Introduction:

Artificial Intelligence has made a revolutionary progress in the last decade. It is creeping into our everyday life through technology like check-scanning machines and GPS navigation. One of the most important technologies that come under Artificial Intelligence is Computer Vision and that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and in general, deal with the extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

Problem Statement:

Detecting the respiratory rate of a person to find out whether the person has a problem is quite tricky task and needs special training and also special scientific gadgets. These gadgets are expensive. On the other hand, common people don't have any gadgets handy, which can detect the heart rate of a person when there is a dire need. People don't buy respiration detection kits as it is expensive and rarely needed. Hence if a person's respiration rate needs to be checked, he/she needs to be taken to the doctor, which is a very time consuming process.

Our Solution:

We propose a project with a goal to **estimate the frequency of a vibrating object from a video**. By modelling the movement of tracked features extracted frame by frame (e.g. corners), we can calculate the frequency at which these features are vibrating. We intend to convert this frequency into an audio signal and replicate the sound produced by the vibrating object at discrete time intervals. Extracting such a property can lead to applications in non-invasive respiratory rate estimation, human blood ow detection, and sound extraction from vibrating objects. Our project takes a more intuitive and simpler approach. By modeling the movement of tracked features (e.g. corners) on a video we can infer some important behavior of a rhythmic object, e.g, the frequency of a tuning fork or the respiratory/heart rate of a sleeping person. In the long run we hope this project will allow people to measure the heart rate easily by recording a video thus removing the need of any special equipment.

Our Method:

We track the rhythmic motions of interesting features frame by frame to determine the frequency of object of constant frequency in the video.

The first step is temporal filtering. We induce temporal filtering by using videos that were captured at a frame rate which is in the range of the frequency of the object (2x to approx 20x frequency of video). In case we have a video of frequency much higher than the frequency of the object, we don't consider all frames and skip a certain constant number of frames (n) which essentially reduces the frame rate by a factor of n. This temporal filtering is necessary to alter out higher frequencies. However, it is important to note that the frame rate of the video has to be at least twice the frequency of the object.

We use the Shi-Tomasi corner detector to decide which features to track. To track all features we use Lucas-Kanade Optical Flow algorithm. We group frames into windows and track features within these windows of frames. Also, these windows are contiguous and have an equal number of frames.

The motion of these feature points in the y -axis is in a rhythmic pattern of a frequency same as that of the object over different windows of frames. Hence, we record the position of features in the y-direction in a window and use it to compute the number of peaks for each feature. In a window, we determine the frequency of each feature using the number of peaks, frame rate, and window size as seen in Figure 1. For each window of frames we determine the mode frequency and show it as a "temporal result". Finally, a histogram containing all the frequencies estimated at each point is computed, showing the number of times a particular frequency value was assigned to a point. We alter out the points that do not belong to the objects movement. We assume that after temporal altering, the features what have higher displacement belong to the object of interest in the video. Hence, we alter out outliers that have low displacements.

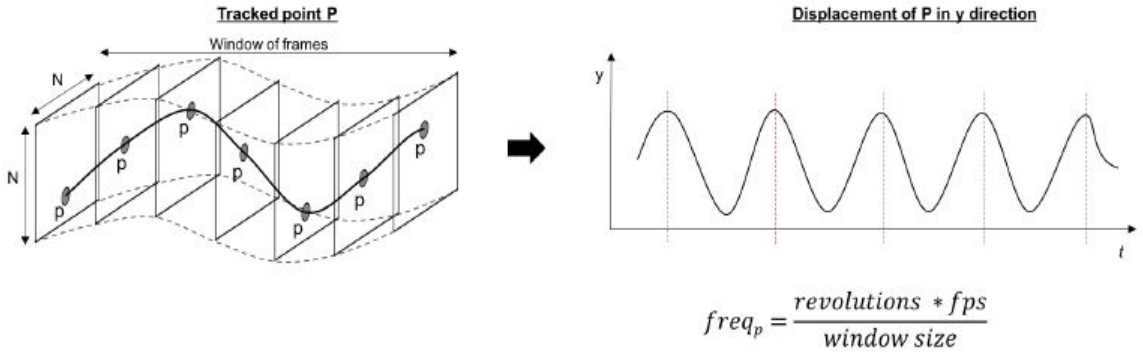


Figure 1: Tracking a point through a window and calculating its frequency

Finally, for every window, we calculate the frequency of object by a voting technique. This technique considers the frequencies of all features within the frame window and each feature votes for a frequency. A histogram is generated and the overall frequency of the object is determined by the mode.

System Components:

1. Input video: A normal video of any object whose frequency the user wants to monitor or a video of a person whose respiratory rate is to me measured.
2. Feature Extraction: Like Harris detector, we are planning to calculate the partial derivatives along x and y-axis and apply Eigen Decomposition to calculate the λmax and λmin. Then use greedy corner selection method to generate a list of interesting points.
3. Feature Tracking: To track all interesting points we will be using pyramidal representation of each frame where it is resized to its half recursively on the assumption that the pixel intensities of an object do not change between consecutive frames and neighboring pixels have similar motion.
4. Frequency Estimation: We will calculate the frequency of each interesting point by recording their revolutions in a window of **T** frames. To get accurate results, it is essential to filter out less important points that do not belong to the object's movement. We will remove these points depending on their revolution and only consider the frequencies of the remaining points. We will build and update a histogram of frequencies every **T** frames which will record all the estimated frequencies over all the video. The final frequency will be calculated by a voting a technique that considers all the frequencies ever estimated by the algorithm.

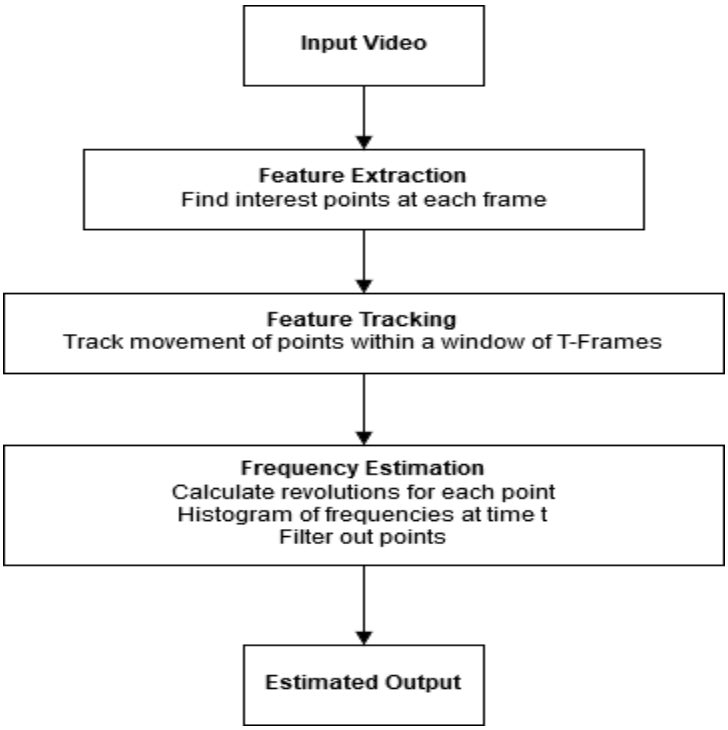


Figure 2: High level components of project

Feature Extraction:

In our approach, we only track interesting points and calculate frequency at these points. To extract these interesting points or features, we use the Shi-Tomasi corner detector developed by. This method made a modification of Harris Corner Detector and is known to give better results with optical ow algorithm which is explained in the next subsection(enter number here). Like Harris detector, the image is converted to grayscale and the local derivative is calculated at each pixel in Figure 3. To do so a window W over the pixel (x; y) is considered and moved by (u; v) for all pixels in the grayscale image. Then the sum of squared differences (SSD) between these two windows is used to approximate the partial derivatives along x axis and y axis as follows.

$$SSD = \begin{bmatrix} u & v \end{bmatrix} H \begin{bmatrix} u \\ v \end{bmatrix}$$

$$H = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$

On eigen-decomposition of matrix H, we get eigenvalues λ_{max} and λ_{min}.

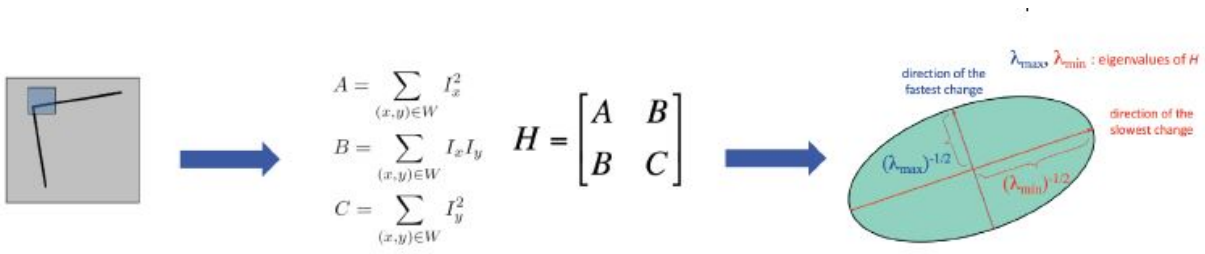


Figure 3: Corner Detection for extracting good features. Image adapted from D. Crandall slides.

The Shi-Tomasi corner detector differs from Harris after this point. The corners selected using a greedy method as follows:

1. Select points with λmin > threshold in list of possible interesting points, L1.

2. Sort the list of possible interesting points, L1, by λmin

3. Select point P with highest λmin in L1 and put it in list of final interesting points, L2, and remove all points in L1 within neighborhood of P

4. Continue till L1 is empty

Optical Flow:

In order to calculate the frequency at the interesting points we found previously, we need to find their location at each frame. To achieve this, we use the Lucas-Kanade tracking algorithm with Pyramidal representation proposed in.
Basically, this algorithm tries to find the new location v in image J of a point $u = [u_x\ u_y]^T$ from image I such that $v = u + d = [u_x + d_x\ u_y + d_y]^T$, where J(v) is "similar" to I(u). The vector d is also known as the optical ow vector, which describes the image movement at a point $x = [x\ y]^T$. The similarity is measured on a image neighborhood (integration window) of size $(2w_x + 1)(2w_y + 1)$. With all this being said, the vector d is defined as the vector that minimizes the following function:

$$\epsilon(d) = \epsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x,y) - J(x+d_x, y+d_y))^2 \quad (1)$$

In order to tackle possible tracking sensitivity to camera motion, changes of lighting or image size, proposed a pyramidal implementation for the original optical ow algorithm.
This pyramid representation is built in a recursive fashion, by resizing the image width and height in n levels, where at each iteration the height and width are divided in 2. This will allow the algorithm to handle large pixel motions, even larger than the integration window defined previously.
This algorithm works as follows: first, the optical ow for a point x is computed at the lowest level L_n, then this result is propagated to level L_{n-1}, and so on until you reach the original image (level L_0). In other words, equation (1) is calculated at every level. Then, the final d optical flow vector is defined by equation (2). This process can be seen in detail in Figure 4.

$$d = \sum_{L=0}^{L_n} 2^T d^T \quad (2)$$

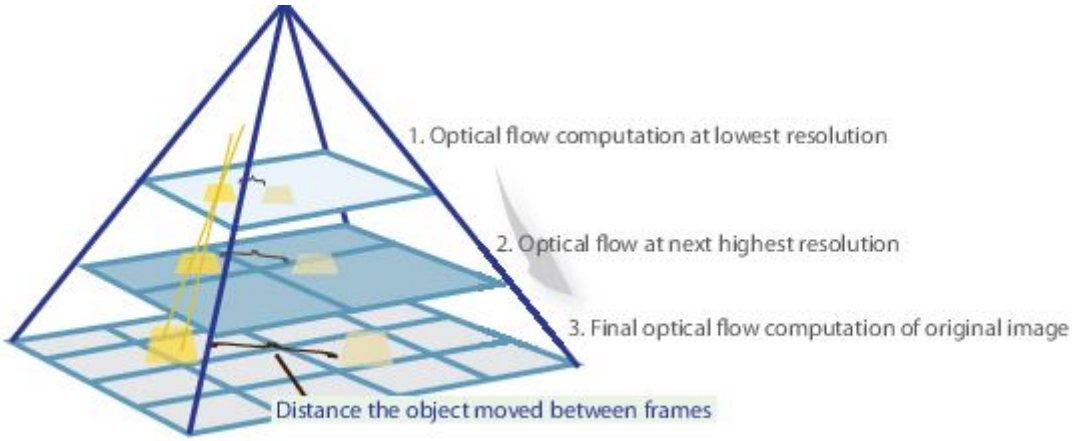


Figure 4: Pyramidal Representation of KLT algorithm. Image extracted from http://www.mathworks.com/help/vision/ref/vision.pointtracker-class.html

Special Proficiencies:

The special skills required for this project are

1) GUI skills: HTML , JavaScript , CSS, Java
2) Analytics and Algorithmic skills: Optical flow algorithm , Lucas-Kanade algorithm KLT algorithm
3) Database skills – SQL , MYSQL
4) Programming Skills: Java, Python, Android, iOS

Justification:

The project covers web, standalone and database technologies which cover all areas that would be in a project/ product. This should give the team enormous exposure to all facets of software technologies. We intend to use UML for designing the project components. This should give the team exposure to software design. Analyzing the various computer visual patterns would increase the team's domain expertise. Also this project is large enough that each member has to contribute. This will test the teams organizational, team work and social skills. The project also has the potential of generating revenue and could be a test of the team's entrepreneurial skills.

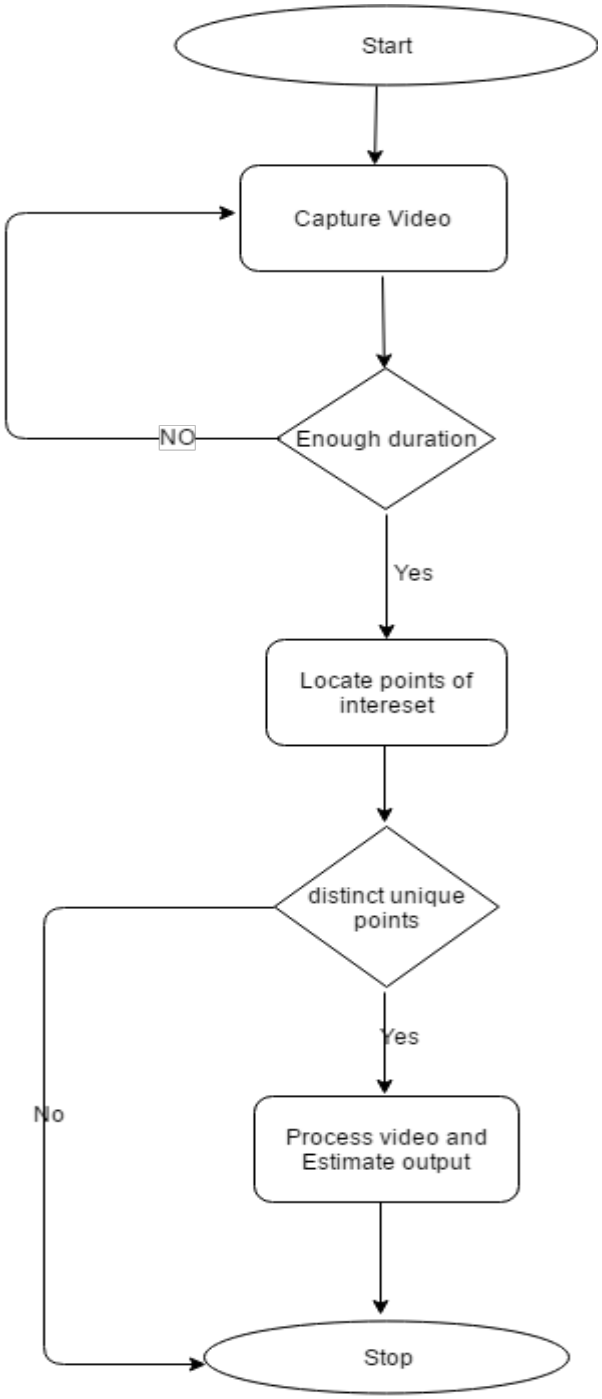
To know how it works, consider the most widely recognized approach to reproduce articles' movements is by building a 3-D demonstrate. Tragically, 3-D demonstrating is costly, and can be practically unthinkable for some items. While calculations exist to track movements in video and amplify them, there aren't ones that can dependably reproduce protests in obscure situations. The tool has many potential uses in engineering, entertainment, and more. For instance, in motion pictures it can be troublesome and costly to get CGI characters to sensibly communicate with their true surroundings. Doing as such obliges movie producers to utilize green-screens and make itemized models of virtual articles that can be synchronized with live exhibitions. However, with IDV, a videographer could take video of a current true environment and make some minor alters like concealing, tangling, and shading to accomplish a comparative impact in significantly less time — and at a small amount of the cost. Designers could likewise utilize the framework to reenact how an old building or scaffold would react to solid winds or a seismic tremor.

Project Documentation:

We will follow Agile methodology with a three-week development cycle. As part of documentation the team will create a requirement document, design document and Acceptance test plan. UML would be used for design. Database design would be done using entity relationship diagrams.

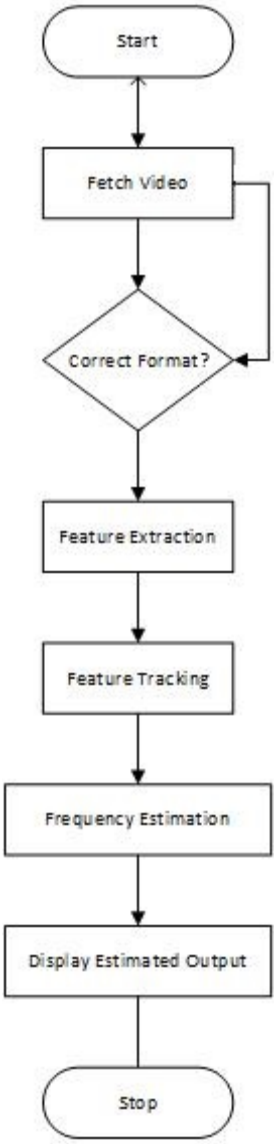
We plan on documenting each phase of project on a bi-weekly basis.

Data Flow Diagram:

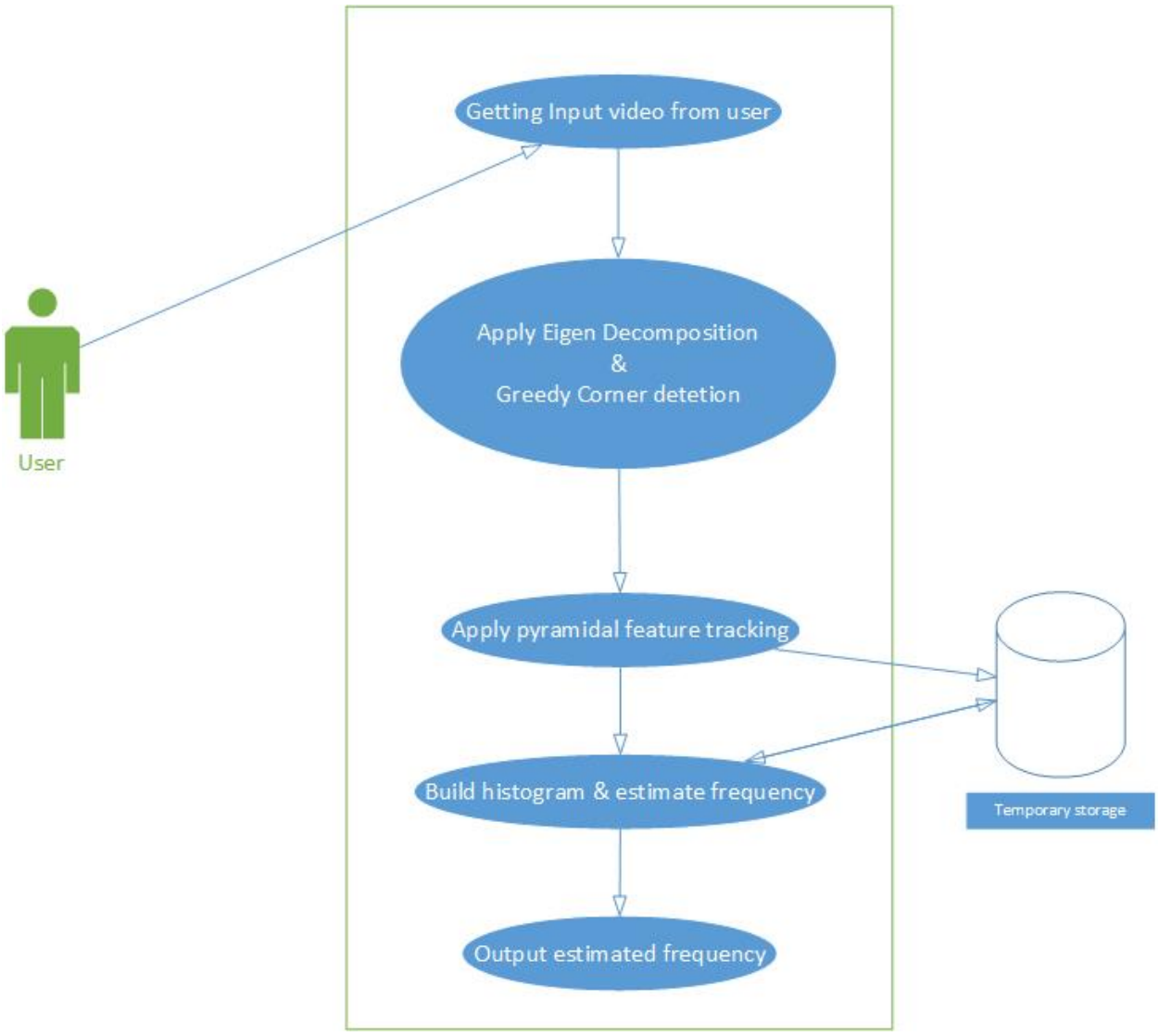


Data Flow

Flow Chart

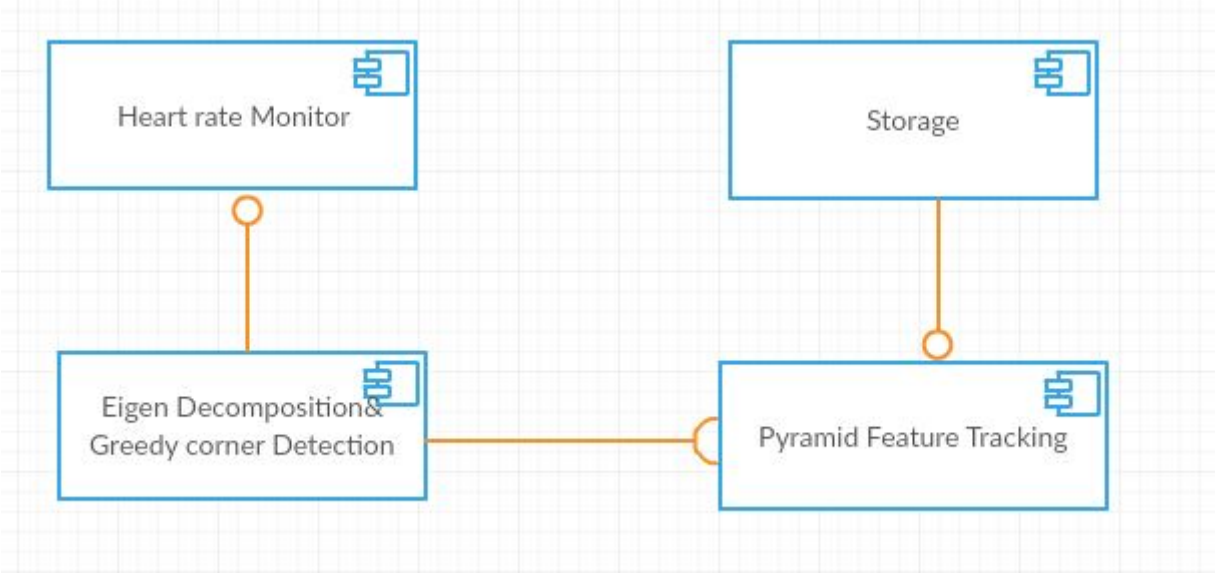

Start
Fetch Video
Correct Format?
Feature Extraction
Feature Tracking
Frequency Estimation
Display Estimated Output
Stop

Use Case Diagram



Activity Diagram

Component Diagram



Experiments and results:

In order to test our approach, we chose to deploy three different experiments: Tuning fork frequency estimation, Heart Rate estimation, and Respiratory rate estimation. Hopefully, these experiments will give us a notion of how accurate our system is and what are the constrains that have to be taken into account for each case scenario.

1. Tuning fork frequency estimation

We started with this particular experiment since tuning forks have a constant frequency along time, and it was easier for us to _ne tune our code. We borrowed some tuning forks of different frequencies from Jacobs School of Music and from the Physics Lab from Indiana University in order to build our own data set. Most of the videos where recorded with out cell phone's camera.
The frequency of the tuning fork should be at least half of the sampling frequency of the camera that is recording the video, otherwise we wont be able to see the tiny vibrations of the tuning fork. Figure 5 shows an example of a tuning fork of 125Hz and a frame rate of 1200 fps. This video was collected from YouTube and the tracked points are shown in green.

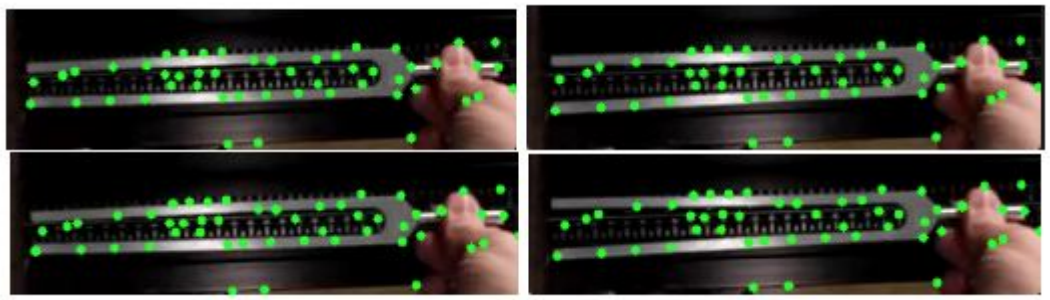


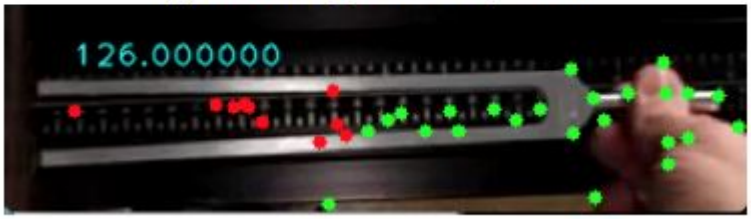
Figure 5: Video Sequence Output

As you can see, it is necessary to remove outliers, like the points on the person's hand, if not, the frequency of these points will be considered as well and will affect the final estimation. This is done by _filtering the points based on their displacement in the y direction. We only consider the point with a displacement greater than half of the maximum displacement ever recorded within a particular window frame. This points are then marked in red, as shown in figure 6(a). Because of the lack of high speed cameras, we were only able to record videos with our cell phones's camera, which have a max fps of 240 with the slow motion feature. This will restrict the amount of experiments possible with tuning forks, since the maximum frequency that we can detect is less that 125Hz.
We also tried to record a video of a slow motion video in order to augment the fps range, but it didn't show good results because of the uncertainty of the actual sample frequency of the video. The results we got for a video of a tuning fork of 125Hz, and 12000 fps are as shown in figure 6.

As you can see from figure 6(a), the first result we obtained was not as accurate as the second one. This is because at the beginning, all points are being considered for the estimation, thus, the points on the person's hand give a biased result. Later, these points are eliminated and more points that belong to the tuning fork are taking into account for the frequency estimation. Figure 7 shows the histogram of all the frequencies ever calculated for all the filtered points along the video. The window size we used for this experiment was 200 frames. The final

histogram showing all the values estimated by the system is shown in figure 7. The partial result we obtained at each frame window were: 120, 120, 126, 126,126, 126, 126, 126, 126. This means that windows were taken into account and for each of them, those values were the more prominent ones.
We tried the same experiment but with a video we recorded with the slow motion feature of our cell phone's camera and the results are shown in figure 8.
For this particular case, a little fine tuning took place, due to a sampling frequency limitation.

(a) First temporal result, 120Hz



(b) Last temporal result, 126Hz

Figure 6: Temporal outputs for video of a tuning fork of 125Hz and 1200 fps

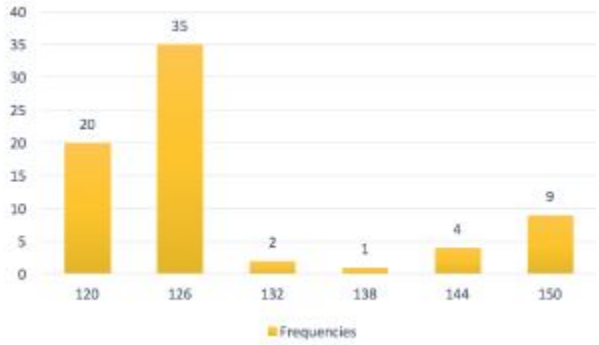


Figure 7: Histogram with temporal frequencies for tuning fork 1

The maximum frame rate we have in our cameras is of 240 fps, and the tuning fork is of 125Hz, which means that we need at least 250fps in order to capture the vibrations. Because of this, we used in our system as sampling frequency 250fps instead of 240. This gave us better results, but not as accurate as in the previous example. Figure 9 shows the final results we collected.
The window size for this experiment is the same as the previous example, 200. And the temporal frequency values are as follow: 117.5, 120, 118.75, 118.75, 118.75, 118.75.

(a) First temporal result, 117.5Hz (b) Last temporal result, 118.75Hz

Figure 8: Temporal outputs for video of a tuning fork of 125Hz and 240 fps

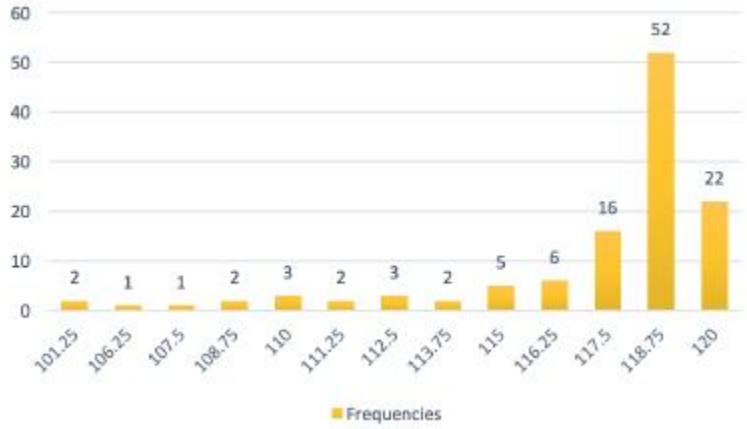


Figure 9: Histogram with temporal frequencies for tuning fork 2

2. Respiratory Rate Estimation

This experiment was about Respiratory Rate Estimation, how many breaths per minute a person takes. We thought of trying this because we wanted to try experiments which can work in conditions of lack of high speed camera and with real applications.
For this experiment we recorded a person sleeping and we measured the breath taken while sleeping. We used a blanket with enough details so it can be easily tracked.

The temporal filtering here was done with the time-lapse video of our cellphone which has a frame rate of 2fps. This is how we learnt the importance of temporal filtering and how to do it. We understood that the frame rate of the video should be within the range of the expected object's frequency.
Results are shown in figure 10.
The window size for this experiment is the same as the previous example, 30. And the temporal frequency values are as follow: 0.333333, 0.333333, 0.333333, 0.333333. Thus, 0:33 * 60 = 19:8 breaths per minute and the expected result was 21 breaths per minute.
The final histogram is shown in figure 11.
This could be applied for baby cam, for example, as a way of monitoring babies in a non-intrusive way.

Figure 10: Final frequency of 0.33Hz for respiratory rate

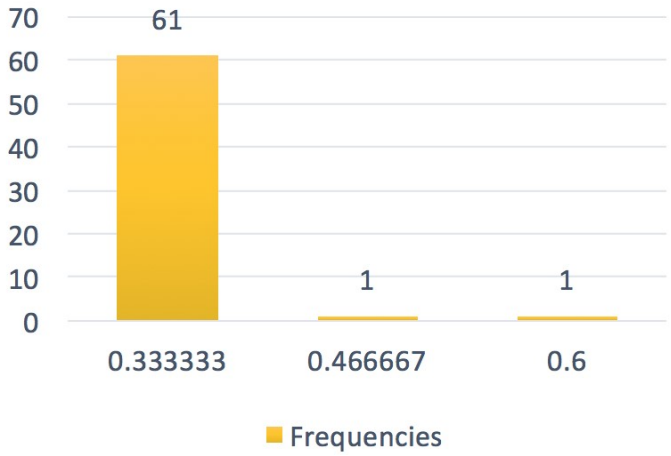


Figure 11: Histogram with temporal frequencies for Respiratory Rate

3. Heart Rate Estimation

Finally, in order to estimate the heart rate of a person based on the involuntary movement of the head when the heart pumps blood to our brain, we used the same system. We took a close up video of a face. With a close up video of a face we also took care of feature outliers that were not on the face. The only difference for this experiment in comparison with the Tuning fork one, was that in order to filter points temporarily, we considered a frame every 5 frames. This is because the video frame rate has to be within the range of the expected frequency, so instead of having a 30 fps video, we lowered that value to 6 fps by only calculating the peaks every 5 frames. The results are shown in figure 12.

The window size for this experiment was 50. And the temporal frequency values are as follow: 1.56, 1.44, 1.56, 1.56, 1.56, 1.44, 1.44, 1.44, 1.44, 1.44, 1.44, 1.44. The final histogram is shown in figure 13.

The expected result for this experiment was a heart rate of 90 beats per minute, so if we multiply our estimated frequency 1:44 * 60 = 86:4 bpm. We also tried taking pulse by taking a video of the veins in our hands. Since we could not make the veins evident enough we couldn't detect any movement nor corners. Additionally, we tried to take a video of the veins in our necks but that experiment failed because, again, no corner points.

We think that as a future work, we could, instead of trying to detect corners, just detect faces and apply our algorithm to the bounding box of the face or just detect face features so we do not need to filter any points or capture the person's face only.

Figure 12: Temporal outputs for Heart Rate estimation

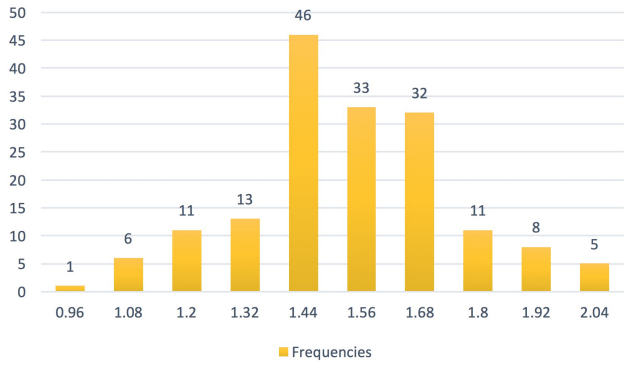


Figure 13: Histogram with temporal frequencies for Heart Rate

Conclusions:

As a conclusion of this project we can say that we successfully achieved the expected results and we replicated experiments proposed in papers with much less equipment, although some constrains.
Our project uses a very intuitive way for calculating the frequencies and it works with scenes that are not that controlled. We didn't use any kind of tripod, o a very controlled environment.

Code Usage:

We used the OpenCV library version 2.4.12. To compile the code just run make.
To run experiments, the code expects the following parameters:
./tracking <video-path> <frame-rate> <frame-window> (optional, default = 100)
If you want to skip frames, like we did in section 5.3, you have to remove the comments of a few lines from line 146 in the code.
If you want to replicate the sound of the tuning fork, you have to save to console's output in a text file and make sure that the Matlab script take that file as input; yes, Matlab is required to run the script.
Example: ./tracking tuning-fork.avi 250 > output.txt
Please notice that if we want to output the video with the tracked point and temporal frequency values, the code starts running very slow. We activate or deactivate this function by setting the variable save output video value in 1 or 0 respectively.

Following are the project deliverable dates

S No.	Project phase	Deadline
1	Requirements Gathering/Analysis	February 3rd 2017
2	Proposal submission	February 14th 2017
2	High Level design	February 17th 2017
3	Development	March 17th 2017
4	Unit testing	March 24th 2017
5	System testing	April 07th 2017
6	Release date	April 18th 2017

Reference materials:

[1] G. Balakrishnan, F. Durand, and J. Guttag. Detecting pulse from head motions in video. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3430{3437, 2013.

[2] J.-Y. Bouguet. Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm. Intel Corporation, 5(1-10):4, 2001.

[3] A. Davis, M. Rubinstein, N. Wadhwa, G. Mysore, F. Durand, and W. T. Freeman. The visual microphone: Passive recovery of sound from video. ACM Transactions on Graphics (Proc. SIGGRAPH), 33(4):79:1{79:10, 2014.

[4] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In IJCAI, volume 81, pages 674{679, 1981.

[5] J. Shi and C. Tomasi. Good features to track. In Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on, pages 593{600. IEEE, 1994.

[6] H.-Y. Wu, M. Rubinstein, E. Shih, J. V. Guttag, F. Durand, and W. T. Freeman. Eulerian video magnification for revealing subtle changes in the world. 2012.

[7] http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0081219