1. The usage of such a program is pretty comprehensive. In medicine, there are many fields where such a program could be used in. Some of the potential applications:  
     
   • Medical diagnosis: Medical professionals can use an automated blood cell detection program to analyze blood samples and diagnose conditions like anemia, leukemia, and infections. This could help save time and improve accuracy in the diagnosis process.

• Blood transfusions: Blood banks could use such a program to quickly detect and count blood cells in donated blood samples, ensuring the blood is safe for transfusion.

• Research: In scientific research, automated blood cell detection could be used to study the effects of drugs, diseases, and genetic disorders on different blood cell types.

• Veterinary medicine: Automated blood cell detection could also be used in veterinary medicine to analyze the blood samples of animals and diagnose various conditions.

1. Python packages that were used during the project are:

Cv2, NumPy, Matplotlib, PySimpleGUI

FilterPy also was used at some point but in the final version there is no this package.

1. I've developed two separate scripts for the video and the photo parts (they share similar logic, but generally, they are two different codes).

Video code:

This code uses computer vision techniques to detect blood cells on a video.

The program first sets up a blob detector with some parameters to filter the detected cells based on color, circularity, convexity, inertia, and area (as cells are mostly circles but can be a little bit another shape) (the color is set to the color that the blood cells are painted in later in the code). It then opens the input video file and sets up the output video file with the same frame rate and dimensions.

In the main loop, the program reads each input video frame, resizes it to reduce processing time, and applies histogram equalization and color thresholding to increase contrast and detect blood cells with the blob detector. The detected cells are then marked in green circles with an ID number, and if a cell is too close to an existing cell, it is not added to the list of detected cells.

Finally, the program displays the resulting image and writes it to the output video file.

Photo code:

The code first loads an image, resizes it and creates a PySimpleGUI layout for an interactive GUI. The GUI contains sliders for adjusting contrast and removing noise, checkboxes for applying logarithmic or gamma transformations, and a checkbox for applying a morphology operation.

The code then enters a while loop to wait for user interaction with the GUI. When the user interacts with the GUI, the code retrieves the current slider and checkbox values, applies the necessary image processing operations, and displays the updated image in the GUI window.

The code then applies edge detection to highlight cell outlines, finds contours in the edge map, and draws the contours on the original image. It also computes the areas of the detected cells.

After the user closes the GUI window, the code displays a histogram of cell areas using Matplotlib.

1. I faced many problems during this project, and even though I decided to submit the project after the deadline to perfect it, I could not fix all of them until now.

My two main ideas of detection were blob detection and edge detection. However, later, I realized that these two methods are not perfect ways to determine blood cells. Blob detection is not best for determining objects that are often void inside. I tried editing the contrast of the picture so that the background would be a much different color than the other objects and painted the cells in some color that would be bright and easy to determine on the light background. After that, I planned to find objects of specific size and shape - blood cells. However, even though this method works, some cells appear undetected. Sometimes one other problem occurs with not perfect detection of blood cells. The blood cell caught in the first frame is sometimes not seen in the second frame, ruining the ID system I made. I tried using some methods to prevent this (for example, the Kalman filtering method that allows one to predict the object's position); however, I did not succeed.

Edge detection also did not appear to be the perfect method for cell detection. Too much noise disturbs the edge detection algorithm, leading to imperfect detection. I also tried editing contrast, but blob detection appeared to be the better way of detection. (However, still, I used edge detection on my photo code because due to the ability to edit contrast with an interface allows me to do the best result for edge detection (it is still not perfect))

1. In my opinion, the best way to improve my code is to try editing contrast even more dramatically so the blob detection would work better, and I still think that Kalman filtering is a great solution to smooth the imperfections of blob detection. I believe that it is the way to make the code perfect.