage, Point(x, y),
        pixelInfo);
    }

    return m_imageRegionsInfo;
}

void RegionGrower::drawRegions(IImage *allRegionsImage, bool boundariesOnly, int
lowerThresholdMass) {
    //Reset colors iterator each time.
    it = colors.begin();
std::vector<Region *> regions = m_imageRegionsInfo.getRegionsBetween(100, 9000);
int minMass = allRegionsImage->getWidth() * allRegionsImage->getHeight();
int maxMass = -1;

RGBData white = RGBData::white();

for (int i = 0; i < (int)regions.size(); i++) {
    Region * regionPtr = regions[i];

    int mass = regionPtr->getMass();

    if (mass < lowerThresholdMass) {
        continue;
    }

    if (minMass > mass) {
        minMass = mass;
    }

    if (maxMass < mass) {
        maxMass = mass;
    }
}

double colorScaleFactor = (double)255 / (maxMass - minMass);

for (int i = 0; i < (int)regions.size(); i++) {
    Region * regionPtr = regions[i];

    int mass = regionPtr->getMass();

    if (mass < lowerThresholdMass) {
        continue;
    }

    int color = (int)((mass - minMass) * colorScaleFactor);

    RGBData rgb((unsigned char)color, (unsigned char)200, (unsigned char)200);

    if (!boundariesOnly) {
        // Draw each point in the Region.
        const std::vector<Point>& points = regionPtr->getPoints();

        for (std::vector<Point>::const_iterator iter = points.begin();
            points.end() != iter;
            ++iter) {
            allRegionsImage->setRGB(iter->x, iter->y, rgb);
        }
    }
}
Point center = regionPtr->getCentroid();

int w = regionPtr->getWidth();
int h = regionPtr->getHeight();
/*allRegionsImage->drawRectangle(center.x - w/2, center.y - h/2,
   center.x + w/2, center.y + h/2,
   RGBAData(255, 0, 0));
*/
regionPtr->drawBoundary(allRegionsImage, getNextColor());
}  // for each region.
}  // end drawRegions

RGBData RegionGrower::getNextColor() {

  it++;

  if (it == colors.end()) {
    it = colors.begin();
  }

  return it->asCOLOR();
}

/*/ ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
*
* Spring 2012
*
* ImageRegionsInfo.h: Contains classes to manage regions in an image. Boundary
* class represents the boundary of a region and provides functions to operate
* on the boundaries of a region. Region class represents the regions in an image
* and provides functions to operate on the regions in an image. ImageRegionsInfo
* class represents all the regions found in the image and provides functions to
* operate on the regions found in the image.
*
* Author: Brian Blaine, Padu Merlotti
** *---------------------------------------------------------------------------*/
#ifndef IMAGE_REGIONS_INFO_H_INCLUDED
#define IMAGE_REGIONS_INFO_H_INCLUDED

#include <vector>
#include <list>
#include <math.h>
#include <float.h>
#include <algorithm>
#include <map>
#include "ColorData.h"

using namespace std;

namespace ImageLib {

class IImage;

class CookedImage;

struct Point {
    int x;
    int y;
    Point(short x, short y) {
        this->x = x;
        this->y = y;
    }
    Point() {}
}

bool operator==(const Point& rhs) const {
    return x == rhs.x && y == rhs.y;
}

bool operator!=(const Point& rhs) const {
    return x != rhs.x || y != rhs.y;
}
};

struct Histograms {
    std::map<WORD, int> hue;
    std::map<WORD, int> saturation;
};

struct HistogramsDelta {
    double hueHistogramDelta;
    double saturationHistogramDelta;
};

/* A boundary is a set of points and a set of turns
  taken to follow the boundary. Used to calculate the area
  of the enclosed region. */

enum Direction {
    Right,
class Boundary {
  std::vector<Point> m_points;
  std::vector<Direction> m_turns;

public:
  void addPoint(const Point& pt, Direction dir);
  int calculateMass() const;
  void drawBoundary(IImage * image, RGBData color);

  inline const std::vector<Direction>& getTurns() const { return m_turns; }

  inline const std::vector<Point>& getPoints() const { return m_points; }
};

class Region {
  Point m_leftMost;
  Point m_rightMost;
  Point m_highest;
  Point m_lowest;
  Point m_centroid;
  bool m_centroidValid;
  CookedImage& m_imageInfo;
  // if I've been merged into a region, this points to him
  Region * m_superceding;
  // the mass of all regions I supercede
  int m_supercededMass;
  // this is all the ones I supercede
  std::vector<Region *> m_superceded;
  std::vector<Point> m_containedPoints;
  Region * * m_indirectHandle;
  void updateExtremities(const Point& pt);
  std::vector<Point>& getPointsInternal();

  Boundary m_boundary;
}
int m_boundaryMass;
void computeBoundaryMass();

int getSimpleMassInternal() const;

void computeCentroid();
//int getNumberOfPoints() const;

double regionSizeWeight;

public:
Region(CookedImage& imageInfo, const Point& newPoint, Region * * handle);
Region * getOwningRegion() const;
Region * getTopRegion();
inline Region * * getHandle() const {
    return m_indirectHandle;
}

void addPoint(const Point& pt);
void supercede(Region *oldRegion);
void ResetIndirectHandle(Region *oldRegion);
int getMass();
const std::vector<Point>& getPoints();
const std::vector<Point>& getBoundaryPoints();

inline bool isSuperceded() const {
    return NULL != m_superceding;
}

Point getCentroid();
Point getSimpleCenter() const;
Point getTop() const {
    return m_highest;
}

int getWidth() const;
int getHeight() const;
HSIData getMeanHSI();
std::vector<HSIData> getHSI();
std::map<WORD, int> getHueHistogram();
std::map<WORD, int> getSaturationHistogram();
Histograms getHistograms();
float getDistance(Region * currentRegion);

double getHistogramDelta(std::map<WORD, int> trainingMap, std::map<WORD, int>
trackingMap);
double getNormalizedArea();

void drawBoundary(IImage * image, RGBData color);
void drawCenter(IImage * image);
void drawCentroid(IImage * image);
void drawCenterMark(IImage * image, Point point, RGBData rgb);

CookedImage& getImageInfo() {
    return m_imageInfo;
}

int getNumberOfPoints() const;

void setRegionSizeWeight(double regionSizeWeight);
double getRegionSizeWeight();

/*/uses the fact that pointers do not have the lower 2 bits set
the color is encoded there*/
const unsigned int REGION_MARK_MASK = 0x00000001;
const unsigned int REGION_PTR_MASK = ~REGION_MARK_MASK;

struct PixelRegionInfo {
    union {
        unsigned int color;
        Region ** ppOwningRegion;
    } r;

    inline bool isOwned() const {
        return 0 != (r.color & REGION_PTR_MASK);
    }

    Region * getOwner() const;
    void setOwner(Region * owner);

    inline bool isMarked() const {
        return 0 != (r.color & REGION_MARK_MASK);
    }

    inline void mark() {
        r.color = REGION_MARK_MASK;
    }

    inline void reset() {
        r.color = 0;
    }

    inline PixelRegionInfo() {
        r.color = 0;
    }
class CookedImage {
    int m_width;
    int m_height;
    std::vector<PixelRegionInfo> m_pixels;

double
    inline void reset(int w, int h) {
        m_pixels.clear();
        m_width = w;
        m_height = h;
        m_pixels.resize(w * h);
    }

    inline int getWidth() const {
        return m_width;
    }

    inline int getHeight() const {
        return m_height;
    }

    inline PixelRegionInfo& getPixel(int x, int y) {
        return m_pixels[m_width * y + x];
    }

    std::vector<PixelRegionInfo> getPixelRegionInfo() {
        return m_pixels;
    }
};

class ImageRegionsInfo {
    // all the regions
    std::vector<Region *> m_regions;
    // the pixels refer to these. The extra layer of indirection allows
    // them to be merged in O(1)
    std::list<Region *> m_regionPtrs;

double
public:
    ImageRegionsInfo();
    ~ImageRegionsInfo();
    Region * addNewRegion(CookedImage& imageInfo, const Point& location, PixelRegionInfo& info);
// gets all the unique regions
// DESTROYS INTERNAL STATE! DO NOT use addRegion until reset() is called
std::vector<Region *> getAllRegions();
std::vector<Region *> getRegionsBetween(int minMass, int maxMass);
Region * getLargestRegion();
float getRegionAlpha(Region *region, int largestRegion);
Point getWeightedCentroid(ImageRegionsInfo& imageRegionsInfo);
Point getColorMatchWeightedCentroid(ImageRegionsInfo& imageRegionsInfo,
    std::map<WORD, int> hueHistogramTraining,
    std::map<WORD, int> saturationHistogramTraining,
    double weight);
Point colorMatchRegionWeightedCentroid;
int ImageRegionsInfo::getEffectiveMass(ImageRegionsInfo& imageRegionsInfo,
    std::map<WORD, int> hueHistogramTraining,
    std::map<WORD, int> saturationHistogramTraining,
    double weight);
void reset();

HistogramsDelta getHistogramsDelta(std::map<WORD, int> trainingHistogram);

double getColorMatch(double weight, double hueHistogramDelta, double
    saturationHistogramDelta);
}
* ImageRegionsInfo.h: Contains classes to manage regions in an image. Boundary
* class represents the boundary of a region and provides functions to operate
* on the boundaries of a region. Region class represents the regions in an image
* and provides functions to operate on the regions in an image. ImageRegionsInfo
* class represents all the regions found in the image and provides functions to
* operate on the regions found in the image
*
* Author: Brian Blaine, Padu Merlotti
** -------------------------------------------------------------------------*/
#include <shlwapi.h>
#include <assert.h>
#include <stdio.h>
#include <list>
#include <algorithm>

#include "ImageRegionsInfo.h"
#include "Image.h"
#include "Tracing.h"
#include "ColorData.h"

using namespace std;

namespace ImageLib {

Region::Region(CookedImage& imageInfo, const Point& newPoint, Region * * handle)
: m_imageInfo(imageInfo) {
    m_leftMost = m_rightMost = m_highest = m_lowest = newPoint;
    m_superceding = NULL;
    m_indirectHandle = handle;
    m_containedPoints.push_back(newPoint);
    m_supercededMass = 0;
    m_centroidValid = false;
}

Region * Region::getOwningRegion() const {
    Region * owner = m_superceding;
    Region * prev = owner;

    while (owner) {
        prev = owner;
        owner = owner->m_superceding;
    }

    return prev;
}

Region * Region::getTopRegion() {
    Region * top = this;
    Region * next = top->m_superceding;

}
while (next) {
    top = next;
    next = top->m_superceding;
}
return top;
}

void Region::addPoint(const Point &pt) {
    m_containedPoints.push_back(pt);
    m_centroidValid = false;
    updateExtremities(pt);
}

void Region::updateExtremities(const Point &pt) {
    if (pt.x < m_leftMost.x) {
        m_leftMost = pt;
    }
    if (pt.x > m_rightMost.x) {
        m_rightMost = pt;
    }
    if (pt.y > m_lowest.y) {
        m_lowest = pt;
    }
    if (pt.y < m_highest.y) {
        m_highest = pt;
    }
}

// get the average of the extremities
Point Region::getSimpleCenter() const {
    Point retval;
    retval.x = m_leftMost.x + (m_rightMost.x - m_leftMost.x) / 2;
    retval.y = m_lowest.y + (m_highest.y - m_lowest.y) / 2;
    return retval;
}

int Region::getWidth() const {
    return (m_rightMost.x - m_leftMost.x);
}

int Region::getHeight() const {
    return (m_lowest.y - m_highest.y);
}
// get the center of mass
Point Region::getCentroid() {
    if (!m_centroidValid) {
        computeCentroid();
    }

    return m_centroid;
}

int Region::getNumberOfPoints() const {
    if (m_containedPoints.empty() == true) {
        return 0;
    }

    int retval = (int)m_containedPoints.size();

    for (std::vector<Region *>::const_iterator it = m_superceded.begin();
        it != m_superceded.end();
        ++it) {
        retval += (*it)->getNumberOfPoints();
    }

    return retval;
}

// Centroid of region. Used to calculate centroid of detected regions and etc
void Region::computeCentroid() {
    m_centroidValid = true;
    int x = 0;
    int y = 0;
    int numTotalPoints = (int)m_containedPoints.size();

    for (int i = 0; i < numTotalPoints; i++) {
        Point pt(m_containedPoints[i]);
        x += pt.x;
        y += pt.y;
    }

    for (std::vector<Region *>::iterator it = m_superceded.begin();
        it != m_superceded.end();
        ++it) {
        int numPoints = (*it)->getNumberOfPoints();
        if (numPoints <= 0) {
            continue;
        }
        
        numTotalPoints += numPoints;
    }

    m_centroid.x = static_cast<float>(x) / numTotalPoints;
    m_centroid.y = static_cast<float>(y) / numTotalPoints;
}
Point pt = (*it)->getCentroid();

x += pt.x * numPoints;

y += pt.y * numPoints;

numTotalPoints += numPoints;
}

m_centroid.x = x / numTotalPoints;

m_centroid.y = y / numTotalPoints;
}

void Region::ResetIndirectHandle(Region *oldRegion) {
    *oldRegion->m_indirectHandle = this;

    /*for (std::vector<Region*>::iterator it = oldRegion->m_superceded.begin();
     it != oldRegion->m_superceded.end();
     ++it)
    {
        ResetIndirectHandle(*it);
    }*/
}

void Region::supercede(Region *oldRegion) {
    // flag it as superceded by this region
    oldRegion->m_superceding = this;
    // update the indirect reference in the table to point to me
    // same for all he supercedes
    ResetIndirectHandle(oldRegion);

    m_superceded.push_back(oldRegion);

    // TODO
    // Experiment - don't copy the points unless you want to draw them
    //m_containedPoints.insert(m_containedPoints.end(),
    //    oldRegion->m_containedPoints.begin(), oldRegion->m_containedPoints.end());
    //    oldRegion->m_containedPoints.clear();
    /*
    m_supercededMass += oldRegion->getSimpleMassInternal();
    updateExtremities(oldRegion->m_leftMost);
    updateExtremities(oldRegion->m_rightMost);
    updateExtremities(oldRegion->m_highest);
    updateExtremities(oldRegion->m_lowest);
int Region::getMass() {
    if (m_boundaryMass < 0) {
        computeBoundaryMass();
    }
    return max(m_boundaryMass, (int)m_containedPoints.size() + m_supercededMass);
}

int Region::getSimpleMassInternal() const {
    return (int)m_containedPoints.size() + m_supercededMass;
}

void Region::drawBoundary(IImage * image, RGBData color) {
    if (m_boundaryMass < 0) {
        computeBoundaryMass();
    }
    m_boundary.drawBoundary(image, color);
}

void Region::drawCenter(IImage * image) {
    Point center = getSimpleCenter();
    image->drawCenterMark(center.x, center.y, RGBData(255, 0, 0));
}

void Region::drawCenterMark(IImage * image, Point point, RGBData rgb) {
    image->drawCenterMark(point.x, point.y, rgb);
}

void Region::drawCentroid(IImage * image) {
    Point center = getCentroid();
    image->drawCenterMark(center.x, center.y, RGBData(255, 255, 255));
}

void Region::computeBoundaryMass() {
    if (m_containedPoints.size() < 10) {
        m_boundaryMass = 10;
        return;
    }
    // Start at the uppermost pixel
    Point startPos = getTop();
    int width = m_imageInfo.getWidth();
int height = m_imageInfo.getHeight();

// a mapping vector of directions
Direction dirs[] = {
    Right,
    UpRight,
    Up,
    UpLeft,
    Left,
    DownLeft,
    Down,
    DownRight,
    Right,
    UpRight,
    Up,
    UpLeft,
    DownLeft,
    Down,
    DownRight
};

// start with the current direction going left
Direction lastDirection = Left;

Point nextPos;

Point lastPos = startPos;

// loop until we hit the start
do {
    bool found = false;
    int start = ((int)lastDirection + 6) % 8;
    int i = start;
    // Search the neighbors of the current boundary pixel
    // in a counter clockwise direction starting with (dir+6)%8, until a pixel
    // inside the region is found.
    do {
        nextPos = lastPos;
        switch (dirs[i]) {
            case Up:
                nextPos.y--;
                break;
            case UpLeft:
                nextPos.x--;
                break;
        }
        found = true;
    } while (found == false);
    lastDirection = nextPos;
} while (lastPos.x == startPos.x && lastPos.y == startPos.y);
```cpp
nextPos.y--; break;

case Left:
    nextPos.x--; break;

case DownLeft:
    nextPos.x--; nextPos.y++; break;

case Down:
    nextPos.y++; break;

case DownRight:
    nextPos.y++; nextPos.x++; break;

case Right:
    nextPos.x++; break;

case UpRight:
    nextPos.y--; nextPos.x++; break;
}

// include only coordinates within the image!
if (nextPos.x >= 0 && nextPos.x < width &&
    nextPos.y >= 0 && nextPos.y < height) {
    PixelRegionInfo& info(m_imageInfo.getPixel(nextPos.x, nextPos.y));

    if (info.getOwner() == this) {
        lastDirection = dirs[i];
        lastPos = nextPos;
        m_boundary.addPoint(nextPos, lastDirection);
        found = true;
        break;
    }
}

i = (i + 1) % 8;
} while (i != start);```
if (!found) {
    Point center = getCentroid();
    DO_TRACE(TraceLib::RegionGrowing, "Failed to find next point. Pos=%d,%d, dir=%d,
    Center=%d,%d\n",
        lastPos.x, lastPos.y, lastDirection, center.x, center.y);
    #ifdef _DEBUG
    // dump a diagnostics file
    FILE * fp = fopen("C:\\debug_image.txt", "wt");

    for (int y = 0; y < m_imageInfo.getHeight(); y++) {
        for (int x = 0; x < m_imageInfo.getWidth(); x++) {
            PixelRegionInfo& info(m_imageInfo.getPixel(x, y));
            fprintf(fp, "%d", info.getOwner() == this ? 1 : 0);
        }
    }
    fprintf(fp, "\n");
    fprintf(fp, "Start: %d, %d
Turns:", startPos.x, startPos.y);
    const std::vector<Direction>& turns(m_boundary.getTurns());
    for (size_t t = 0; t < turns.size(); t++)
        fprintf(fp, " %d", turns[t]);
    fprintf(fp, "\n");
    fclose(fp);
    __debugbreak();
    #endif

    break;
}

while (lastPos != startPos);

m_boundaryMass = m_boundary.calculateMass();
}

const std::vector<Point>& Region::getPoints() {
    return getPointsInternal();
}

std::vector<Point>& Region::getPointsInternal() {
    // coalesce the superceded regions into this one
for (size_t i = 0; i < m_superceded.size(); i++) {
    std::vector<Point>& pts = m_superceded[i]->getPointsInternal();
    m_containedPoints.insert(m_containedPoints.end(), pts.begin(), pts.end());
    pts.clear();
}

return m_containedPoints;
}

const std::vector<Point>& Region::getBoundaryPoints() {
    return m_boundary.getPoints();
}

float Region::getDistance(Region * currentRegion) {
    return 0;
}

Histograms Region::getHistograms() {
    Histograms histograms;

    std::vector<HSIData> regionHSI = this->getHSI();

    for (std::vector<HSIData>::const_iterator iter = regionHSI.begin(); regionHSI.end() != iter; ++iter) {
        // Add 1 to the value for the key.
        ++histograms.hue[iter->hue];
        ++histograms.saturation[iter->saturation];
    }

    return histograms;
}

/*
 * Get a map of the hues from a region. The key is the hue and the value is the number
 * of time that the hue occurs. i.e. a histogram.
 */
std::map<WORD, int> Region::getHueHistogram() {
    std::map<WORD, int> hueHistogram;

    std::vector<HSIData> regionHSI = this->getHSI();

    for (std::vector<HSIData>::const_iterator iter = regionHSI.begin(); regionHSI.end() != iter; ++iter) {
        // Add 1 to the value for the hue key.
        ++hueHistogram[iter->hue];
    }
return hueHistogram;
}

/*
 * Get a map of the saturations from a region. The key is the saturation and the value is
 * the number of time that the saturation occurs. i.e. a histogram.
 */
std::map<WORD, int> Region::getSaturationHistogram() {
  std::map<WORD, int> saturationHistogram;
  std::vector<HSIData> regionHSI = this->getHSI();

  for (std::vector<HSIData>::const_iterator iter = regionHSI.begin(); regionHSI.end() !=
       iter; ++iter) {
    // Add 1 to the value for the saturation key.
    ++saturationHistogram[iter->saturation];
  }

  return saturationHistogram;
}

/* Calculate the total difference between the training map and the current tracking map
 for a current region. */
double Region::getHistogramDelta(std::map<WORD, int> trainingMap, std::map<WORD, int>
trackingMap) {
  double delta = 0;
  std::map<WORD, int>::iterator findIterator;

  for (std::map<WORD, int>::const_iterator trainingMapIterator = trainingMap.begin();
       trainingMapIterator != trainingMap.end(); ++trainingMapIterator) {
    findIterator = trackingMap.find(trainingMapIterator->first);
    if (findIterator != trackingMap.end()) {
      // DO_TRACE(TraceLib::Algorithm, "%d / %d",
      // min(trainingMapIterator->second, findIterator->second),
      // max(trainingMapIterator->second, findIterator->second));
      delta += (float) min(trainingMapIterator->second, findIterator->second) / (float)
               max(trainingMapIterator->second, findIterator->second);
    }
  }

  if (findIterator != trainingMap.end()) {
    // DO_TRACE(TraceLib::Algorithm, "Training histogram size: %d", trainingMap.size());
  }
\[
\text{delta} = \frac{\text{delta}}{\text{trainingMap.size()}};
\]

\[
\text{DO_TRACE}(	ext{TraceLib::Algorithm}, \text{"Region histogram delta: \%f", delta});
\]

\[
\text{return delta;}
\]

\[
\text{std::vector<HSIData> Region::getHSI()} \{
\text{std::vector<HSIData> hsiDatas};
\text{const std::vector<Point>& points = this->getPoints();}

\text{for (std::vector<Point>::const_iterator iter = points.begin(); points.end() != iter; ++iter)} \{
\text{PixelRegionInfo& pixelInfo = this->getImageInfo().getPixel(iter->x, iter->y);}

\text{unsigned int color = pixelInfo.r.color;}

\text{//DO_TRACE(TraceLib::Scoring, \text{"color: \%u \n", color);}

\text{unsigned char red = (color >> 0) & 0xFF;}
\text{unsigned char green = (color >> 8) & 0xFF;}
\text{unsigned char blue = (color >> 16) & 0xFF;}
\text{//unsigned char d = (color >> 24) & 0xFF;}

\text{//DO_TRACE(TraceLib::Scoring, \text{"red: \%u, green: \%u, blue: \%u\n", red, green, blue);}\
\text{RGBData* rgbdata = new RGBData(red, green, blue);}\
\text{HSIData hsidata = rgbdata->toHSI();}
\text{hsiDatas.push_back(hsidata);}\
\text{delete rgbdata;}

\text{//DO_TRACE(TraceLib::Scoring, \text{"Hue: \%u, Saturation: \%u, Intensity: \%u\n", hsidata.hue, hsidata.saturation, hsidata.intensity);}\
\}

\text{return hsiDatas;}
\}

\text{void Region::setRegionSizeWeight(double regionSizeWeight)} \{
\text{this->regionSizeWeight = regionSizeWeight;}
\}

\text{double Region::getRegionSizeWeight()} \{
\text{return this->regionSizeWeight;}
\}

\text{void PixelRegionInfo::setOwner(Region * owner)} \{
\text{r.ppOwningRegion = reinterpret_cast<Region * *>(reinterpret_cast<unsigned int>(owner->getHandle()) & REGION_PTR_MASK)} \}
Region * PixelRegionInfo::getOwner() const {
  Region * * tmp = reinterpret_cast<Region * *>(r.color & REGION_PTR_MASK);

  if (tmp) {
    Region * owner = (*tmp)->getOwningRegion();

    if (owner) {
      return owner;
    }

    return (*tmp);
  }

  return NULL;
}

Region * ImageRegionsInfo::addNewRegion(CookedImage& imageInfo, const Point& location, PixelRegionInfo& info) {
  m_regionPtrs.push_front(NULL);
  Region *newRegion = new Region(imageInfo, location, &m_regionPtrs.front());
  m_regions.push_back(newRegion);
  m_regionPtrs.front() = newRegion;
  info.setOwner(newRegion);
  assert(info.getOwner() == newRegion);
  return newRegion;
}

/**Get the largest region in terms of points*/
Region * ImageRegionsInfo::getLargestRegion() {
  int retval = 0;
  Region * largestRegion = NULL;

  std::vector<Region *> regions = getAllRegions();

  for (std::vector<Region *>::const_iterator it = regions.begin(); it != regions.end(); ++it) {
    if (((*it)->getNumberOfPoints() > retval) {
      largestRegion = *it;
      retval = largestRegion->getNumberOfPoints();
    }
  }

  return largestRegion;
}
return largestRegion;
}

ImageRegionsInfo::ImageRegionsInfo() {
}

ImageRegionsInfo::~ImageRegionsInfo() {
    reset();
}

void ImageRegionsInfo::reset() {
    for (size_t i = 0; i < m_regions.size(); i++)
        //Issue with deleting from this when empty
        //delete m_regions[i];
    m_regions.clear();
    m_regionPtrs.clear();
}

std::vector<Region *> ImageRegionsInfo::getAllRegions() {
    std::vector<Region *> retval;
    for (std::vector<Region *>::iterator it = m_regions.begin(); m_regions.end() != it; ++it)
        if (!(*it)->isSuperceded()) {
            retval.push_back(*it);
        }
    return retval;
}

std::vector<Region *> ImageRegionsInfo::getRegionsBetween(int minMass, int maxMass) {
    std::vector<Region *> retval;
    for (std::vector<Region *>::iterator it = m_regions.begin(); m_regions.end() != it; ++it)
        if (!(*it)->isSuperceded() && (*it)->getMass() >= minMass && (*it)->getMass() <= maxMass) {
            retval.push_back(*it);
        }
    DO_TRACE(TraceLib::Algorithm, "%d/%d regions", retval.size(), m_regions.size());
    return retval;
}
Point ImageRegionsInfo::getWeightedCentroid(ImageRegionsInfo& imageRegionsInfo) {
  unsigned short xNumerator = 0;
  unsigned short yNumerator = 0;
  unsigned short denominator = 0;

  Point weightedCentroid;

  std::vector<Region *> regions = imageRegionsInfo.getRegionsBetween(100, 9000);

  int largestRegion = imageRegionsInfo.getLargestRegion()->getNumberOfPoints();
  DO_TRACE(TraceLib::Algorithm, "Largest Region: %d points.", largestRegion);
  for (std::vector<Region *>::iterator iter = regions.begin(); regions.end() != iter; ++iter) {
    int regionAlpha = (unsigned short) getRegionAlpha((*iter), largestRegion);
    xNumerator += regionAlpha * (*iter)->getCentroid().x;
    yNumerator += regionAlpha * (*iter)->getCentroid().y;
    denominator += regionAlpha;
  }
  if (denominator != 0) {
    weightedCentroid.x = xNumerator / denominator;
    weightedCentroid.y = yNumerator / denominator;
  }

  // Centroid of detected regions. Equation 4.
  return weightedCentroid;
}

Point ImageRegionsInfo::getColorMatchWeightedCentroid(ImageRegionsInfo& imageRegionsInfo,
  std::map<WORD, int> hueHistogramTraining,
  std::map<WORD, int> saturationHistogramTraining,
  double weight) {
  double xNumerator = 0;
  double yNumerator = 0;
  double denominator = 0;
  double alpha = 0.0;
Point colorWeightedCentroid;
std::vector<Region *> regions = imageRegionsInfo.getRegionsBetween(100, 9000);

std::vector<Region *>::iterator iter = regions.begin();

int largestRegion = imageRegionsInfo.getLargestRegion()->getNumberOfPoints();

for (; regions.end() != iter; ++iter) {
    double hueHistogramDelta = (*iter)->getHistogramDelta(hueHistogramTraining, (*iter)->getHueHistogram());
    double saturationHistogramDelta = (*iter)->getHistogramDelta(saturationHistogramTraining, (*iter)->getSaturationHistogram());

    DO_TRACE(TraceLib::Algorithm, "HHD %f", hueHistogramDelta);
    DO_TRACE(TraceLib::Algorithm, "SHD %f", saturationHistogramDelta);

    //First param weight in getColorMatch is "a weighting factor that can be used to emphasize or deemphasize the effect of hue relative to saturation on the color match"
    double colorMatch = getColorMatch(0.2, hueHistogramDelta, saturationHistogramDelta);

    DO_TRACE(TraceLib::Algorithm, "colorMatch %f", colorMatch);

    alpha = getRegionAlpha(*iter, largestRegion);
    DO_TRACE(TraceLib::Algorithm, "Alpha %f", alpha);
    xNumerator += (alpha + weight * colorMatch) * (*iter)->getCentroid().x;
    yNumerator += (alpha + weight * colorMatch) * (*iter)->getCentroid().y;
    denominator += alpha + weight * colorMatch;
}

colorWeightedCentroid.x = (short)(xNumerator / denominator);

colorWeightedCentroid.y = (short)(yNumerator / denominator);

return colorWeightedCentroid;
}

/*Effective mass - Equation 9 in Algorithm.doc*/
int ImageRegionsInfo::getEffectiveMass(ImageRegionsInfo& imageRegionsInfo, std::map<WORD, int> hueHistogramTraining, std::map<WORD, int> saturationHistogramTraining, double weight) {
    double numerator = 0.0;
    double denominator = 0.0;

int effectiveMass;

std::vector<Region *> regions = imageRegionsInfo.getRegionsBetween(100, 9000);

std::vector<Region *>::iterator iter = regions.begin();

int largestRegion = imageRegionsInfo.getLargestRegion()->getNumberOfPoints();

double alpha = getRegionAlpha(*iter, largestRegion);

for (; regions.end() != iter; ++iter) {
    double hueHistogramDelta = (*iter)->getHistogramDelta(hueHistogramTraining, (*iter)-
>getHueHistogram());
    double saturationHistogramDelta = (*iter)-
>getHistogramDelta(saturationHistogramTraining, (*iter)->getSaturationHistogram());

    double colorMatch = getColorMatch(weight, hueHistogramDelta, saturationHistogramDelta);

    numerator += (alpha + weight * colorMatch) * (*iter)->getNumberOfPoints();
    denominator += alpha + weight * colorMatch;
}

effectiveMass = numerator / denominator;

DO_TRACE(TraceLib::Algorithm, "Effective mass %f", effectiveMass);

return effectiveMass;
}

/*Area is defined as the number of points in a region.*/
/* A region's alpha is the area divided by the area of the largest region.*/
/* Used in calculating the weighted centroid of all regions. See Algorithm.doc*/
/* equation 4.*/
float ImageRegionsInfo::getRegionAlpha(Region *region, int largestRegion) {
    //DO_TRACE(TraceLib::Algorithm, "Region Points: %d", region->getNumberOfPoints());
    //DO_TRACE(TraceLib::Algorithm, "Region Alpha: %f", (region->getNumberOfPoints() / (float) largestRegion));

    return (float) region->getNumberOfPoints() / (float) largestRegion;
}

/**
* The difference between the trained and detected region histograms for hue and saturation.
  * Equation 5 and 6.
  */

HistogramsDelta ImageRegionsInfo::getHistogramsDelta(std::map<WORD, int> trainingHistogram)
{
  HistogramsDelta histogramsDelta;
  histogramsDelta.hueHistogramDelta = 0.0;
  histogramsDelta.saturationHistogramDelta = 0.0;

  for (std::vector<Region *>::iterator iter = m_regions.begin(); m_regions.end() != iter; ++iter)
  {
    histogramsDelta.hueHistogramDelta += (*iter)->getHistogramDelta(trainingHistogram,
    (*iter)->getHueHistogram());
    histogramsDelta.saturationHistogramDelta += (*iter)->
    >getHistogramDelta(trainingHistogram, (*iter)->getSaturationHistogram());
  }

  return histogramsDelta;
}

/****************************************************************************
  * Equation 7.
  * Color match between the trained and detected region.
  */
  double ImageRegionsInfo::getColorMatch(double weight, double hueHistogramDelta, double
  saturationHistogramDelta) {
    return hueHistogramDelta + (weight * saturationHistogramDelta) / (1.0 + weight);
  }

void Boundary::addPoint(const Point& pt, Direction dir) {
  m_points.push_back(pt);
  m_turns.push_back(dir);
}

int Boundary::calculateMass() const {
  double a = m_turns.size();        // the area includes the boundary!
  int x = m_turns.size();
  int y = x;

  for (std::vector<Direction>::const_iterator it = m_turns.begin();
    it != m_turns.end();
    ++it) {
    switch (*it) {
    case Right:
a -= y; x++; break;

case UpRight:
a -= (y + 0.5); y++; x++; break;

case Up:
y++; break;

case UpLeft:
a += (y + 0.5); y++; x--; break;

case Left:
a += y; x--; break;

case DownLeft:
a += (y - 0.5); y--; x--; break;

case Down:
y--; break;

case DownRight:
a -= (y - 0.5); y--; x++; break;

} }

return (int)a;
}

void Boundary::drawBoundary(IImage * image, RGBData color) {
    for (size_t i = 0; i < m_points.size(); i++) {
        Point& pt(m_points[i]);
        image->setRGB(pt.x, pt.y, color);
    }
}
/* -------------------------------*/

* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
*
* Spring 2012
*
* Image.h: Definition class to represent and operate on images.
*
* Author: Brian Blaine, Padu Merlotti
* ** -----------------------------------------------*/

#ifndef IMAGELIB_IMAGE_H_INCLUDED
#define IMAGELIB_IMAGE_H_INCLUDED

#include "ColorData.h"
namespace ImageLib {

// interface for accessing images

class IImage {

public:
    virtual ~IImage() {}

    virtual int getWidth() const = 0;
    virtual int getHeight() const = 0;

};

#endif /* IMAGELIB_IMAGE_H_INCLUDED */
virtual RGBData getRGB(int x, int y) = 0;
virtual void setRGB(int x, int y, const RGBData& color) = 0;

virtual void drawRectangle(int xUpperLeft, int yUpperLeft, int xLowerRight, int yLowerRight, RGBData color);

virtual void drawCenterMark(int x, int y, RGBData color);

virtual BYTE * getData() = 0;

virtual void SaveBitmapToFile(BYTE * pBitmapBits, LONG lWidth, LONG lHeight, WORD wBitsPerPixel, LPCTSTR lpszFileName) = 0;

void IImage::drawCircle(int xCenter, int yCenter, float radius, RGBData color);

namespace ImageLib {

void IImage::drawCircle(int xCenter, int yCenter, float radius, RGBData color) {
    float x, y, z;
    z = 0.5;
    float r2 = radius * radius;
    for (x = -radius; x <= radius; x++) {
        y = sqrt(r2 - x * x) + z;
        setRGB(xCenter + (int) x, yCenter + (int) y, color);
        setRGB(xCenter + (int) x, yCenter - (int) y, color);
    }
}
void IImage::drawRectangle(int xUpperLeft, int yUpperLeft,
    int xLowerRight, int yLowerRight,
    RGBData color) {
    xUpperLeft = std::min(std::max(0, xUpperLeft), getWidth() - 1);
    yUpperLeft = std::min(std::max(0, yUpperLeft), getHeight() - 1);
    xLowerRight = std::max(0, std::min(getWidth() - 1, xLowerRight));
    yLowerRight = std::max(0, std::min(getHeight() - 1, yLowerRight));

    for (int x = xUpperLeft; x < xLowerRight; x++) {
        setRGB(x, yUpperLeft, color);
        setRGB(x, yLowerRight, color);
    }

    for (int y = yUpperLeft; y < yLowerRight; y++) {
        setRGB(xUpperLeft, y, color);
        setRGB(xLowerRight, y, color);
    }
}

void IImage::drawCenterMark(int x, int y, RGBData color) {
    const int LEN = 10;
    int w = getWidth();
    int h = getHeight();

    for (int i = std::max(x - LEN, 0), j = std::max(y - LEN, 0), j2 = std::min(y + LEN, h - 1);
        i <= std::min(x + LEN, w - 1) && j <= std::min(y + LEN, h - 1) && j2 >= std::max(y - LEN, 0);
        i++, j++, j2--) {
        setRGB(i, j, color);
        setRGB(i, j2, color);
    }
}
/* Thesis: Vision Based Person Following Using an Improved
   Image Segmentation Approach

   Spring 2012

   ColorData.h: Contains structures to store color info about the image and
   utility methods.

   Author: Brian Blaine, Padu Merlotti

   *****************************************************************************/

#ifndef IMAGELIB_COLORDATA_H_INCLUDED
#define IMAGELIB_COLORDATA_H_INCLUDED

#define NOMINMAX
#include <windows.h>

namespace ImageLib {

struct HSIDataNormalized {
    float hue;
    float saturation;
    float intensity;
};

struct HSIData {
    HSIData() : hue(0xFFFF) {
    }

    HSIData(WORD h, WORD s, WORD i) : hue(h), saturation(s), intensity(i) {
    }

    bool isValid() const {
        return 0xFFFF != hue;
    }
}

};
WORD hue;
WORD saturation;
WORD intensity;

bool operator< (const HSIData& rhs) const {
    if (hue < rhs.hue) {
        return true;
    }

    if (hue > rhs.hue) {
        return false;
    }

    if (saturation < rhs.saturation) {
        return true;
    }

    if (saturation > rhs.saturation) {
        return false;
    }

    return intensity < rhs.intensity;
}

class RGBData {

public:
    unsigned char blue() const {
        return GetBValue(color);
    }

    unsigned char green() const {
        return GetGValue(color);
    }

    unsigned char red() const {
        return GetRValue(color);
    }

    unsigned int asCOLOR() const {
        return color;
    }

    RGBData(unsigned char r, unsigned char g, unsigned char b) {
        color = RGB(r, g, b);
    }

    RGBData(const RGBData& rhs) {

color = rhs.color;
}

RGBData(COLORREF clr) : color(clr) {
}

RGBData() : color(0) {
}

bool operator==(const RGBData& rhs) const {
    return color == rhs.color;
}

bool operator<(const RGBData& rhs) const {
    return color < rhs.color;
}

static RGBData black() {
    return RGBData(0, 0, 0);
}

static RGBData white() {
    return RGBData(255, 255, 255);
}

static RGBData pink() {
    return RGBData(255, 192, 203);
}

HSIData toHSI() const;
void setIntensity(WORD intensity);

private:
    COLORREF color;
};
namespace ImageLib {

HSIData RGBData::toHSI() const {
    WORD h, i, s;
    ColorRGBToHLS(color, &h, &i, &s);

    HSIData retval;
    retval.hue = h;
    retval.saturation = s;
    retval.intensity = i;
    return retval;
}

void RGBData::setIntensity(WORD intensity) {
    WORD h, i, s;
    ColorRGBToHLS(color, &h, &i, &s);
    i = intensity;
    color = ColorHLSToRGB(h, i, s);
}
}

/* -------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* Spring 2012
*
* AbstractArbiter.h: The arbiter is responsible for selecting which behavior
* or set of behaviors that will have their output carried to the intended
* destination
* */
class CAbstractArbiter {

public:
    CAbstractArbiter(void) {
    };
    ~CAbstractArbiter(void) {
    };

    void virtual Arbitrate() = 0;
};
#define OUTPUT_SPEED       0x100
#define OUTPUT_MAX_SPEED   0x150
#define OUTPUT_TURN     0x200
#define OUTPUT_PAN     0x300
#define OUTPUT_DELTA_PAN 0x350
#define OUTPUT_TILT     0x400
#define OUTPUT_STOP_MODE 0x500

using namespace std;

class CBehavior {
public:
    CBehavior(int frequency);

    //overloading operator LessThan based on priority
    virtual bool operator<(CBehavior& beh) const;

    //maximum priority is 0
    int m_priority;

    virtual void ExecuteWithFrequency();
    virtual void Execute() = 0;
    virtual void Reset() = 0;
    virtual string GetName() = 0;

    int GetFrequency() {
        return m_frequency;
    }

    void SetFrequency(int frequency);

    std::map<int, double> outputList;

private:
    int m_frequency;
    double m_interval;
    bool isItTime();
    void ResetTimer();

protected:
    //timer stuff
    int getTimeAccum();
    int m_timeLimit;
    int m_timeAccum;
    long long m_tickStarted;
};
/* ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* 
* Spring 2012
* 
* <filename>.
* <very brief file description>
* 
* Author: Brian Blaine
** ---------------------------------------------------------------------------*/

#include "Behavior.h"
#include <windows.h>

bool CBehavior::operator <(CBehavior& beh) const {
    return this->m_priority < beh.m_priority;
}

CBehavior::CBehavior(int frequency) : m_priority(INT_MAX) {
    SetFrequency(frequency);
    QueryPerformanceCounter((LARGE_INTEGER *)&m_tickStarted);
}
void CBehavior::SetFrequency(int frequency) {
    m_frequency = frequency;  // in Hertz
    m_interval = ((1.0 / frequency) * 1000);  // in milliseconds
}

void CBehavior::ExecuteWithFrequency() {
    if (isItTime()) {
        Execute();
        ResetTimer();
    }
}

void CBehavior::ResetTimer() {
    QueryPerformanceCounter((LARGE_INTEGER *)&m_tickStarted);
}

bool CBehavior::isItTime() {
    return (getTimeAccum() > m_interval);
}

int CBehavior::getTimeAccum() {
    LARGE_INTEGER ticksPerSecond;
    LARGE_INTEGER tick;  // A point in time

    // get the high resolution counter's accuracy
    QueryPerformanceFrequency(&ticksPerSecond);

    // what time is it?
    QueryPerformanceCounter(&tick);

    // convert the tick number into the number of milliseconds
    // since the object was created...
    return int((((double)(tick.QuadPart - m_tickStarted)) / (double)ticksPerSecond.QuadPart) * 1000.0);
}
class CPanTiltArbiter : public CAbstractArbiter {

public:
    CPanTiltArbiter();
    ~CPanTiltArbiter();

    virtual void Arbitrate();

    bool m_panEnabled;
}
bool m_tiltEnabled;

double GetLastPan() {
    return m_pan;
};

double GetLastTilt() {
    return m_tilt;
};

private:
    CPhidgetServoHandle _servo;
    double m_pan;
    double m_tilt;

    //values obtained empirically
    void SetServo(int servoID, double servoCount);
};
/* Thesis: Vision Based Person Following Using an Improved
   * Image Segmentation Approach
   * Spring 2012
   * PanTiltArbiter.cpp: Implementation for pan and tilt.
   * The arbiter is responsible for implementing a concrete arbiter
   * that will select which behavior will have its output carried
   * to the intended destination arbitred device
   * Author: Brian Blaine, Padu Merlotti
   ** -------------------------------------------------------------------------*/

#include "PanTiltArbiter.h"
#include "BehaviorList.h"
#include "Tracing.h"

CPanTiltArbiter::CPanTiltArbiter() {
    CPhidgetServo_create(&_servo);
    CPhidget_open((CPhidgetHandle)_servo, -1);

    m_pan = 0.0;
    m_tilt = 0.0;
}

CPanTiltArbiter::~CPanTiltArbiter() {
    CPhidget_close((CPhidgetHandle)_servo);
    CPhidget_delete((CPhidgetHandle)_servo);
}

void CPanTiltArbiter::Arbitrate() {
    CBehaviorList * behList = CBehaviorList::Instance();
    CBehavior * beh;

    // Arbitrate pan. Zero degrees is centered in front of the platform. Positive is to camera right
    if (m_panEnabled) {
        for (int i = 0; i < behList->getCount(); i++) {

beh = behList->getItem(i);

if (beh->outputList.find(OUTPUT_PAN) != beh->outputList.end()) {
    // Send command to pan and tilt controller
    m_pan = beh->outputList[OUTPUT_PAN];
    DO_TRACE(TraceLib::PanTiltControl, "pan = %f. Controlled by %s", m_pan, beh->GetName().c_str());
    SetServo(SERVO_PAN, m_pan);
    break;
}
}

/* Arbitrate tilt. Zero degrees is centered in front of the platform. Positive is up*/
if (m_tiltEnabled) {
    for (int i = 0; i < behList->getCount(); i++) {
        beh = behList->getItem(i);

        if (beh->outputList.find(OUTPUT_TILT) != beh->outputList.end()) {
            // Send command to pan and tilt controller
            m_tilt = beh->outputList[OUTPUT_TILT];
            DO_TRACE(TraceLib::PanTiltControl, "tilt = %f. Controlled by %s", m_tilt, beh->GetName().c_str());
            SetServo(SERVO_TILT, m_tilt);
            break;
        }
    }
}

void CPanTiltArbiter::SetServo(int servoID, double servoCount) {
    CPhidgetServo_setMotorPosition(_servo, servoID, servoCount);
}

/* ==========================================================================
 * Thesis: Vision Based Person Following Using an Improved
 * Image Segmentation Approach
 */
The arbiter is responsible for implementing a concrete arbiter that will select which behavior will have its output carried to the intended destination arbitrated device.

Author: Brian Blaine, Padu Merlotti

```
#include "SegwayRmp.h"
#include "AbstractArbiter.h"
#include "behControlPan.h"

#define MAX_SPEED_METERS_PER_SEC     3.5555
#define MAX_TURN_DEG_PER_SEC        14.0000

class CSegwayArbiter : public CAbstractArbiter {

public:
    CSegwayArbiter(CSegwayRmp * rmp_object, CbehControlPan * behControlPan)
    : m_rmp(rmp_object)
        , m_maxSpeedMS(0.8)           //MAX_SPEED_METERS_PER_SEC)
        , m_maxTurnDegS(3.0)          //MAX_TURN_DEG_PER_SEC )
        , m_behControlPan(behControlPan)
        , m_currentSpeed(0.0)
        , m_currentTurn(0.0) {
    }

    ~CSegwayArbiter() {
    }

    virtual void Arbitrate();

    CSegwayRmp * m_rmp;
    CbehControlPan * m_behControlPan;
    double m_maxSpeedMS;
    double m_maxTurnDegS;

    double GetCurrentSpeed() {
        return m_currentSpeed;
    }

    double GetCurrentTurn() {
        return m_currentTurn;
    }
```
public:
    double m_currentSpeed;
    double m_currentTurn;
};
#include "BehaviorList.h"
#include "Tracing.h"

/*Go through all behaviors with output targeted to 
the segway rmp and select the one with higher priority*/
void CSegwayArbiter::Arbitrate() {
    CBehaviorList * behList = CBehaviorList::Instance();
    CBehavior * beh;

double velMS = 0.0;
double turnDegS = 0.0;
bool eStop = false;

    //check if there is any E-Stop message (OUTPUT_STOP_MODE)
    for (int i = 0; i < behList->getCount(); i++) {
        beh = behList->getItem(i);

        if (beh->outputList.find(OUTPUT_STOP_MODE) != beh->outputList.end()) {
            //e-stop. set speed and turn to zero
            velMS = 0.0;
            turnDegS = 0.0;
            eStop = true;
            DO_TRACE(TraceLib::Segway, "E-STOP from: %s. Priority = %d",
                     beh->GetName().c_str(), beh->m_priority);
            break;
        }
    }

    if (!eStop) {
        //arbitrate maximum speed
        for (int i = 0; i < behList->getCount(); i++) {
            beh = behList->getItem(i);

            if (beh->outputList.find(OUTPUT_MAX_SPEED) != beh->outputList.end()) {
                m_maxSpeedMS = MAX_SPEED_METERS_PER_SEC * beh->outputList[OUTPUT_MAX_SPEED];
                DO_TRACE(TraceLib::Segway, "Winner behavior for max speed: %s. (%f). Priority = %d",
                         beh->GetName().c_str(), m_maxSpeedMS, beh->m_priority);
                break;
            }
        }

        //arbitrate speed
        for (int i = 0; i < behList->getCount(); i++) {
            beh = behList->getItem(i);

            if (beh->outputList.find(OUTPUT_SPEED) != beh->outputList.end()) {
                //send command to segway rmp and don't check for
// speed commands anymore
velMS = beh->outputList[OUTPUT_SPEED] * m_maxSpeedMS;
DO_TRACE(TraceLib::Segway, "Winner behavior for speed: %s. (%f). Priority = %d",
    beh->GetName().c_str(), velMS, beh->m_priority);
break;
}
}

// arbitrate steering
for (int i = 0; i < behList->getCount(); i++) {
    beh = behList->getItem(i);
    if (beh->outputList.find(OUTPUT_TURN) != beh->outputList.end()) {
        // send command to segway rmp and don't check for
        // speed commands anymore
        double turn = beh->outputList[OUTPUT_TURN];
        turnDegS = m_maxTurnDegS * turn;

        DO_TRACE(TraceLib::Segway, "Winner behavior for turn (turn,degs): %s. (%f,%f).
            Priority = %d",
            beh->GetName().c_str(), turn, turnDegS, beh->m_priority);
        break;
    }
}    // if (!eStop)

m_currentSpeed = m_currentSpeed;
m_currentTurn = turnDegS;
m_rmp->SendActuation(velMS, turnDegS);
}
class CPIDController {

public:
    CPIDController(double kp = 1, double ki = 0, double kd = 0);
    ~CPIDController(void);

public:
    double GetKp() {  
        return _kp;
    }

    double GetKi() {  
        return _ki;
    }
};
double GetKd() {
    return _kd;
};

void SetKp(double kp) {
    _kp = kp;
}

void SetKi(double ki) {
    _ki = ki;
}

void SetKd(double kd) {
    _kd = kd;
}

public:
    double CalcOutput(double error);
    void Reset();

private:
    bool _isFirstRun;
    double _kp, _ki, _kd;
    double _lastError, _integralError;
};
/* ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* Spring 2012
* PIDController.cpp: Implementation for a discrete PID controller. Used by
* various PID controllers for control.
* Author: Brian Blaine
* ---------------------------------------------------------------------------*/
#include <iostream>
#include <fstream>
#include "PIDController.h"

using namespace std;

int main() {
    cout << "Starting the app\n";

    ifstream iFile;
    iFile.open("input.txt");

    ofstream oFile;
    oFile.open("output.txt");

    double refSignal, control = 0.0, output = 0.0;

    CPIDController pid(1.3, 0.5, 0.1);

    if (iFile.is_open()) {
        while (!iFile.eof()) {
            iFile >> refSignal;

            control = pid.CalcOutput(refSignal - output);
        }
    }
// apply output to process
    output += control;

    oFile << output << "\n";

iFile.close();
oFile.close();

return 0;
}

CPIDController::CPIDController(double kp, double ki, double kd) {
    _kp = kp;
    _ki = ki;
    _kd = kd;

    _isFirstRun = true;
    _integralError = _lastError = 0.0;
}

CPIDController::~CPIDController(void) {
}

double CPIDController::CalcOutput(double error) {
    if (_isFirstRun) {
        _lastError = error;
        _isFirstRun = false;
        return (error * _kp);
    }

    _integralError += 0.5 * (error + _lastError);

double derivativeError = (error - _lastError);
_lastError = error;

    return
        (error * _kp) + // proportional
        (_integralError * _ki) + // integral
        (derivativeError * _kd); // differencial
}

void CPIDController::Reset() {
    _isFirstRun = true;
    _integralError = _lastError = 0.0;
class CBehavior {

public:
    CBehavior(int frequency);

    //overloading operator LessThan based on priority
    virtual bool operator<(CBehavior& beh) const;

    //maximum priority is 0
    int m_priority;

    virtual void ExecuteWithFrequency();
    virtual void Execute() = 0;
    virtual void Reset() = 0;
    virtual string GetName() = 0;

    int GetFrequency() {
        return m_frequency;
    }

    void SetFrequency(int frequency);
std::map<int, double> outputList;

private:
int m_frequency;
double m_interval;
bool isITime();
void ResetTimer();

protected:
//timer stuff
int getTimeAccum();
int m_timeLimit;
int m_timeAccum;
long long m_tickStarted;
};

/*------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
* 
* Spring 2012
*
* BehaviorList.cpp: The behavior list is a singleton that holds a unique
* polymorphic list of behaviors present in the system.
* The main methods are AddBehavior() and ExecuteAll().
*/
#include "BehaviorList.h"
#include <algorithm>

CBehaviorList * CBehaviorList::_instance = 0;

CBehaviorList * CBehaviorList::Instance() {
    if (_instance == 0) {
        _instance = new CBehaviorList();
    }

    return _instance;
}

bool PredBehaviorLessThan(CBehavior * beh1, CBehavior * beh2) {
    return *beh1 < *beh2;
}

/*CBehaviorList is responsible for releasing the memory occupied by the Behaviors in its list.*/
void CBehaviorList::AddBehavior(CBehavior * behavior) {
    _behaviorList.push_back(behavior);

    sort(_behaviorList.begin(), _behaviorList.end(), PredBehaviorLessThan);
}

void CBehaviorList::ExecuteAll() {
    vector<CBehavior *>::iterator iter;

    for (iter = _behaviorList.begin(); iter != _behaviorList.end(); iter++)
        (*iter)->Execute();
}

void CBehaviorList::Reset() {
    vector<CBehavior *>::iterator iter;

    for (iter = _behaviorList.begin(); iter != _behaviorList.end(); iter++)
        (*iter)->Reset();
}
CBehaviorList::~CBehaviorList(void) {
    vector<CBehavior *>::iterator iter;

    for (iter = _behaviorList.begin(); iter != _behaviorList.end(); iter++)
        delete (*iter);
}

int CBehaviorList::getCount() {
    return (int)_behaviorList.size();
}

CBehavior * CBehaviorList::getItem(int idx) {
    return _behaviorList.at(idx);
}

/********************************************
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
*
* Spring 2012
*
* behControlPan.h: Definition file for controlling the pan of the camera.
*
* Author: Brian Blaine, Padu Merlotti
** -----------------------------------------------*/
#pragma once
#include "behcontrolservo.h"
#include "PanTiltArbiter.h"

using namespace std;

class CbehControlPan : public CbehControlServo {

public:
CbehControlPan(int frequency, int setPoint, CPersonSearch &personSearch, CPanTiltArbiter *panTiltArbiter, int scale, int upperLimit, int lowerLimit);

public:
  virtual void Execute();
  virtual void Reset();
  virtual string GetName();

private:
  int PixelsToDegrees(int controlVar) {
    return 0;
  }

  CPanTiltArbiter * m_panTiltArbiter;
};

string CbehControlPan::GetName() {
  return "behControlPan";
}

CbehControlPan::CbehControlPan(int frequency, int setPoint, CPersonSearch &personSearch, CPanTiltArbiter *panTiltArbiter, int scale, int upperLimit, int lowerLimit) : CbehControlServo(frequency, setPoint, personSearch,
```cpp
scale, upperLimit, lowerLimit) {
    m_pid->SetKp(0.3);
    m_pid->SetKi(0.0);
    m_pid->SetKd(0.2);
    m_panTiltArbiter = panTiltArbiter;
}

void CbehControlPan::Execute() {
    if (m_personSearch.m_trackingInfo.IsTargetLocked) {
        // compute error compared to set reference
        DO_TRACE(TraceLib::PanTiltControl, "m_personSearch.m_trackingInfo.x_pos %f",
                 m_personSearch.m_trackingInfo.x_pos);
        DO_TRACE(TraceLib::PanTiltControl, "setPoint %f", m_setPoint);
        double error = m_setPoint + m_personSearch.m_trackingInfo.x_pos;
        DO_TRACE(TraceLib::PanTiltControl, "error %f", error);

        // compute control variable using PID controller
        double controlVar = m_pid->CalcOutput(error);
        DO_TRACE(TraceLib::PanTiltControl, "controlVar %f", controlVar);

        // scale control variable into servo range
        double delta = controlVar * m_scale;
        DO_TRACE(TraceLib::PanTiltControl, "delta %f", delta);
        double output;

        if (outputList.end() == outputList.find(OUTPUT_PAN)) {
            output = ServoCenter();
            DO_TRACE(TraceLib::PanTiltControl, ">>>>>>>>>>>>>>>>>>>>>CENTERING CAMERA!!!!!!");
        } else {
            output = outputList[OUTPUT_PAN] + delta;
        }

        output = m_panTiltArbiter->GetLastPan() + delta;
        Clip(output);

        DO_TRACE(TraceLib::PanTiltControl, "error=%f,controlVar=%f,delta=%f,pan output=%f",
                 error, controlVar, delta, output);

        // generate output
        DO_TRACE(TraceLib::PanTiltControl, "Pan output = %f", output);
        outputList[OUTPUT_PAN] = output;
    } else {
        // outputList.erase(OUTPUT_PAN);
        DO_TRACE(TraceLib::PanTiltControl, "NO Target");
    }
```
m_pid->Reset();
}
}

void CbehControlPan::Reset() {
    outputList[OUTPUT_PAN] = ServoCenter();
}

/* ---------------------------------------------------------------------------
 * Thesis: Vision Based Person Following Using an Improved
 * Image Segmentation Approach
 * 
 * Spring 2012
 * 
 * behControlTilt.h: Definition file to control the tilt of the camera.
 * 
 * Author: Brian Blaine
 ** ---------------------------------------------------------------------------*/
#pragma once
#include "behcontrolservo.h"
#include "PanTiltArbiter.h"

using namespace std;

class CbehControlTilt : public CbehControlServo {

public:
    CbehControlTilt(int frequency, int setPoint, CPersonSearch &personSearch, CPanTiltArbiter *panTiltArbiter, int scale, int upperLimit, 
                     int lowerLimit);

public:
    virtual void Execute();
    virtual void Reset();
    virtual string GetName();

private:
    int PixelsToDegrees(int controlVar) {
        return 0;
    }
CPanTiltArbiter * m_panTiltArbiter;

/* ---------------------------------------------------------------------------
* Thesis: Vision Based Person Following Using an Improved Image Segmentation Approach
* Spring 2012
* behControlTilt.cpp: Implementation class to control the tilt of the robot based on the y coordinate of the person.
* Author: Brian Blaine, Padu Merlotti
** -------------------------------------------------------------------------*/
#include "behControlTilt.h"
#include "Tracing.h"

string CbehControlTilt::GetName() {
    return "behControlTilt";
}

CbehControlTilt::CbehControlTilt(int frequency,
    , int setPoint
    , CPersonSearch &personSearch
    , CPanTiltArbiter * panTiltArbiter
    , int scale, int upperLimit, int lowerLimit)
    : CbehControlServo(frequency,
        , setPoint, personSearch, scale
        , upperLimit, lowerLimit) {
    m_pid->SetKp(0.1);
    m_pid->SetKi(0.0);
    m_pid->SetKd(0.0);
    m_panTiltArbiter = panTiltArbiter;
}
void CbehControlTilt::Execute() {
    if (m_personSearch.m_trackingInfo.IsTargetLocked) {
        //compute error compared to set reference
        DO_TRACE(TraceLib::PanTiltControl, "m_personSearch.m_trackingInfo.y_pos %f", m_personSearch.m_trackingInfo.y_pos);
        DO_TRACE(TraceLib::PanTiltControl, "setPoint %f", m_setPoint);
        double error = m_setPoint - m_personSearch.m_trackingInfo.y_pos;
        DO_TRACE(TraceLib::PanTiltControl, "error %f", error);
        //compute control variable using PID controller
        double controlVar = m_pid->CalcOutput(error);
        DO_TRACE(TraceLib::PanTiltControl, "controlVar %f", controlVar);
        //scale control variable into servo range
        double delta = controlVar * m_scale;
        double output;
        if (outputList.end() == outputList.find(OUTPUT_TILT)) {
            output = ServoCenter();
            DO_TRACE(TraceLib::PanTiltControl, ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>CENTERING CAMERA!!!!!!!");
        } else {
            output = outputList[OUTPUT_TILT] + delta;
        }
        output = m_panTiltArbiter->GetLastTilt() + delta;
        Clip(output);
        //generate output
        DO_TRACE(TraceLib::PanTiltControl, "Tilt output = %f", output);
        outputList[OUTPUT_TILT] = output;
    } else {
        //outputList.erase(OUTPUT_TILT);
        DO_TRACE(TraceLib::PanTiltControl, "NO Target");
    }
}

void CbehControlTilt::Reset() {
    outputList[OUTPUT_TILT] = ServoCenter();
}
class CbehControlSpeed : public CBehavior {

public:
    CbehControlSpeed(int frequency, CPersonSearch &personSearch, CSegwayArbiter * arbiter);
    ~CbehControlSpeed(void);

    //speed is in meters per second
    double getSpeed() {
        return m_speed;
    }

    virtual void Execute();
    virtual void Reset();
    virtual string GetName();

public:
    double m_speed;

protected:
    CPersonSearch &m_personSearch;
    CPIDController m_pid;
    CSegwayArbiter * m_arbiter;
};
#include "behControlSpeed.h"
#include <math.h>

string CbehControlSpeed::GetName() {
  return "behControlSpeed";
}

CbehControlSpeed::CbehControlSpeed(int frequency, CPersonSearch &personSearch, CSegwayArbiter *arbiter)
  : CBehavior(frequency),
    m_personSearch(personSearch),
    m_arbiter(arbiter) {
  m_speed = 0.0;

  m_pid.SetKp(1.0);
  m_pid.SetKi(0.0);
  m_pid.SetKd(0.0);
};

CbehControlSpeed::~CbehControlSpeed(void) {
}

void CbehControlSpeed::Execute() {
  if (m_personSearch.m_trackingInfo.IsTargetLocked) {
    double speed = 0.0;
    double mass = m_personSearch.m_trackingInfo.mass;
    //we only compute if mass is decreasing
    // (target is getting more distant from us)

    if (mass < 0) {
      double normTurnRate = abs(m_arbiter->m_currentTurn / m_arbiter->m_maxTurnDegS);
      speed = -(mass * (1 - normTurnRate));       //if turning, we reduce speed
    }

    outputList[OUTPUT_SPEED] = speed;
  } else {

  }
}
outputList[OUTPUT_SPEED] = 0.0;
}
}

void CbehControlSpeed::Reset() {
    outputList.erase(OUTPUT_SPEED);
}
#include "behControlSpeed.h"
#include <math.h>

string CbehControlSpeed::GetName() {
    return "behControlSpeed";
}

CbehControlSpeed::CbehControlSpeed(int frequency, CPersonSearch &personSearch, CSegwayArbiter
* arbiter)
    : CBehavior(frequency)
    , m_personSearch(personSearch)
    , m_arbiter(arbiter) {
    m_speed = 0.0;

    m_pid.SetKp(1.0);
    m_pid.SetKi(0.0);
    m_pid.SetKd(0.0);
};

CbehControlSpeed::~CbehControlSpeed(void) {
}

void CbehControlSpeed::Execute() {
    if (m_personSearch.m_trackingInfo.IsTargetLocked) {
        double speed = 0.0;
        double mass = m_personSearch.m_trackingInfo.mass;
        //we only compute if mass is decreasing
        // (target is getting more distant from us)

        if (mass < 0) {
            double normTurnRate = abs(m_arbiter->m_currentTurn / m_arbiter->m_maxTurnDegS);
            speed = -(mass * (1 - normTurnRate));       //if turning, we reduce speed
        }

        outputList[OUTPUT_SPEED] = speed;
    } else {
        outputList[OUTPUT_SPEED] = 0.0;
    }
}

void CbehControlSpeed::Reset() {
    outputList.erase(OUTPUT_SPEED);
}
/* -------------------------------*/
* Thesis: Vision Based Person Following Using an Improved
* Image Segmentation Approach
*  
* Spring 2012
*  
* behControlSteering.cpp: Behavior implementation for robot steering. Steers
* the robot based on the last output from the pan controller.
*  
* Author: Brian Blaine, Padu Merlotti
** -------------------------------*/
#include "behControlSteering.h"
#include "Tracing.h"
string CbehControlSteering::GetName() {
    return "behControlSteering";
}
CbehControlSteering::CbehControlSteering(int frequency, CbehControlPan * behControlPan, int 
maxIntegration) : CBehavior(frequency), 
  m_behControlPan(behControlPan), m_maxIntegration 
  { maxIntegration } 
  m_turn = 0.0; 
}

CbehControlSteering::~CbehControlSteering(void) {
}

void CbehControlSteering::Execute() {

  //get last output from pan control
  double pan;

  if (m_behControlPan->outputList.find(OUTPUT_PAN) != m_behControlPan->outputList.end()) {
    pan = m_behControlPan->outputList[OUTPUT_PAN];
  } else {
    return;
  }

  //calculate the delta from current pan position to pan center
  double servoCenter = m_behControlPan->ServoCenter();

  double delta = pan - servoCenter;

  //average last N if available
  integrator.push_front(delta);

  if (integrator.size() > m_maxIntegration) {
    integrator.pop_back();
  }

  double avg = 0.0;
  double acc = 0.0;

  for (list < double >::iterator li = integrator.begin(); li != integrator.end(); li++)
    acc += (*li);

  avg = acc / m_maxIntegration;

  double turnScaled = (2.0 / (double)(m_behControlPan->m_upperLimit - m_behControlPan->
m_lowerLimit)) * avg;

  DO_TRACE(TraceLib::Segway, "behCS::Execute: pan=%.f, delta=%.f, avg=%.f, turnScaled=%.f", pan, 
delta, avg, turnScaled);
outputList[OUTPUT_TURN] = turnScaled;
}

void CbehControlSteering::Reset() {
    outputList.erase(OUTPUT_TURN);
}

#include <string>
#include <vector>
namespace TraceLib {
    enum TraceCategory {
        Algorithm,
        Tracking,
        ChainCode,
        Scoring,
        Listener,
        CameraControl,
class Tracing {
    static Tracing * s_tracer;

    std::vector<bool> m_enabledFlags;

    Tracing();
    ~Tracing();

    public:
    static Tracing * getTracer();

    void Trace(TraceCategory cat, const char * fmt, ...);
    void SetEnabled(TraceCategory cat, bool enabled);
    inline bool IsEnabled(TraceCategory cat) const {
        return m_enabledFlags[cat];
    }

    #include <string>
    #include <vector>
    
    static const char * TraceNames[] = {
        "Algorithm",
        "Tracking",
        "ChainCode",
        "Scoring",
        "Listener",
        "CameraControl",
        "PanTiltControl",
        "Performance",
        "Thresholding",
        "RegionGrowing",
        "TrackingState",
        "Segway",
        "Train",
        "Joystick",
        "PersonSearch",
        "Servo",
        "TraceCategoryLAST",
    };

    static const char * TraceNames() {
        return TraceNames;
    }
};

PanTiltControl,
Performance,
Thresholding,
RegionGrowing,
TrackingState,
Segway,
Train,
Joystick,
PersonSearch,
Servo,
TraceCategoryLAST,
};
/* ---------------------------------------------------------------------------
 * Thesis: Vision Based Person Following Using an Improved
 * Image Segmentation Approach
 * Spring 2012
 * Tracing.cpp: Implementation class for logging library. Logs statements
 * with the user defined logging level.
 * Author: Brian Blaine, Padu Merlotti
 ** ---------------------------------------------------------------------------*/
#define VC_EXTRALEAN
#define WIN32_LEAN_AND_MEAN
#include "Tracing.h"
#include <stdio.h>
#include <stdarg.h>
#include <windows.h>
define _CRTDBG_MAP_ALLOC
namespace TraceLib {
    Tracing::Tracing() {
        m_enabledFlags.resize((size_t)TraceCategoryLAST, false);
    }
    Tracing::~Tracing() {
    }
    Tracing * Tracing::getTracer() {
        if (!s_tracer) {
            s_tracer = new Tracing;
        }
        return s_tracer;
    }
}
void Tracing::Trace(TraceCategory cat, const char * fmt, ...) {
    if (!m_enabledFlags[(size_t)cat]) {
        return;
    }

    va_list args;

    va_start(args, fmt);

    char buf[8192];

    int numChars = sprintf_s(buf, sizeof(buf), "%-14.14s:", TraceNames[cat]);

    numChars += vsprintf_s(buf + numChars, sizeof(buf) - numChars, fmt, args);

    va_end(args);

    if (numChars > 0 && buf[numChars - 1] != '\n') {
        buf[numChars++] = '\n';
        buf[numChars++] = '\0';
    }

    OutputDebugString(buf);
}

void Tracing::SetEnabled(TraceCategory cat, bool enabled) {
    m_enabledFlags[cat] = enabled;
}

Tracing * Tracing::s_tracer = NULL;