STUDY ON OPTICAL PROPERTIES AND STABILITY OF GOLD NANOSTARS FOR LOCALIZED SURFACE PLASMON RESONANCE (LSPR) BASED BIOSENSING APPLICATIONS

Rahul Kumar

A Thesis

In

The Department

Of

Mechanical, Industrial, & Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science (Mechanical Engineering) at

Concordia University

Montreal, Quebec, Canada

July 2018

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: ___Rahul Kumar___________________________________

Entitled: ___Study on Optical Properties and Stability of the Gold Nanostars for Localized Surface Plasmon Resonance (LSPR) based Biosensing Applications__________

and submitted in partial fulfillment of the requirements for the degree of

_____Master of Applied Science (Mechanical Engineering)___________

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____________Dr. Gerard Gouw___________________	Chair
_____________Dr. Ramin Sedaghati ________________	Examiner
_______________Dr. A.Bagchi______________________	Examiner
_________Dr. Muthukumaran Packirisamy____________	Supervisor

Approved by ____
Chair of Department or Graduate Program Director

____
Dean of Faculty

Date ____July 30, 2018____

ABSTRACT

Study on Optical Properties and Stability of Gold Nanostars for Localized Surface Plasmon Resonance (LSPR) based Biosensing Applications

by

Rahul Kumar

Masters of Applied Science in Mechanical and Industrial Engineering

Concordia University, Montreal, QC

Exosomes are small vesicles (typically 30-120 nm), formed through the inward budding of endocytic compartments and secreted through fusion of these vesicle-containing endosomes with the plasma membrane. Increasing evidence suggests that exosomes play an important role in cell-to-cell communication through the transport and delivery of cellular components such as lipids, proteins, and nucleic acids. Exosomes preferably accumulate at solid tumor sites due to leaky vasculature and abnormal lymphatic drainage, making them an attractive candidate for detecting cancer.

In recent years, nanoparticles have paved pathway to detect exosomes by incorporating of targeting functionality onto nanoparticle surfaces of the nanoparticles. Among various nanoparticles, Gold nanostars possess interesting tunable properties that can be exploited in different nanomedicine applications including drug delivery systems, thermal-ablation, and image contrast agents.

As nanoparticle properties are directly related to their size and shape, it is a fundamental criterion to ensure high control and precision during the synthesis to obtain anisotropic nanoparticles with desired properties. Gold nanostars were synthesized by seed-mediated method using biocompatible capping agents to control and stabilize nanoparticle morphology during the reactions. First step was to obtain small gold nanoparticles that served as seeds for the growth of branches to finally obtain nanoparticles with the desired star-shape. Gold seeds were obtained by chemical reaction method incorporating citrate as capping agent; monodisperse colloidal nanoparticles (30 ± 5 nm core diameter) were efficiently obtained. Characterization showed that gold seeds possessed a well-defined spherical structure. Silver nitrate was added to the growth solution which acts as a catalyst to activate the site for the formation of branches.

However, in most cases these nanostars possess poor long-term stability, short branch length, polydispersity and suffer from aggregation. In order to address these issues, this thesis focuses characterization of physical properties such as size and morphology using numerical and experimental methods to enhance the synthesis and stabilization of gold nanostars for Localized Surface Plasmon Resonance (LSPR) based biosensing of exosomes. The results show a highly sensitive biosensing platform with prolonged shelf life of more than a year. Biomolecule such as Streptavidin, Biotin, PEGylated (PEG) and Vn peptide (Vn96) were used as surface functionalization molecules on gold nanostars for detecting exosomes. The results show great promise in comparison to standard spherical nanoparticles.

ACKNOWLEDGEMENTS

I would like to convey my gratitude, first and foremost, to my advisor Professor Muthukumaran Packirisamy for engaging me in his stimulating and first-rate Optical-Bio Microsystems group. I am honored to have worked under such an illustrious scientist, and for participating in a discipline with so many diverse and fascinating elements. Dr. Packirisamy has guided and mentored me throughout numerous exciting projects at the highest standards of academic research. I am confident that he has impressed on me his keen and exhaustive approach to achieving a commendable caliber of output.

Another person to whom I am immensely indebted is Dr. Simona Badilescu, the head supervisor in the lab. Dr. Badilescu has been instrumental at every milestone of my scientific progress. She has co-authored each of my papers and has been unfailingly patient and helpful with me throughout these seminal graduate years. Her formidable expertise and gumption have been indispensible to each undertaking.

I thank Atlantic Cancer Research Institute, Moncton, NB, Canada for supply of Vn96 and MCF 7 exosome. I must also express my appreciation for the incessant assistance and camaraderie of all my colleagues in the Optical Bio-MEMS lab group.

Lastly, I would like to extend my thanks to those in my personal sphere of life who have provided me with much peripheral support in my academic activities, namely my mother, Sangeeta Kumar, my father, Arvind Kumar, and my brother, Major. Nitin Kumar.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

A	The linear coefficient of absorption
B	The two-photon absorption coefficient
E	Electric field
$\mathbf{E_T}$	Transmitted energy
$\mathbf{E_0}$	Induced Energy
N	Number density of electrons
P	Polarization
S	Surface area of nanoparticle
V	Volume
h	Planck’s constant
$\mathbf{k_B}$	Boltzmann’s constant
m	Effective mass of an electron nm – nanometer or 10^{-9} m
$\mathbf{\Gamma}$	Collision frequency of electrons
$\boldsymbol{\delta}$	Size of edge grid at phase interface
$\mathbf{\Delta}$	Grid size for bulk domain
$\boldsymbol{\varepsilon}$	Dielectric function
$\boldsymbol{\varepsilon_0}$	Permittivity of vacuum
$\boldsymbol{\mu}$	Permeability of the material
$\boldsymbol{\sigma}$	Sensitivity

ACRONYMS

ABC	Absorbing boundary conditions
AuNSs	Gold nanostars
AuNP	Gold nanoparticles
B	Branch model of a nanostar
CTAB	Acetyl trimethylammonium bromide, used in synthesis of nanoparticles
EM	Electromagnetic
NIR	Near Infrared
NP	Nanoparticle or particles of size in the 1-200 nm range
NS	Nanostar
OL	Optical limiter or limiting
PML	Perfectly matched layer
PDMS	Polydimethylsiloxane is a polymeric organo-silicon compound
S	Spherical core model of a nanostar
S1B	NS model of sphere & one branch
S2B	NS model of sphere & two branches SHO – single harmonic oscillator
SPR	Surface plasmon resonance
LSPR	Longitudinal SPR
TSPR	Transverse SPR
VIS	Visible range electromagnetic radiation: 390 - 770nm wavelength

CHAPTER 1

Introduction

1.1 Introduction

The definition of the cancer has evolved in the last decades. The complexity of the cells interaction creates a unique microenvironment, contributing to the cancer development. Understanding these cell interactions can be a key fact to developing an early stage cancer detection tools. The recent development of nanoscale science and technology gave rise to the development of such tools. With their help very low amounts of biomolecules were detected, analyzed, mimicked and even externally manipulated.

A nanoparticle is defined as "a particle with all three external dimensions in the nanoscale" [1]. Potentially, nanoparticles have applications in a wide range of fields, such as medicine engineering, energy and many more [2, 3]. Within this medical field, a huge range of different nanoparticles are either studied or either currently used as a standard of care.

A main division within this field of nanoparticles can be made between organic and inorganic nanoparticles. Organic nanoparticles consist of biological molecules where the most studied ones are bilipid layered structures called micelles. Inorganic nanoparticles comprise a huge variety of nanoparticles such as quantum dots, iron oxide, silica, gold nanoparticles etc.

Gold nanoparticles (AuNPs) which are capable of hosting localized oscillations of conduction band electrons, called plasmons, are very suitable to perform as an active component for such tunable hybrid system. AuNPs are chemically rather inert so the chemical properties of molecules attached to them are preserved. On the other hand, due to the special optical characteristics of AuNPs emerging from plasmons, the field distribution around them can be changed and therefore nuance optical properties of molecules situated nearby.

Surface plasmons are electromagnetic waves transmitted across the juncture of metal and dielectric environments. Due to the sensitivity of this effect to a slight electromagnetic trigger, this phenomenon is constructively employed as a method of sensing label-free biological entities [4]. In a variation of this method, one can introduce nanoparticles to host the conduction electron oscillations induced by the incident light, and use this as a biosensing technique. Such technologies have developed into what is known as localized plasmon surface resonance (LSPR). The smaller decay length of SPR versus LSPR is illustrated in the Fig 1.1. LSPR works by recording the wavelength at which an absorption of the incident light is caused by a certain resonant frequency that is associated with the dielectric environment. Such an approach has many benefits, such as:

- The optical hardware needed for LSPR is much less complex since no prism is needed to couple the light, so the instrument can be made smaller and more affordable.
- Since the angle is not important, the instrument is much more robust against vibration and mechanical noise.
- LSPR is not as sensitive to bulk refractive index changes, which causes errors in experimental data, because it has a much shorter electromagnetic field decay length.
- No strict temperature control is needed, simplifying the instrument.
- The sensor chips can be manufactured at a much more affordable price.
- Easier to use and maintain.

This smaller sensing volume means that LSPR is more sensitive to molecular binding and less sensitive to bulk effects. The smaller decay length and sensitivity associated with LSPR reduces artifacts caused by external variables such as temperature drift or buffer refractive index changes.

Also, since only a portion of the absorbed energy of the incident light remains as light in the form of scattering, and the remainder is dissipated by Landau damping [4], there will be a transduction of light to heat. This local thermal source has many potential applications, for example in catalysis or phase transition [5].

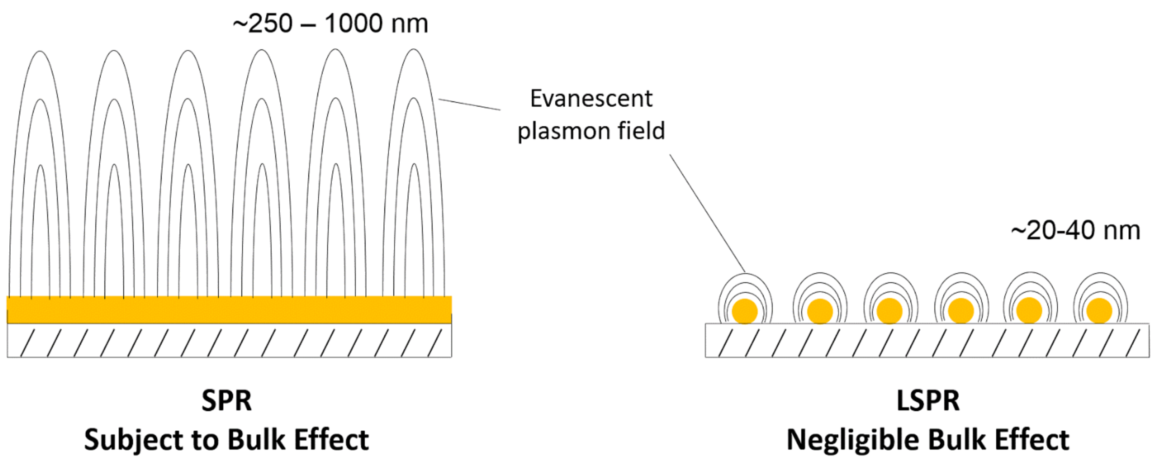


Fig 1.1: Schematic representing the difference between the principles of traditional SPR. and LSPR [4].

All these types of nanoparticles have their advantages and disadvantages to use them in the medical field [6, 7]. Anisotropic AuNPs are of special interest due to their structural, optical and catalytic properties, which are different and most often superior to those of spherical AuNPs.

For example, the high electric field enhancements that anisotropic nanoparticles (rods, cubes, prisms or stars, among others) present at sharp edges and tips, render them extremely attractive as plasmonic enhancers for surface enhanced Raman scattering. Among many existing anisotropic gold nanostructures star-shaped nanoparticles (gold nanostars) have achieved a huge interest. These novel nanoplatforms have numerous applications in various fields, mainly due to their plasmon tunability into the near infrared (NIR) region and the multiple hot-spots generated at their branches as shown in Fig 1.2.

Fig 1.2: 3D Representation of Gold Nanostar.

In this context, this thesis focuses on gold nanostars as functional plasmonic nanoparticles for biosensing application. **Chapter 2** presents simulation, analysis, and implementation of morphology tuning of gold nanostars structures. The gold nanostar was simulated using 3D Finite-Difference Time-Domain (FDTD) techniques to investigate the effect of morphological changes on the localized surface plasmon resonance (LSPR) properties. The dependence of the size and the branch dimension of the nanostar on the LSPR properties were analyzed in the simulation. Four model parameters have been modified to investigate the sensitivity of LSPR detection: (a) the wavelength of incident light, (b) the diameter of the core of nanostar, (c) the length of the branch of the nanostar, and (d) the thickness of the branch of the nanostar. The simulation results show that the variation in the branch dimension influences the LSPR absorption measurement. The FDTD simulation can be a useful tool for designing an LSPR nanoparticle biosensor. **Chapter 3** is study of morphology and stability of gold nanostars were investigated under different pH environment. The surface morphologies were observed on glass substrates using ultraviolet and visible (UV-vis) spectroscopy, and SEM. It is found that gold nanostars can be readily stabilized just by adjusting the initial pH conditions of growth solution. The particle size distribution of gold nanostars under different pH environment has been investigated using UV-vis spectroscopy and found to be highly dependent on pH. At an optimal pH of 11, gold nanostars obtained are highly monodisperse, longer branch length and have surface plasmon at 720 nm. For other pH conditions, particles are non-uniform and polydisperse, showing a red-shift in plasmon peak due to aggregation and large particle size distribution. Through time-dependent UV-Vis spectra studies hypothesize the pH-dependent stabilization mechanism wherein the formation and stabilization of AuNS solution were affected greatly by the intarmolecular aggregation induced by pH of the growth solution. The information obtained in this study can be used to design stable gold nanostars with longer shelf life for biosensing applications. **Chapter 4** focuses on the effects of sonication on the size, morphology, and stability of gold nanostars was investigated for the first time. It was found that the seed-mediated method, followed by the sonication treatment developed in this study, offers new opportunities to synthesize aqueous suspensions of monodispersed gold nanostars with prolonged stability. The results indicate that at a sonication frequency of 25 kHz, the maximum average nanostars size of 120 nm is attained. Increasing the sonication frequency, the average size of nanostars decreased significantly and, consequently, the position of the Localized Surface Plasmon Resonance (LSPR) band, could be easily tuned. It is thought that the change in the morphology is due to the collapse of the cavitation bubble which generates extreme conditions of pressure and temperature. Cavitation images in the gold nanostars solution were for the first time extracted by using an original software. It is found that the gold nanostars, after ultrasound treatment, are substantially more stable because their aggregation becomes much slower. The use of the sonication technique to stabilize gold nanostars, without any additional chemical stabilizer, is both technologically and scientifically important. It can be expected that by using this technique, to be able to produce large volumes of consistent quality, stable gold nanostars. **Chapter 5** is dedicated to study the effects of sonication, temperature and immobilization of gold nanostars on the polymer to develop a sensing platform. The morphology and the spectral characteristics of the embedded nanostars have been investigated. Our results show that the nanostars aggregates are stable for several months when embedded in the poly (vinyl alcohol) matrix. In the original aqueous solution, nanostars, without sonication, revert to the spherical morphology in 1-2 weeks. The results show that the stabilization of nanostars occurs principally through aggregation. Aggregation minimizes their surface energy, due to the strong van der Waals forces acting between the individual nanostars. Embedding in the polymer matrix is assisted by the presence of hydrated protons (hydronium ions) that result from the sonochemical reduction of water molecules. The refractive index sensitivity tests performed using Localized Surface Plasmon Resonance (LSPR) of AuNS-polymer composite films demonstrated high sensitivity of gold nanostars aggregates in sensing the surrounding environment. Indeed, the refractive index sensitivity is found to be in the range of 550-600 nm/RIU, compared to 100 nm/ RIU found previously for the PDMS Gold nanospheres composite. **Chapter 6** is dedicated to the unique properties of Gold nanostars (AuNSs) modulated through surface functionalization, for biosensing of MCF-7 exosomes in investigated in this paper. Here, a study of AuNSs-surface-functionalized with streptavidin-biotinylated PEG-Vn96, focusing the targeting of MCF 7 exosomes is carried out. The behavior of modified AuNSs, introduced in a PVA/PDMS composite has been investigated and the critical role of biotinylated PEG-Vn96 in targeting the MCF 7 exosomes is studied for the first time with the help of localized surface plasmon resonance (LSPR) spectrum Polydimethylsiloxane (PDMS) is a dominant material in the fabrication of microfluidic devices. However, its key property of hydrophobicity has hindered its use in microfluidics, which requires the channels to have hydrophilic surface properties. We developed, optimized, and characterized a method to produce PDMS with a hydrophilic surface via the deposition of PVA mixed with NaOH and demonstrated its suitability for droplet generation. The proposed method is simple, quick, effective, and inexpensive and allows the device to be selectively patterned with both hydrophilic and hydrophobic regions. The thorough understanding of the fundamental properties of bioconjugated AuNSs is of great importance for the design of highly sensitive and reliable functionalized AuNSs-based biosensor.

CHAPTER 2

Modeling The Effects of Morphology on Plasmonic Behaviour of Gold Nanostars

2.1 Introduction

Seeking architectural building blocks with controlled morphology and property is a significant challenge for future nanodevices and applications [8]. The localized surface plasmon resonance (LSPR) of noble metal nanoparticles, originated from the collective oscillations of conduction electrons, are strongly determined by the size, shape, composition, and the surrounding environment [9]. Gold nanostars are not as highly monodisperse as those with other forms. The LSPR in gold nanostars could introduce tunable plasmon peaks from visible to the near-infrared region by tuning the shape and branch length [10]. Gold nanostars are considered as a solid core with protruding prolate tips, reflecting the multiple bands in their spectra. In some cases, well-defined peaks in the ensemble spectra can be observed as shown in Fig 2.1 [11 – 14]. With different tip geometries and asymmetric morphology, one might expect branched nanoparticles to have complicated LSPR spectral features that are lost in ensemble measurements. But due to the asymmetric shape, the low Raman signals can be enhanced when the analyte molecules are placed in proximity to the surface of a plasmonic nanostructured material or located in so-called hot spots, i.e., locations such as edges, vertices, tips, or intermetallic junctions, where the local electric field is expected to be the highest [15].

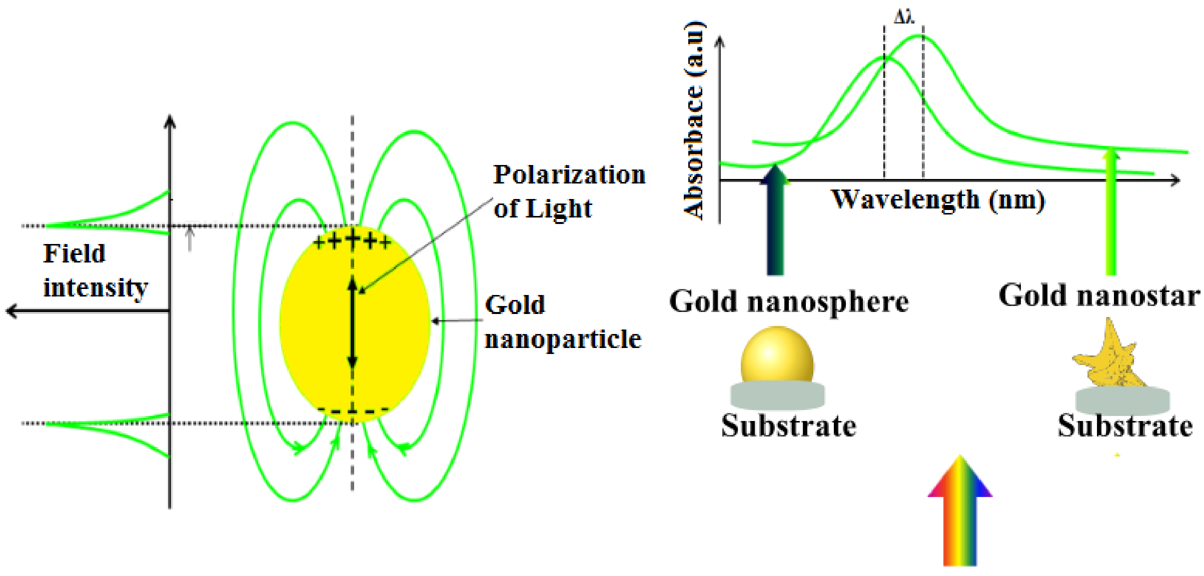


Fig 2.1: Schematic of localized surface plasmon resonance and localized surface plasmon resonance sensor [13].

In this paper, we investigated the LSPR of gold nanostars corresponding to the unique morphology. The optical response of gold nanostars (AuNSs) - PDMS composite to an incident light wave is determined by the electrons of the metallic particles embedded in the polymer. More specifically, the electric field of the incident light excites the valence electrons setting them collectively into a coherent oscillation, localized at the surface of the particle. At resonance, a maximal amount of energy is transferred, which is noticed as a local maximum of an absorption spectrum and is known as the localized Surface Plasmon Resonance (LSPR). A spherical particle has a single dipolar resonant mode, but particles with shapes of lower order symmetry, like rods and cones, may have several dipolar modes. To synthesize AuNS, we adopted a seed-mediated growth method. Spherical Au nanoparticles (AuNP) with a mean diameter of 40 nm were prepared as a seed and dispersed in an Au precursor ($HAuCl_4$) solution. Star-shaped Au nanoparticles with multiple branches were formed when ascorbic acid and silver nitrate were injected into the seed solution. We further controlled the overall size of AuNS by changing the molar concentration ratio between Au^{3+} and AuNP seeds

([Au^{3+}]/[AuNP]). Higher [Au^{3+}]/[AuNP] ratios led to the synthesis of larger AuNS as shown in Fig 2.2. To simulate the optical properties of nanostars, their complex structures were deconstructed into solids of simpler shapes that were numerically modeled by using the FDTD method. FDTD solves Maxwell's equations repeatedly along a time evolving spatial grid. This will yield the information about the far-field extinction and the near-field enhancements at the nanoparticle surface. FDTD simulations were carried out on a model after an individual nanostar [16, 17]. LSPR of complex structures such as these branched gold nanoparticles can be understood regarding the plasmon hybridization model [18]. Thus the nanostars can be broken down into the spherical core and the elongated tips. The numerical analysis aimed to reveal the contribution of each such elemental component to the overall optical response of the AuNS.

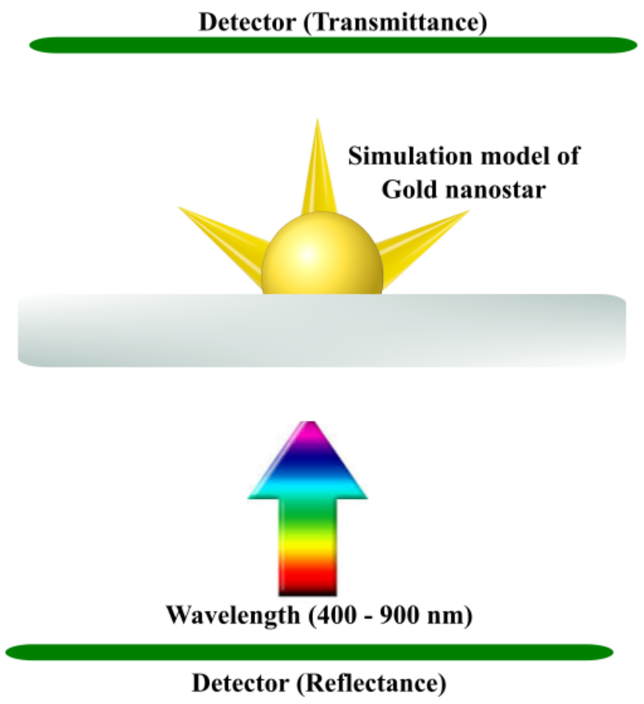


Fig 2.2: Schematic of Gold Nanostar LSPR Simulation Condition Diagram [18].

2.2 Numerical and Experimental Techniques

2.2.1 Numerical Modelling of Gold Nanostar

To start with mathematical calculations, assume a quasistatic approach using a conical branch morphology. In a conical shape, both major and minor radii and their aspect ratio are very important in analyzing the structure; therefore, the exact analytical electromagnetic solution of cone is discussed using numerical solutions of Maxwell's equations. The cone is irradiated by a polarized light in z-direction, in the visible wavelength range as seen in Fig 2.2. This electrostatic approximation is valid only when $\boldsymbol{\epsilon_b} = \left(\boldsymbol{2\pi d}/\boldsymbol{\lambda}\right) \ll \mathbf{1}$ [19], where $\boldsymbol{\epsilon_b}$ is the electric permittivity of the background medium, λ is the incident light wavelength and d is, the average diameter of the particle. Localized surface plasmon resonance (LSPR) is due to the surface plasmon waves produced from the collective oscillations of electrons by illuminating the nanoparticles. The oscillation of electrons in metal is described by the dielectric function as by the Drude-Lorentz [19] model as:

$$\boldsymbol{\epsilon(\omega)} = \boldsymbol{\epsilon_\infty} - \frac{\boldsymbol{\beta} - \boldsymbol{\epsilon_\infty}}{\mathbf{1} + \boldsymbol{i\varphi\vartheta}} + \frac{\boldsymbol{\delta}}{\boldsymbol{i\varphi\epsilon_b}} \qquad \textbf{[Eq 2.1]}$$

Where $\boldsymbol{\epsilon_\infty, \beta, \vartheta, \varphi, i, \delta}$ represent infinite frequency permittivity, static permittivity, relaxation time, free space permittivity, and angular frequency, conductivity, respectively. The infinite frequency permittivity and conductivity must meet the conditions $\boldsymbol{\epsilon_\infty} \geq 1$ and $\boldsymbol{\delta} > \frac{\boldsymbol{\epsilon_b(\epsilon_\infty - \beta)}}{\boldsymbol{\vartheta_0}}$ if $\boldsymbol{\beta} < \boldsymbol{\epsilon_\infty}$ [20].

Another model is called the extended Debye model [21]:

$$\boldsymbol{\epsilon(\omega) = 1 + \sum_{k=1}^{6} \frac{\Delta}{\partial\vartheta^2 + i\varepsilon\vartheta + \in}} \qquad \textbf{[Eq 2.2]}$$

Where $\Delta, \boldsymbol{\partial}, \boldsymbol{\varepsilon}, \in$ are constants that provide the best fit for various metals (e.g., gold and silver).

2.2.2 3D FDTD with non-uniform mesh

In this study, Rsoft software was applied for the FDTD simulation. The primary challenge with simulating gold nanoparticles in 3-D is that due to the lossy and dispersive property of gold, ultra-small grid size has to be used to mesh the gold to guarantee numerical stability. Generally speaking, the grid size should be smaller than one hundredth of the wavelength. As a result, the total amount of the stored field's data on these finely meshed grids is huge and can easily exceed the available amount of memory. Such a drawback can be overcome by using a non-uniform mesh configuration. Since gold nanoparticles are very small and that occupy a very limited space, the simulation region could be meshed in such a way that, gold and its vicinity are meshed with ultrafine grids, while other areas are meshed with more coarse grids. A meshing configuration of a gold nanostars is shown in Fig 2.3.

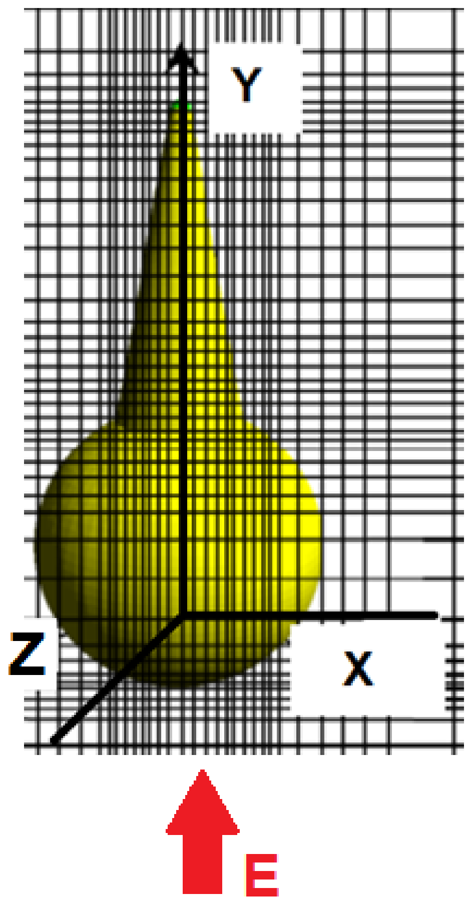


Fig 2.3: Schematic showing the principle of the non-uniform grid used to compute the properties of one-branch nanostar 3D-model. The grid used was much finer, but it has been coarsened for clarity. The nanostructure is excited by a wavelength that is emitted from the launch pad, drawn in blue color. The monitor is shown as the black line rectangle and the Perfectly matched layer as the outer gray strip.

The region that the gold nanostar occupies is meshed by 1.25 nm grids. It should be noted, that, the regions connected to the finely meshed region are meshed with gradually varying grid sizes to guarantee the numerical stability. Such a non-uniform mesh is very effective in performing the simulations. A comparison of different mesh configurations used in simulating gold nanostar is given in Fig 2.4. It can be observed that as the mesh is refined, the memory required for the non-uniform mesh goes up slightly.

The stability factor of FDTD, γ_{FDTD} is defined as [23]

$$\gamma_{FDTD} = \frac{c\delta t}{n} \qquad \textbf{[Eq 2.3]}$$

Where c is the speed of light in vacuum, δt is the time step and n is the smallest grid size. In all the simulations presented in this thesis, γ_{FDTD} is set to be equal to 0.25, which gives very good numerical stability.

Once the cell size has been chosen, the time step is being chosen by the RSoft software automatically, according to stability considerations.

The two modes used for the simulation are:

TE mode: This waveguide mode is dependent upon the transverse electric waves, also sometimes called H waves, characterized by the fact that the electric vector (E) being always perpendicular to the direction of propagation [23].

TM mode: Transverse magnetic waves, also called E waves are characterized by the fact that the magnetic vector (H vector) is always perpendicular to the direction of propagation [23].

2.2.3 Simulation models and methodology of analysis

The synthesis of AuNS is a very complex process that cannot be controlled in sufficient detail to achieve a specific morphology. For this reason, our goal in simulating the absorbance of nanostars has not been to model the exact optical response of an actual gold nanostar, but only of an idealized structure of the average gold nanostar. Thus, the core of the AuNS was approximated by a sphere of diameter "D" with radially oriented multiple branches. Each branch was approximated by a truncated right circular cone described by its length "L", the half angle aperture "α" and a hemispherical tip of radius "r" as illustrated in Fig 2.4. Table.2.1 summarizes the definitions used and the values given to all the parameters used in describing a typical AuNS.

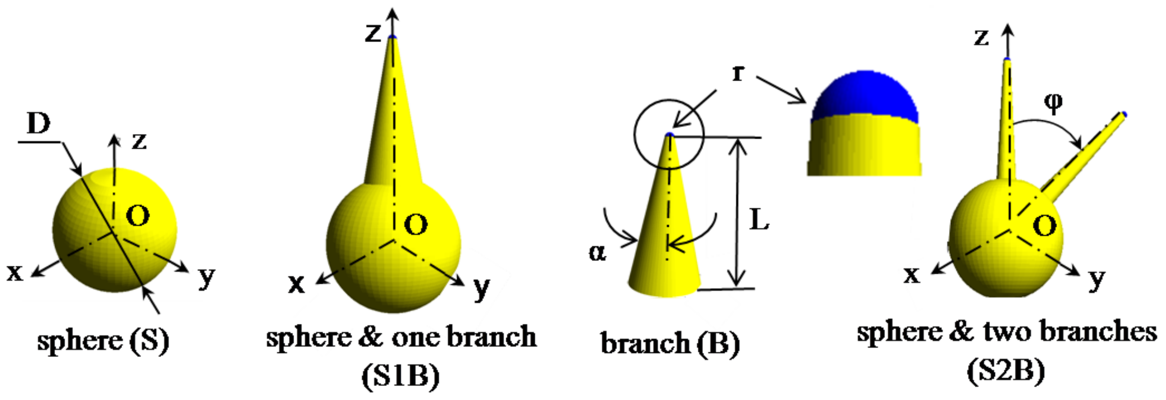


Fig 2.4: Parameters defining the morphology of an AuNS FDTD-simulation model S, S1B, B and S2B

Table 2.1: Parameters used in describing a typical AuNS.

Parameter	Definition	Value(s)
D [nm]	Diameter of the core (fixed)	100
N	Number of branches	1, 2
L [nm]	Length of branch	100, 125, 150
R [nm]	Radius of hemispherical tip (fixed)	3
α [°]	Half angle aperture of the branch	3, 10, 15
φ [°]	Branch orientation angle on the axis Oz	0, 45, 180

Step 1: In the first stage of simulation, the absorption spectrum normalized on the maximum absorption value for the TE mode, and TM mode. For the TE mode, there is two absorption peaks at 560 nm and 790nm because the electric field of the TE mode is polarized only in the

X direction and thus excites both the LSPR mode in the gold nanostar, as indicated in Fig 2.5. There is a peak around 570 nm for both TE and TM mode because the circular shape of gold nanoparticle supports degenerate modes.

For this purpose, TE mode is better than TM mode because TE mode excites both the LSPR peak so that the shift of this LSPR peak can be determined without being possibly mixed with other LSPR peaks. Therefore, in the following discussions, we focused on the TE mode excitation only.

Step 2: All models had coplanar branches, lying in the plane of the paper. Within this plane, the orientation of each branch was then fully described by the angle "φ" on the vertical axis Oz of the {x,y,z} reference system. Furthermore, the contribution of the constituent elements of the overall absorbance spectrum of an AuNS model was analyzed by using their dimensions as key parameters with values estimated from SEM micrographs of the AuNS samples.

However, to reduce the large number of the parameters and analyses required by the broad polydispersity of the AuNSs, the diameter of the sphere of the NS core was held constant at a value of D = 100 nm. Therefore, a second stage of the analysis would be required to compensate for the missing variability of the optical response caused by holding the sphere diameter at a constant value. For the same reason, the radius of the semispherical tip of each branch was held constant at r = 3 nm. In this case, only the magnitude of the electric field enhancement is affected locally, regions surrounding the branch tips, without affecting the spectral peak positions of the LSPR modes [24, 25].

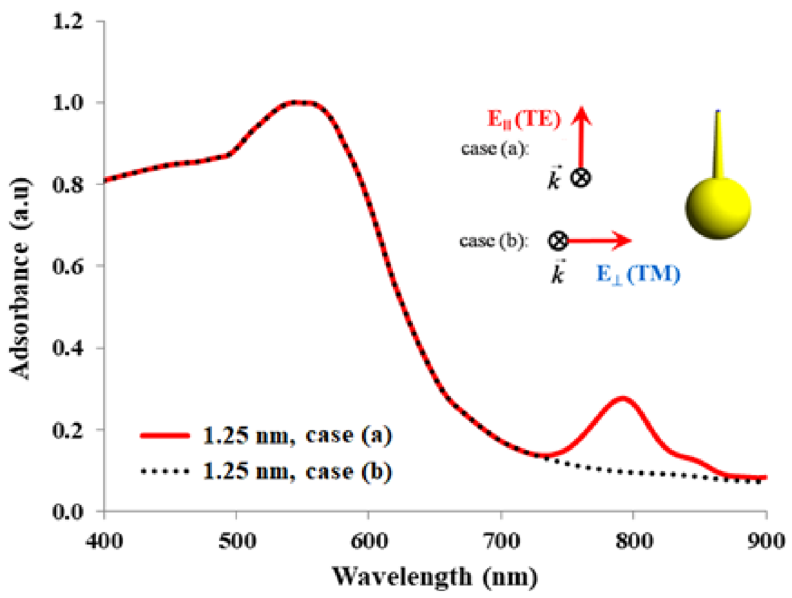


Fig 2.5: Normalized absorption spectra of a single gold nanostar on the maximum absorption value for the TE mode and TM mode.

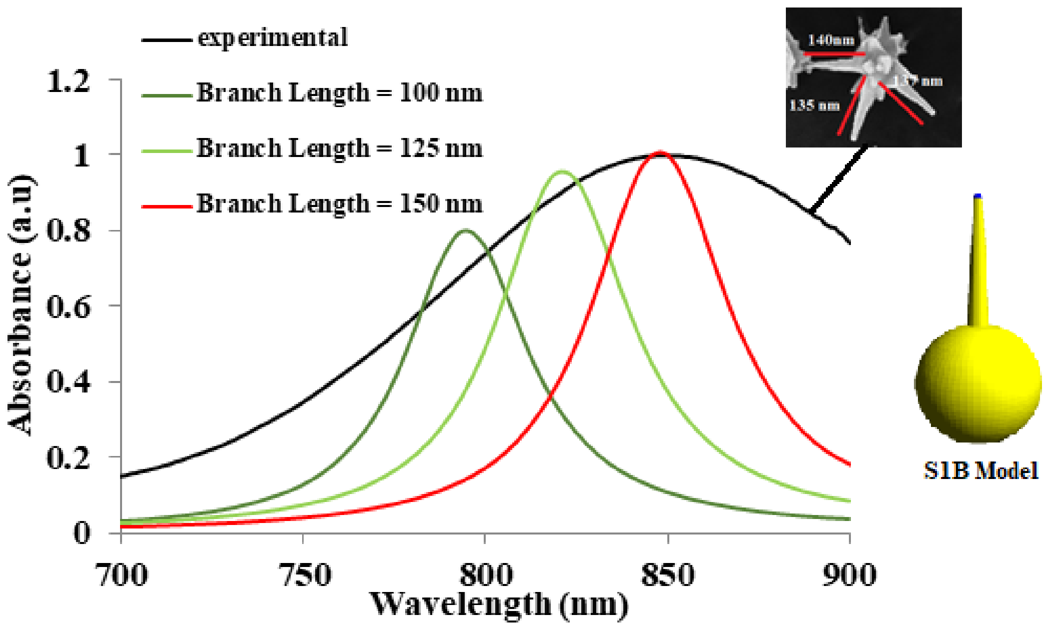
2.3 RESULTS

2.3.1 Influence of the branch length on the LSPR of Gold Nanostar

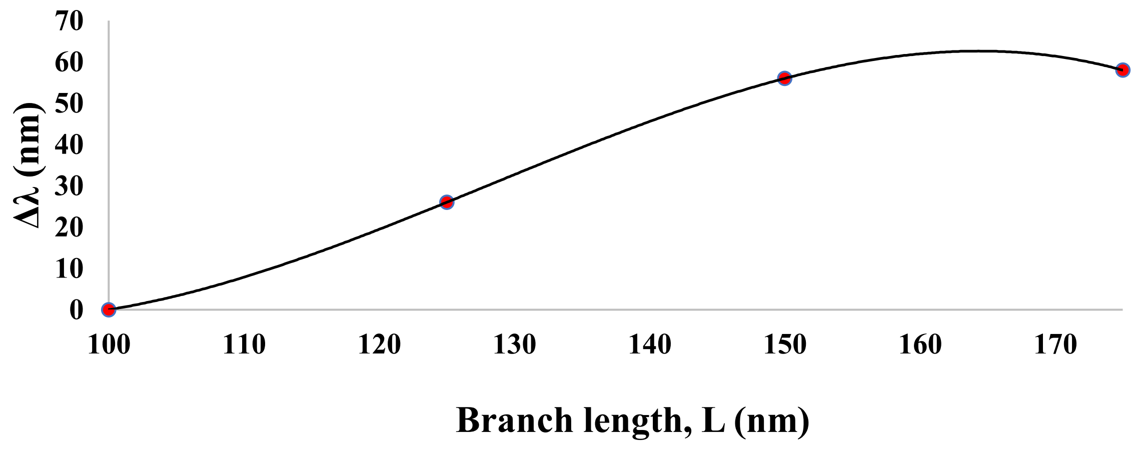
The attachment of a branch of half-aperture angle α = 3° to the sphere reduces its order of symmetry and renders it anisotropic. Therefore, its absorbance spectrum becomes dependent on the polarization of the incident light. When the polarization plane of the incident light wave is parallel to the branch, a longitudinal SPR mode is excited in the branch. The peak wavelength of the plasmon is red-shifted on the plasmon band of the sphere. Keeping the half-aperture angle of the branch at a constant value of α = 3° and varying only the length of the branch, the peak wavelength of the longitudinal SPR red-shifts on the plasmon band. The absorbance curves for the branch lengths of 100, 125 and 150 nm have plasmon bands located at 794 nm, 820 nm and 850 nm, respectively as shown in Fig 2.6a. The shift in the wavelength was evaluated for concerning the gold nanostars cluster with the average branch size of 140 nm.

However, as the branch length was 175 nm, there was no shift noticed in the plasmon band. Fig 2.6b shows the SPR peak wavelength as a function of the branch length. The SPR peak wavelength increase as the branch length increases.

In reality actual AuNSs have branches of different morphologies and the total response of the nanostars is an ensemble of response of the individual component particles. Hence, we estimate that the spectrum of an ensemble of polydisperse AuNSs by adding the individual spectra and compare with the experimental data [26].



(a)



(b)

Fig 2.6: (a) Absorbance spectra of the S1B models, semi-aperture angle α=3° and branch lengths 100, 125, 150, and 175 nm respectively, normalized to the peak intensity of the S model. **(b)** Change of the SPR peak wavelength as a function of the branch length.

2.3.2 Effect of number of branches

Due to its high order of symmetry, the model of the spherical core alone is isotropic and has a single dipolar LSPR mode around 550 nm, irrespective of the polarization of the incident light wave. The attachment of a branch to the sphere reduces the order of symmetry of the structure, which becomes anisotropic. Its absorbance spectrum is, therefore, dependent on the polarization of the incident light. When the incident light beam is polarized parallelly to the axis of the branch, as shown in Fig 2.3, a longitudinal LSPR mode is excited in the branch. Its plasmon band is red-shifted on the plasmon band of the sphere. Keeping the half angle aperture and the length of the branch at a constant value of α = 3° and 100 nm respectively, and varying only the angle between branch, the wavelength shift of the longitudinal LSPR on the plasmon band of the sphere strongly depends on the branch orientation. The absorbance curves corresponding to the branch orientation of 45°⊥, 45°∥, 180°, single branch and spherical structure, have plasmon bands located at 794 nm, 838 nm, 794 nm, 794 nm and 550 nm, respectively as seen in Fig 2.8. The absorbance spectrum of the sphere is represented by the single curve, plotted in black, of Fig 2.7, in response to both s- and p-polarized light. This is one of the most interesting features in the simulated spectra. In order to understand the origins of this feature, it is important to know that maximum spectral shift can be achieved when the tips of the AuNS is aligned along the direction of incident light polarization due to E-field more efficiently concentrated between tips.

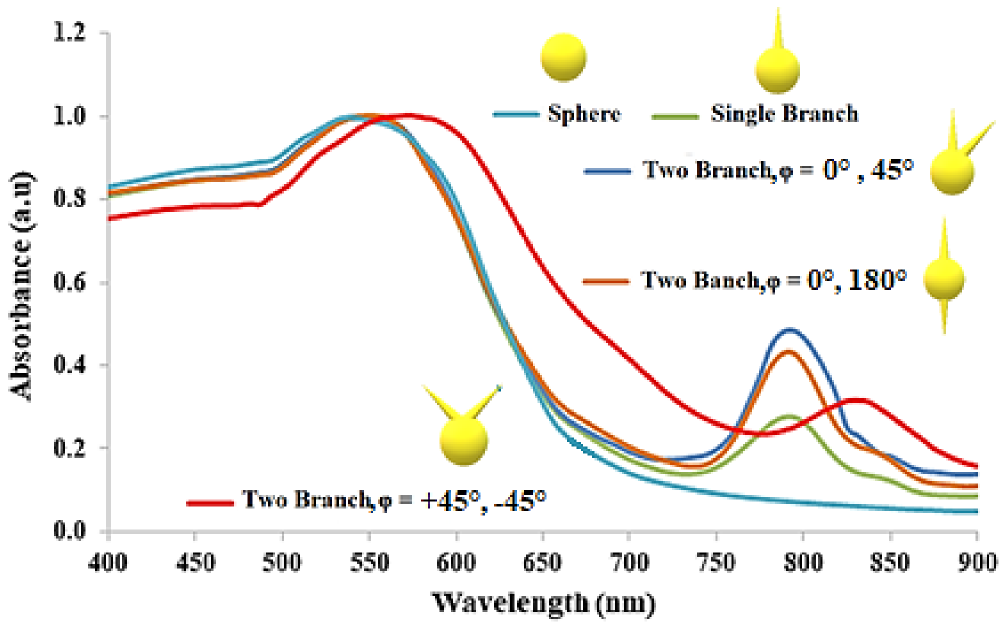


Fig 2.7: FDTD-calculated absorbance spectra normalized on the absorbance of the LSPR of the spherical core, located at 550 nm. The blue curve represents the spectrum of the Sphere model, which is unchanged for light polarized parallelly or perpendicularly to the axis of the branch. The absorbance spectrum of the model is dependent on the orientation of the polarization of light. In this case, the structure was illuminated only by light polarized parallelly to the branch axis. The absorbance spectra of branch orientation of 45°$\perp$, 45°$\parallel$, 180°, single branch and spherical structure are represented, keeping the half angle aperture and the length of the branch at a constant value of α=3° and 100 nm, respectively.

2.3.3 Effect of branch aperture on the LSPR of the Gold Nanostars

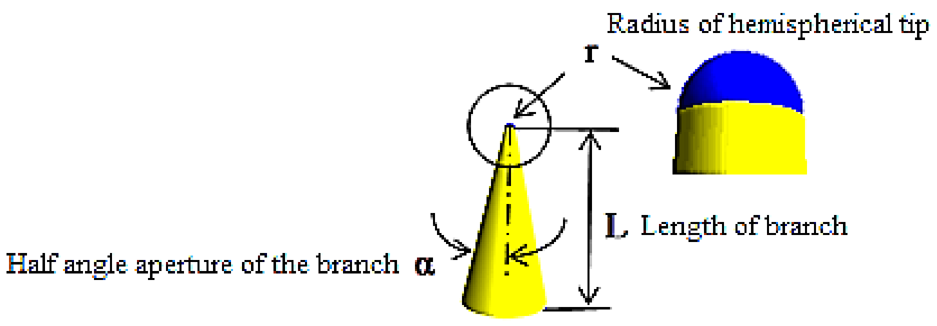
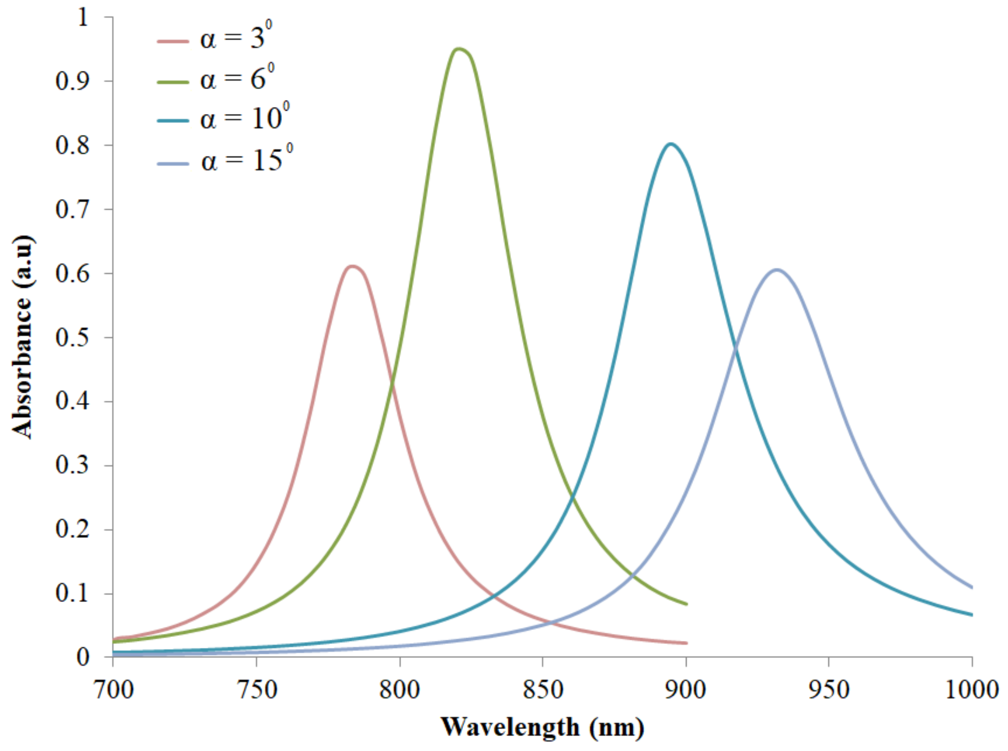

Radius of hemispherical tip
r
L Length of branch
Half angle aperture of the branch α

Fig 2.8: Pictorial representation of the half angle aperture of the of the branch

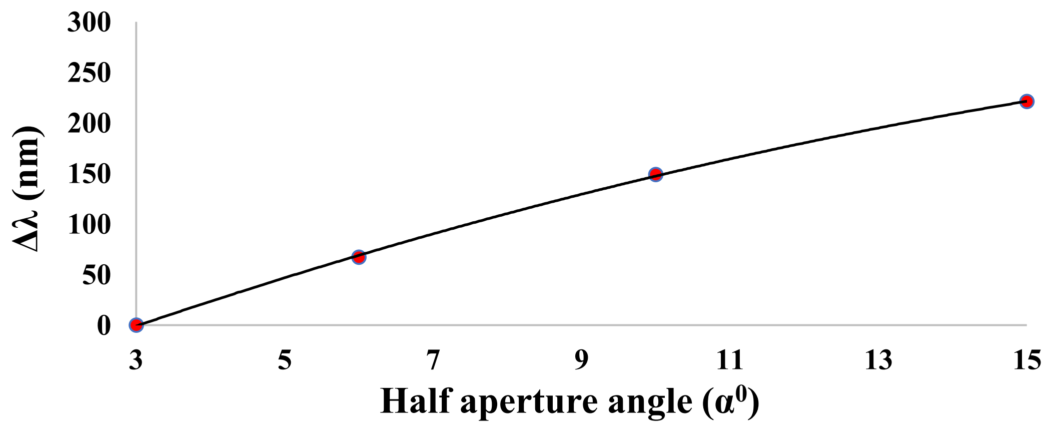
As illustrated in Fig 2.8, the branch of the nanostars is conical in geometry. A longitudinal slice of this cone of branch shows the angular aperture. The angle α is one-half the angular aperture of the branch.

Increasing the aperture of a branch increases the red-shift of its plasmon band, compared to the spectra of the model with the same branch lengths but of the constant aperture. The spectra of the upper part of Fig 2.9a, correspond to the sphere with one branch models with half-angle aperture increased to α = 3°, 6°, 10°, and 15° illuminated by light polarized parallelly to the axis of the branch with unchanged branch lengths. The absorbance curves for the half-angle aperture of α = 3°, 6°, 10°, and 15° have plasmon bands located at 756 nm, 813 nm and 895 nm, and 967 nm respectively. Both spectral location and absorbance of the second LSPR are highly dependent on the aperture of the branches. This feature is useful for understanding the location and absorbance of the second LSPR peak. It also demonstrates that branches of different geometries can compensate each other to form by superposition of any reasonably similar spectrum of the whole AuNS. Fig 2.9b, shows the SPR peak wavelength as a function of the Half aperture angle (α^0). The SPR peak wavelength increase as the half aperture angle (α^0) increases.

Both spectral location and absorbance of the second LSPR depend on the aperture of the branches. This feature is useful for understanding the effect on absorbance of the second LSPR peak. It also demonstrates that branches of different geometries can compensate each other to form by superposition any reasonably similar spectrum of the whole AuNS. Therefore, it is necessary to under the key parameters through which the plasmon bands can be blue shifted or red shifted.



(a)



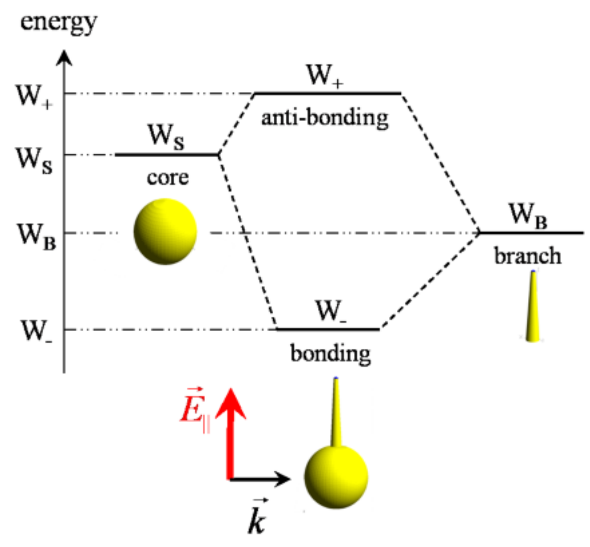
(b)

Fig 2.9: (a) Absorbance spectra of the Sphere with single branch models, semi-aperture angle α=3°, 6°, 10°, and 15° nm respectively, normalized with respect to the peak intensity of the S model and has a branch length of 100 nm **(b)** Change of the LSPR peak wavelength with reference to Sphere model is shown as a function of the aperture angle.

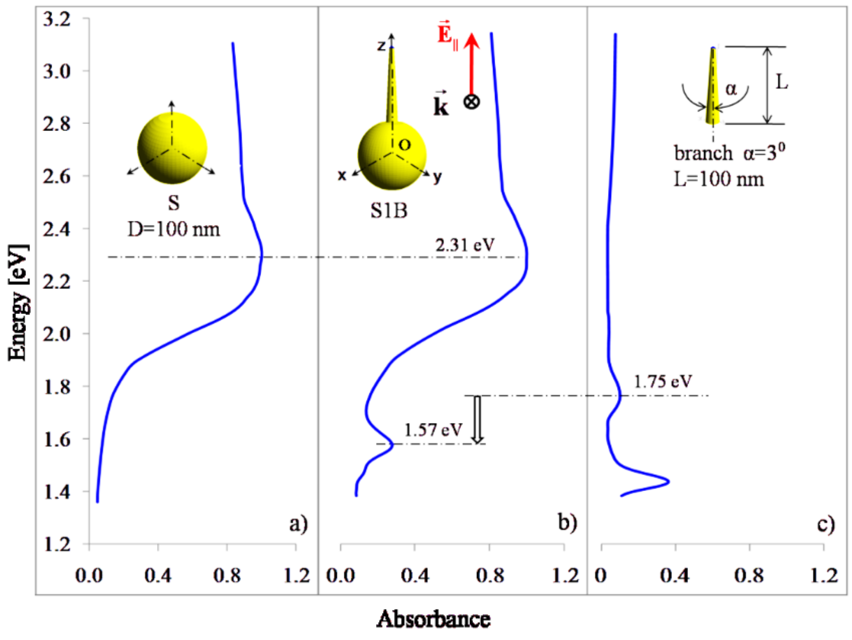
2.3.4 Plasmons Hybridization Interaction

Comparing the first two LSPR modes of the sphere with one branch model (core and one branch) with the corresponding LSPR modes of the separate core and branch models, a clear shift of the spectral positions is noticed, as shown in Fig 2.10a and Fig 2.10b. The left and right-hand side panel of each figure present the absorbance spectra of the separate core and branch models, while in the middle panel the spectra of the core-and-branch union models are shown, all resulting from illumination with light linearly polarized parallelly to the branch axes. In the presented spectra, the branch of half angle aperture α is 3° and length L is 100 nm: The second LSPR plasmon band is red-shifted. This shift is maintained for modified lengths and apertures of the branch and can be understood as caused by the interaction between plasmons. According to the plasmon hybridization theory [27, 28] as shown in Fig 2.10a, when individual plasmons on separate structural parts come in sufficient proximity to each other, they interact forming bonding and antibonding plasmons of lower and higher energy, respectively. In this case, gold nanostar, geometry-dependent plasmon resonances result from the interaction between the frequency plasmon response of a sphere and that of a branch. The elementary resonances interact with each other and hybridize in a way that is analogous to the hybridization between atomic orbitals. This interaction results in the plasmon resonances splitting into two new resonances: the lower-energy symmetric or “bonding” plasmon (W_-), and the higher-energy antisymmetric or “antibonding” plasmon (W_+) as shown in Fig 2.10a [29, 30].

Thus, in Fig 2.10b, the separated branch of 3° half angle aperture and length 100 nm has its LSPR positioned at 1.75 eV or 708 nm. This wavelength is red-shifted to 1.57 eV or 793.7 nm when the branch is attached to the sphere, which can be interpreted as being caused by the formation of a stabilizing lower energy or "bonding" plasmon.



(a)



(b)

Fig 2.10: Energy level diagram of various Nano shapes **(a)** The energy level diagram of the hybridization theory of plasmons [7, 23]. **(b)** Energy diagram of plasmon interaction in AuNS, model S1B where the branch parameter values are α=3° and L=100 nm.

2.5 Conclusion

Through FDTD-analyses of the optical response of idealized AuNS models, it was shown that the geometry of the branches is a key tuning factor of the absorbance peak of AuNSs. This can be shifted both in intensity and position, depending on the length and aperture of the branches. Since actual AuNSs have branches of different morphologies and because an ensemble of AuNSs is the superposition of the individual optical responses of its component particles, we estimate that the spectrum of an ensemble of polydisperse AuNSs by averaging the component spectra would and compared with the experimental data. To adjust the overall optical response of the AuNSs for maximal sensitivity within a well-defined region of the optical spectrum, the most effective means of fine-tuning may be desired. It is, therefore, useful to know the rate of shift of the second LSPR on each factor causing the shift. This is especially important if AuNS synthesis can be carried out in a controlled fashion, in which case the optical properties of AuNSs can be tuned for optimal performance before synthesis. We can conclude that the LSPR fine-tuning by adjusting the aperture parameter of the branch is much more effective than adjusting the length. Therefore the knowledge of the optical properties depending on the above geometrical parameters is of high importance in tailoring AuNSs of desired optical properties, such as maximal sensitivity.

CHAPTER 3

Tuning of Morphology and Stability of Gold Nanostars through pH Adjustment

3.1 Introduction

Among a variety of possible nanoparticle morphologies, gold nanostars (AuNSs), comprising a central core from which multiple sharp branches protrude, have been pinpointed as highly efficient surface-enhanced Raman scattering (SERS) substrates, due to their tunable localized surface plasmon resonances (LSPR) and nanoantenna effects by controlling the core size, along with tip length and sharpness [31 – 38]. A number of methods have been reported for the synthesis of AuNSs, though, either a seedless, or a seed-mediated approach, using capping molecules such as poly(N-vinylpyrrolidone) (PVP) or cetyltrimethylammonium bromide (CTAB), but also under surfactant-free conditions [32, 37, 39 – 41]. PVP-capped AuNS can be synthesized with tailored dimensions and optical properties [32, 42], and used for ultrasensitive SERS detection [43], but the SERS signal is affected by analyte chemisorption due to the adsorbed polymer, which can be removed only via a time-consuming process [33]. For this reason, both the core and the morphology of the branches have a strong influence on the position of the LSPR band. Due to the strong enhancement of the electric field and light concentration on the tips of the branches, AuNS can be successfully used as very efficient SERS substrates, sensors, as well as for targeted photothermal therapy and photoacoustic imaging emerging applications.

However, AuNSs typically display poor stability, resulting in changes in the particle's morphology (reshaping) and the corresponding LSPR blueshifts, which has also been reported for other nanostructures with sharp tips [45, 46]. In the absence of capping molecules, the colloidal stability is also compromised, as van der Waals attractions become dominant, eventually leading to particle aggregation [47]. These observations clearly indicate that the surface properties of AuNSs play a critical role in the colloidal and chemical stability, which is crucial for SERS relevant applications [48].

In this work, the impact of pH over a wide range (4.5 to 11.0) on the morphology and plasmonic properties of gold nanostars, obtained by seed-mediated synthesis, were systematically investigated.

3.2 Materials and Methods

3.2.1 Materials

The hydrochloric acid solution, silver nitrate ($AgNO_3$), sodium citrate ($Na_3C_6H_5O_7$), ascorbic acid ($C_6H_8O_6$) and PVA (C_2H_4O) (MW 89,000-98,000g/mol) were purchased from Sigma-Aldrich. Gold chloride (hydrogen tetrachloroaurate) (HAuCl4 ·4H2O) was acquired from Alfa Aesar Chemical.

3.2.2 Synthesis of Gold nanostars

The Gold nanostars were synthesized, by using a seed-mediated method. We further extended this protocol to enable the size control of the stars, from approximately 45 to 116 nm in size, which translates to tuning capabilities of the longitudinal plasmon peak in the NIR region, from around 655 to over 750 nm.

The seed solution was prepared by adding 15mL of a sodium citrate solution to100 mL of boiling 1mM $HAuCl_4$ solution. The solution turns, from the original yellow, first to dark blue, and finally, to purple red. This solution is termed “growth solution”. For the synthesis of nanostars, 10mL of 0.25 mM solution of gold precursor ($HAuCl_4.3H_2O$), 10µL of 1M HCl, 100µL of seed solution are mixed in an Erlenmeyer flask and stirred for 5 min. 50µL of the100mM ascorbic acid solution and 100µl of the 2mM $AgNO_3$ solution are added simultaneously and stirred for another 10 min when the solution turns to blue. The solution is then stored at 4 °C to stop the aggregation process and allow the growth of the branches. This solution is termed “Nanostars solution” [49].

3.2.3 Selection and preparation of the buffer solutions

Acidic buffer solutions (pH 3.5–7.0) were prepared by combining different proportions of Citric acid monohydrate, $C_6H_8O_7 \cdot H_2O$, Trisodium citrate dihydrate, $C_6H_5O_7Na_3 \cdot 2H_2O$, solutions. x ml 0.1M-citric acid and y ml 0.1M-trisodium citrate are mixed as shown in Table 3.1.

The citric acid and trisodium citrate dihydrate were chosen to prepare the acidic pH buffer solutions because [43]:

(a) The ions adsorbed on the surface of gold particles are the OH^{-1} and the citrate ions. Therefore, the buffer solution does not introduce new species.

(b) Citrate ions reduce the impact of counter ions and enhance the stability of the colloidal AuNPs solution.

Table 3.1: Preparation of the acidic buffer solutions

pH	x ml 0.1 M citric acid	y ml 0.1 M trisodium citrate
4.5	49.5	50.5
5.0	35.0	65.0
5.5	25.5	74.5
6.0	77.5	88.5
6.5	5.00	95.0

Highly alkaline pH buffer solutions (pH > 8) were prepared by adding an amount (x ml) of 0.1 M solution of sodium hydroxide as shown in Table 2. Sigma-Aldrich buffer reference center was used as a guide to prepare the buffer solutions.

Table 3.2: Preparation of the basic buffer solutions

pH	x ml 0.1 Msodium hydroxide
8.0	13.5
9.0	19.0
10.0	25.5
11.0	40.0

3.3 Results and Discussion

3.3.1 Effect of pH on the morphology of Gold Nanostars (AuNSs)

The influence of the pH of the nanostars solution on the particle coverage was investigated using SEM. The experimental procedure was as follows: The pH value of the nanostars solution was adjusted to 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 8.0, 9.0, 10.0 and 11.0 by adding the corresponding buffer solution. The nanostars solution was distributed uniformly on the surface of a glass slide. By immersing it for 5 hrs in the solutions with different pH values. After 5 hrs, the glass slide was removed from the solutions and cleaned with distilled water. The surface morphologies of gold nanostars on glass slides were observed using SEM. Table 3 shows the average particle size. When the pH value was decreased from 7 to 4.5, the particle size decreased sharply, while for pH values larger than 8 pH, the particle size increased.

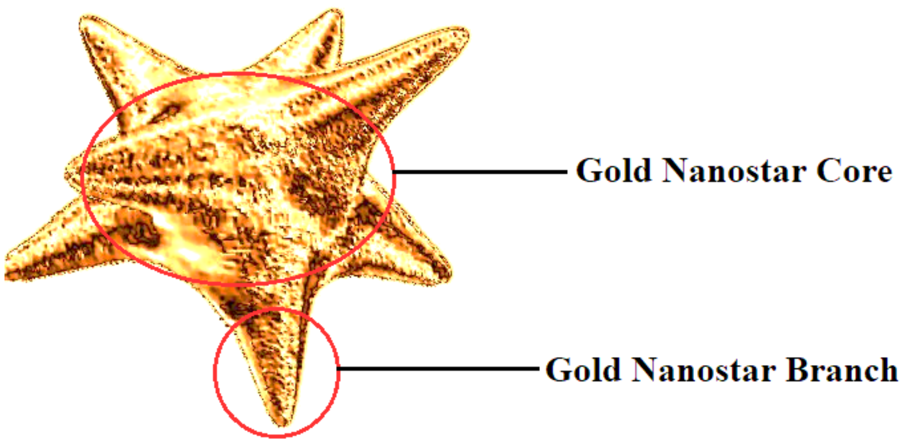


Fig 3.1: 3D representation of Gold nanostar core and branches as found in SEM images

Table 3: Average size of AuNS with respect to the pH of the solution

(At least 100 particles were measured to calculate their average sizes)

pH of AuNS solution	**Core Diameter (Average) (nm)**	**Branch Length (Average) (nm)**	**Average Size of AuNS (nm)**
4.5	55 ± 5	Spherical Transformation	No nanostars left in the nanostars solution
7.0	45 ± 5	20 ± 10	60 ± 10
11.0	45 ± 5	40 ± 20	90 ± 20

A unique 3D nanostar model as shown in Fig 3.1 was designed for gold nanostars, using the morphology obtained from their corresponding SEM images shown in Fig 3.2a. For a model, the branches protrude normally to the core surface, but in the model, they were randomly positioned on the core such as to maximize the inter-branch distance.

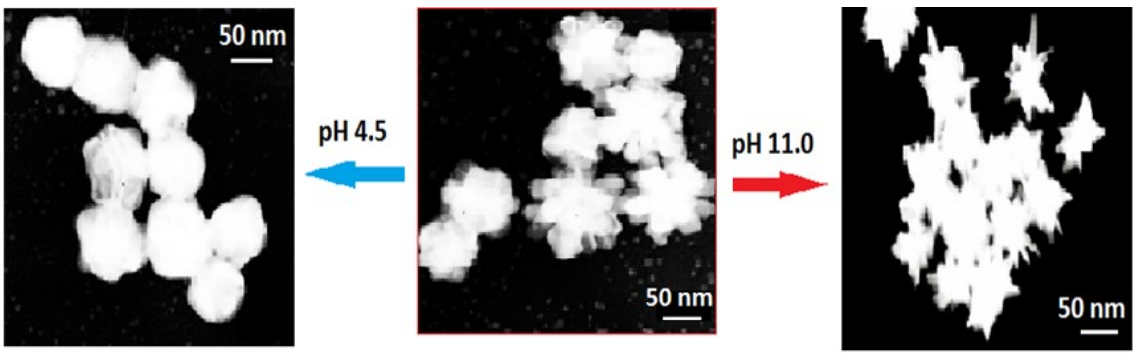
The average size of the AuNSswere60 ± 10 nm. When the basic buffer was added to 20 ml of nanostars solution, the first observation was an immediate color change i.e., the solution became dark blue and there was an abrupt increase in the absorbance values. It was observed that the higher the pH of the nanostars solution the darker is the color of the solution. When the pH of the AuNS solution was basic, the obtained gold nanostars were larger in size as shown in Table3. For example, the average sizes of the AuNS at pH 11.0 was found to be 90 ± 20 nm, as indicated by the SEM image in Fig 3.2a. Upon further study of the SEM images, it was found that the increase in the size of AuNS was due to the increase in the branch length, the core diameter of AuNS was almost constant, around 45 ± 5 nm. The average branch length at pH 11.0 was 40 ± 20 nm as observed by SEM.

In contrast, when the acid buffer was added to 20 mL nanostars solution, the diameter of the core of the AuNS increased from 45 nm to 55 nm as shown in Fig 3.2a, but the morphology change from nanostars to nanospheres was noticed as the reaction progressed. This result was consistent with the classic seeding growth of spherical gold nanocrystals reported in the literature [50, 51, 52].

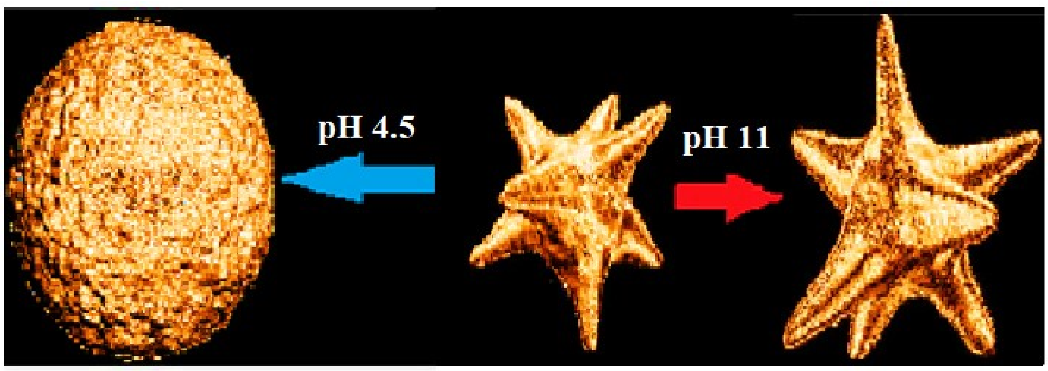
Fig 3.2c represents the UV-visible spectra carried out to characterize the morphology of AuNS. Typically, the LSPR of AuNS is represented by two peaks. The first peak is noticed around 500 nm – 550 nm which indicates the presence of nanospheres and the second peak is noticed at between 630 nm and above. Therefore, it is necessary to mention that not all seeds in the growth solution are transformed in the AuNSs, some of them will grow and form nanospheres [53 – 58].

The LSPR band of AuNS experienced a gradual red-shift from 650 nm to 750 nm when the pH was increased from 7.0 to 11.0. This may indicate the increase in the branch length of the AuNSs. An increase in the absorbance peak was also noticed with increase in the pH, which clearly indicated the increase in the number of AuNS particles.

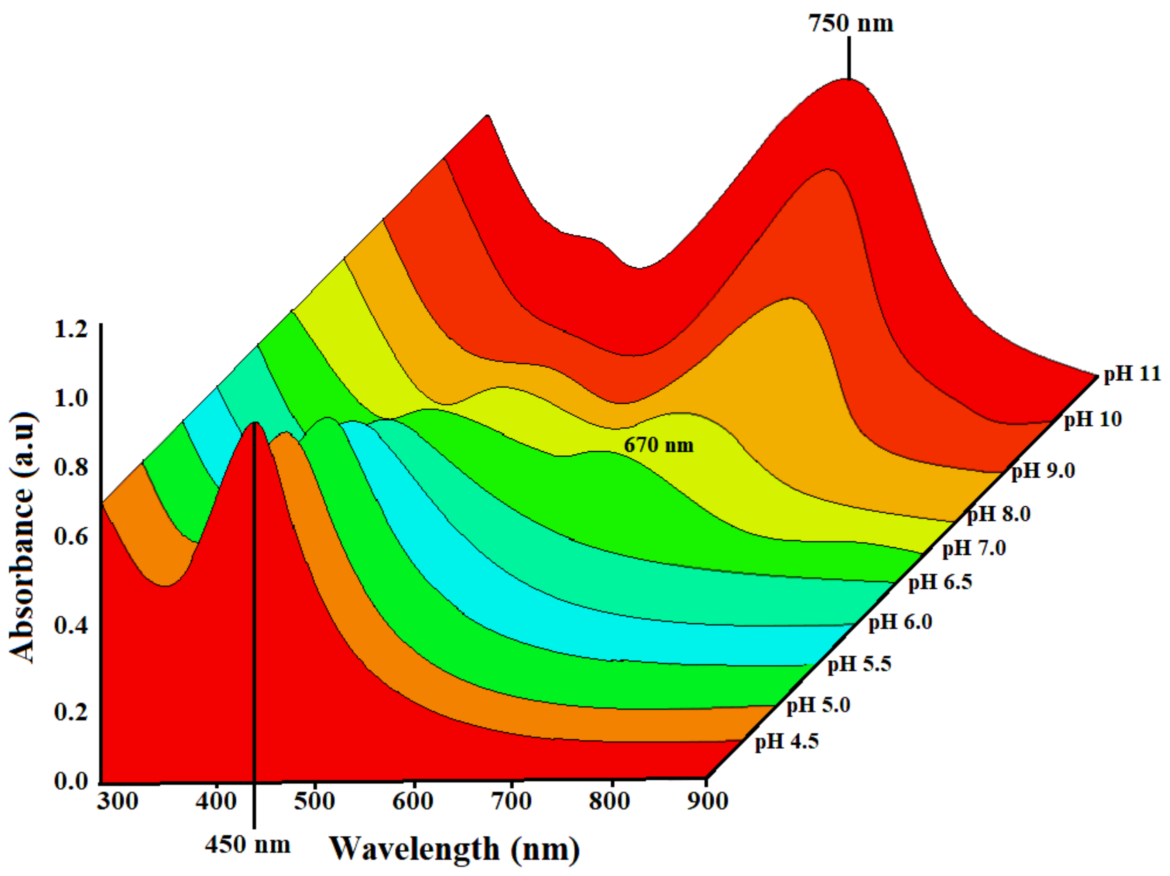
A gradual blue shift was experienced from 640 nm to 540 nm as the pH decreased from 7.0 to 4.5. The increase in the absorbance peak was also noticed with decrease in pH, which clearly indicated the increase in the number of AuNPs particles transformed from nanostars to nanospheres as shown in Fig 3.2a.



(a)



(b)



(c)

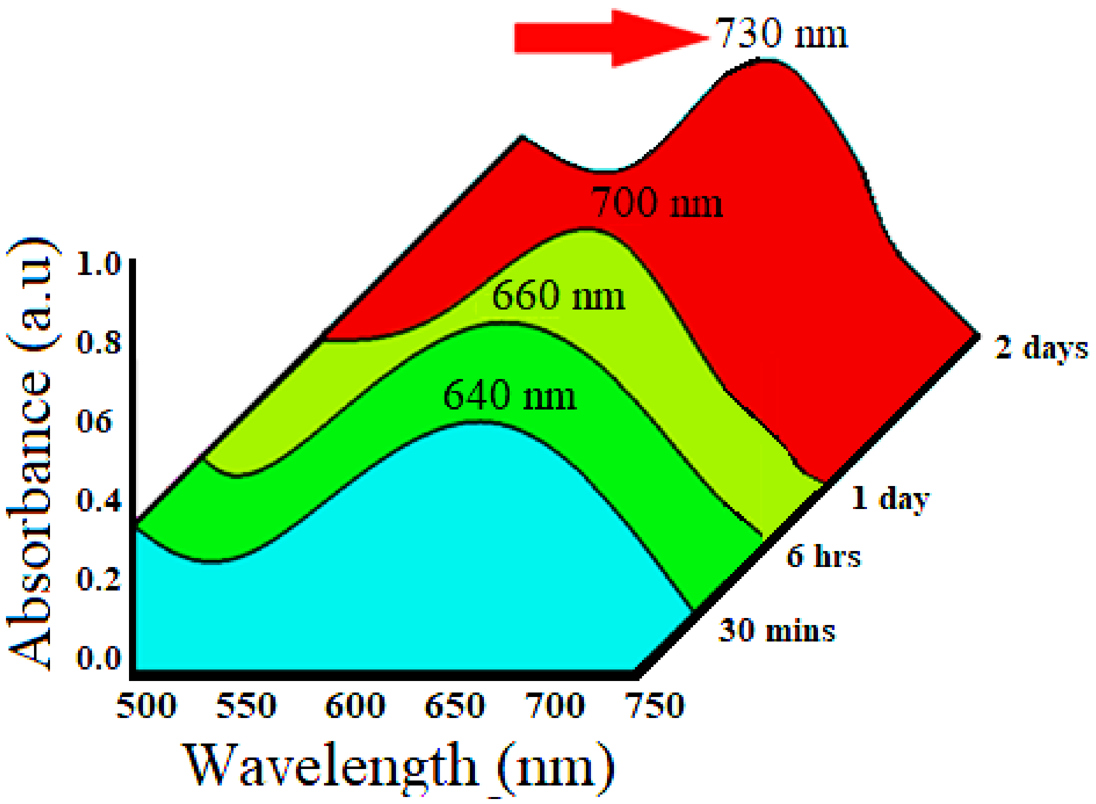
Fig 3.2: (a) SEM of AuNSs at pH 4.5, pH 7.0 and pH 11.0. **(b)** Schematic of the morphology change of AuNSs under different pH environments. **(c)** UV–vis absorption spectra of AuNS samples at pH values 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 8.0, 9.0, 10.0 and 11.0. (N = 10 Samples)

3.3.2 Temporal evolution of AuNS samples at pH 11

When 5 ml of base buffer was added to 20 ml nanostars solution to increase the pH of the nanostars solution from 7.0 to 11.0, a red shift from 660 nm to 720 nm was observed in the localized plasmonic spectra shown in Fig 3.3a. This indicates an increase in the overall size of the gold due to the formation of longer branches. The change in the color of the gold nanostars solution from light blue to dark blue was noticed when the base buffer was added to the solution.

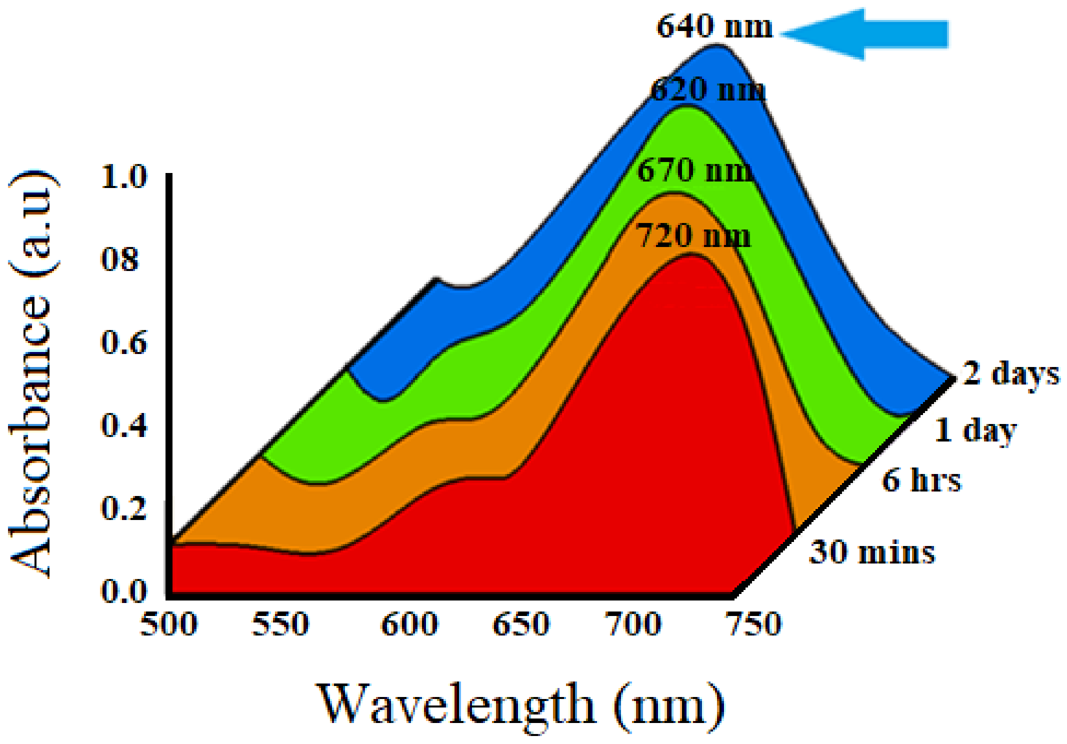
The position of their plasmon band was constant for a significant length of time i.e., even after 2 days, which shows that AuNS were stable in the basic medium as depicted in Fig 3.3a, whereas the AuNS solution at pH 7.0 aggregated and formed a black residual, resulting in a blue shift in the LSPR spectra of AuNS solution.

According to the classic Mie theory, the wavelength of the LSPR band is affected by three factors: (1) the nanoparticle's size; (2) the conduction electron density of the particle surface plasma; and (3) the dielectric constant of the metal particles [59, 60]. The increase in pH value initiated an electrophoretic migration, causing the particle surface to absorb more negative ions which leaded to stronger electrostatic repulsion between the particles. As a result, the particle spacing increased, and the particles did not aggregate. These results demonstrated that the nanostars solution with a higher pH produced a larger red-shift and are stable for longer period.



730 nm
700 nm
660 nm
640 nm
Absorbance (a.u)
1.0
0.8
06
0.4
0.2
0.0
500
550
600
650
700
750
Wavelength (nm)
2 days
1 day
6 hrs
30 mins

(a)



640 nm
620 nm
670 nm
720 nm
Absorbance (a.u)
1.0
08
0.6
0.4
0.2
0.0
500
550
600
650
700
750
Wavelength (nm)
2 days
1 day
6 hrs
30 mins

(b)

Fig 3.3: (a) The temporal evolution of UV–vis absorption spectra of AuNS samples at pH 11. (N = 10 Samples) **(b)** Represents the temporal evolution of UV–vis absorption spectra of AuNS samples at pH 7.0. (N = 10 Samples)

3.3.3 Temporal evolution of AuNS samples at pH 4.5

When 5 ml of acid buffer was added to 20 ml nanostars solution to decrease the pH of the nanostars solution from 7.0 to 4.5, a blue shift from 660 nm to 540 nm was observed as shown in Fig 3.4. Upon further investigation, it was found that the nanostars have undergone morphological transformation to form nanospheres. The decrease in pH of the nanostars solution causes the dissociation of AuNS into Au^+ and Ag^+ ions. Initially, this results in the development of nucleation sites along with the instability that causes the nanoparticles to combine and form larger agglomerates. This is evident from the increase of the absorbance of the LSPR band and the blue shift of the plasmon band. These results demonstrated that the nanostars solution with a lower pH produced a larger blue-shift and are highly unstable.

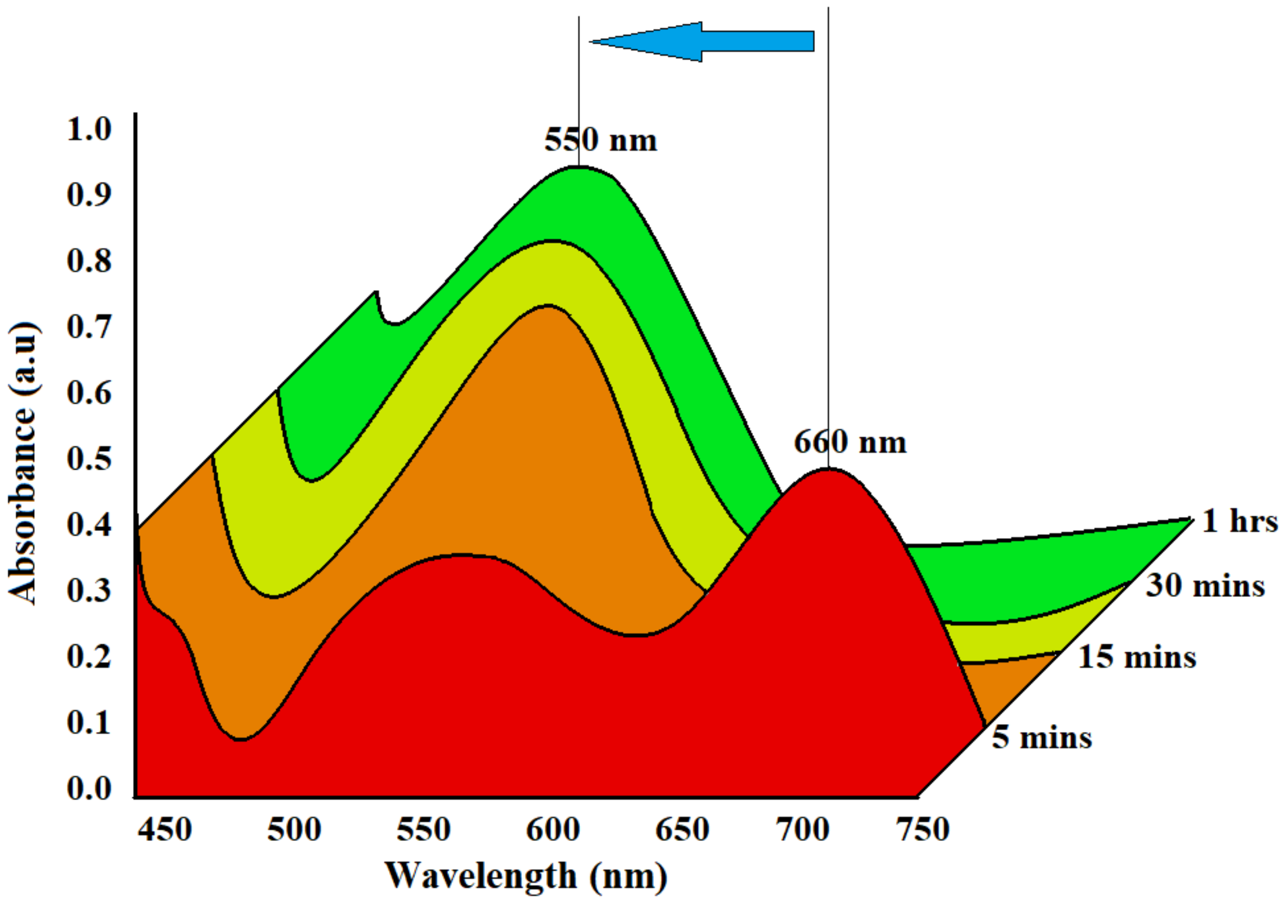


Fig 3.4: The temporal evolution of UV–vis absorption spectra of AuNS samples at pH 4.5 (N = 10 Samples)

3.3.4 Shape-tuning effect of base buffer at pH 11

To get more insights into the shape-directing effect of base buffer at pH 11, a series of experiments were performed in which the pH of the growth solution was varied from 4.5 to 11.0. In a typical experiment, 49.5 mL of 0.1 M citric acid and 50.5 mL of 0.1 M trisodium citrate was added to the 100 mL of growth solution to maintain the pH of 4.5. The LSPR peak observed at 540 nm confirms the presence of gold nanospheres at pH 4 as shown in Fig 3.5a. After 20 min of adding the acid buffer, a freshly prepared (60 mL) of 0.1 M NaOH base buffer was added to the growth solution to attain a pH of 11. Interestingly, a red shift in the LSPR peak from 540 nm to 670 nm is observed. The anisotropic growth of AuNPs upon addition of base buffer is shown in Fig 3.5a. A weak absorbance peak at 540 nm was observed throughout the experiment suggesting the presence of other spherical particles in the growth solution. Interestingly, the spectra show a tail extending into the red as shown in Fig 3.5a, indicating that the resulting AuNPs show significant aggregation. The broadness of the peak indicates the wide size distribution of the nanoparticles' sizes in the growth solution. The spectra provide an understanding about the morphology of noble metal nanoparticles.

The glass slide was placed in growth solutions with different pH values for 5 h. After 5 h, the glass slide was then removed from the sols/buffer solutions and cleaned with distilled water. The surface morphologies of AuNPs on glass slide were observed using SEM. The SEM image shown in Fig 3.5b, indicated that particles with a spherical morphology have aggregated when the pH of the growth solution is maintained at 4.5 pH. Upon addition of the basic buffer to increase the pH of the growth from 4.5 to 11 pH, a unique change in the morphology was observed in the SEM image shown in Fig 3.5b. The particles obtained after adding the base buffer to growth solution showed an anisotropic morphology. This could be an explanation to the peak intensity increases with time which might be due to the formation of tiny AuNPs which simultaneously converted into the bigger AuNPs with anisotropic morphology. Therefore, the results indicate the controlled pH environment is found to assist the morphological change of AuNPs.

The anisotropy of AuNPs may show a significant increase in the LSPR bandwidth but it would not be correct to conclude that the particles are AuNS, as once the nanostars are transformed to nanospheres, it is thermodynamically not possible for the particle to revert to its original morphology due to the increase in entropy of the transformation [61, 62].

The observed results summarized in Fig 3.5a and Fig 3.5b indicated the shape-tuning of base buffer at pH 11. Further, due to the presence of weak plasmonic band at 540 nm throughout the experiment indicates that the anisotropic morphology seen in SEM is not that of a nanostars but more of the aggregated tiny particles in the growth solution which has undergone reduction and aggregation phenomena upon pH change.

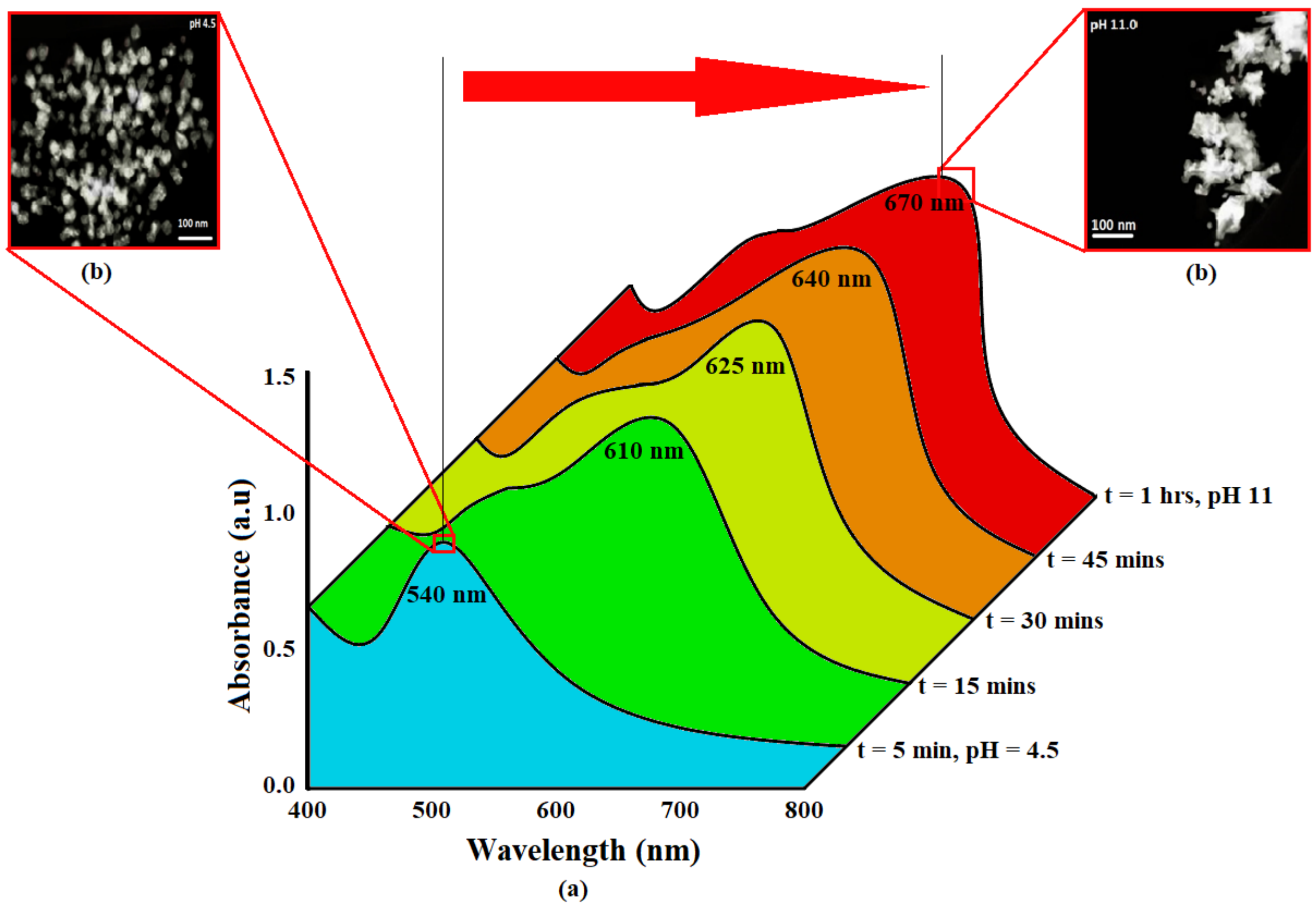


Fig 3.5: **(a)** The temporal evolution of UV-Visible spectra when the pH of the AuNSs was increased from 4.5 to 11.0 by the addition of 60 mL of 0.1 M NaOH base buffer to the growth solution. **(b)** SEM images of the nanoparticles at pH 4.5 and pH 11. (N = 10 Simples)

3.4 Conclusion

The results demonstrate that it is possible to address the issue of the stability of the AuNS considering the pH of the storing solution. The morphology, shape of the spectra and the stability of gold nanostars increased with increase in the pH of nanostars solution to 11.0. The AuNS retained their original optical properties and shape even after two weeks at room temperature. The

SEM image shows the cause of increase in the size of the nanostars to be that of increase in the branch length of gold nanostars. Upon further investigation of the shape-tuning effects of base buffer at pH 11, it is concluded that unlike the transformation of nanostars to nanospheres which has been reported in the literature, the nanostars transformed from nanopheres cannot be reversed to their original form again.

CHAPTER 4

Sonochemical Stabilization of Gold Nanostars: Effect of Sonication Frequency on the Size, Shape and Stability of Gold Nanostars

(AdvNanoBioM&D: 2017: 1(4):172-181, ISSN: 2559 – 1118)

4.1 Introduction

During the last decades, a wide variety of synthetic methods have been developed to prepare gold nanostars: *in situ* methods, seed-mediated methods [63, 64], one-pot methods [65 – 70], etc. More on the synthesis of nanostars can be found in the excellent review of Liz-Marzán et al. [71]. Although most of the methods can successfully produce gold nanostars (AuNSs), the synthesis, generally, requires the use of one, or even more stabilizers, to protect the freshly formed nanostars against aggregation.

Gold nanostars, due to their high surface area to volume ratio, are instable and have a strong tendency toward aggregation. Aggregation, due to Van der Waals, and/or electrostatic attraction forces, may alter the properties, reactivity, fate, transport, and biological interactions (e.g., bioavailability and uptake) of gold nanostars. As the aggregation barrier increases with increasing size, larger particles are significantly more stable than the smaller ones. Despite several attempts to improve the stability of surfactant-free gold nanostars, keeping the nanostars' morphology unchanged and avoiding aggregation over the time, is still challenging [72, 73].

Sonochemistry, the application of ultrasounds to chemical reactions and processes, is now successfully applied in nanochemistry as well. Because of the extreme conditions that can be achieved, unique mechanisms and reaction pathways may result. The wavelengths of ultrasounds are much longer than the molecular dimensions, thus, no direct interactions at molecular level can take place between ultrasounds and chemical species. Ultrasounds create acoustic cavitation, that is, the formation, growth, and implosive collapse of bubbles in a liquid. The collapse of the bubble can be assumed to occur adiabatically because it happens in such a short time. At this point, the accumulated energy, stored in the bubble, will be released in a localized spot, leading to extreme conditions such as a temperature of ~5000 K and a pressure of ~1000 bar. These localized "hot spots" are subjected to very high heating and cooling rates, which may hinder the aggregation and may influence the morphology of the nanoparticles [74 – 79].

In this work, the stabilizing effect of ultrasounds on nanostars is thoroughly investigated. We were especially interested to establish a relationship between the frequency of ultrasounds and the size and shape of nanostars, as well as the effect of ultrasounds on the aggregation kinetics. As far as we know, this is for the first time that sonochemistry is used to increase the stability of nanoparticles.

4.2 MATERIALS AND METHODS

4.2.1 Materials

Hydrochloric acid solution, silver nitrate, sodium citrate, ascorbic acid and PVA (MW 89,000-98,000) were purchased from Sigma-Aldrich. Gold chloride (hydrogen tetrachloroaurate) ($HAuCl_4 \cdot 4H_2O$) was acquired from AlfaAesar Chemicals.

4.2.2 Ultrasound treatment

Sonication was carried out using a horn type sonicator (Type: CCH-5838D-25LB PZT-4, frequency: 25 kHz, 40 kHz, 80 kHz, 120 kHz; radiating surface: 45 mm; resonance impedance (Ω): 10-25; static capacity (pF) ±10%: 3500-3800; insulation resistance: 10000 Mohm; input power (Watt): 100W). Immediately after the synthesis, a tightly closed vial, containing 10mL of gold nanostars solution, was subjected to an ultrasound treatment provided by a horn type sonicator (CCH-5838D-25LB). There was not direct contact between the radiating surface and the liquid medium. The gold nanostar solution, after synthesis, was sonicated for 5, 10, 15, and 20 min, respectively.

4.2.3 Synthesis of Gold Nanostars

Gold nanostars were synthesized by a seed-mediated method [64]. The colloidal solution was prepared by adding 15mL of a sodium citrate solution (1%) to100 mL of boiling 1mM $HAuCl_4$ solution. The solution turns, from the original yellow, first to dark blue, and finally, to purple red, the color of colloidal gold.

For the synthesis of nanostars, 10mL of a 0.25mM solution of gold precursor ($HAuCl_4.3H_2O$), 10µL of 1M HCl, 100µL of seed solution are mixed in an Erlenmeyer flask and stirred for 5 min. 50µL of the 100mM ascorbic acid solution and 100µl of the 2mM $AgNO_3$ solution are added simultaneously and stirred for another 10 min, when the solution turns to blue. The solution is then stored at 4^0C to stop the aggregation process and allow the growth of the branches.

4.2.4 Bubble Imaging

The experimental setup for visualization of the cavitation bubble collapse is shown in Fig 4.1. This device consists of a set of light sources, the CCH-5838D-25LB ultrasound wave generator, a high-speed 240 fps Mini Action camera, and a laptop. The camera is used to extract and record the images. The speed of image extraction is determined by the size of the image. For example, an image extraction speed of 240frames/s is used for an image size of 1,070 × 90 pixels. When the high-speed camera stopped recording the image files, the data were extracted and uploaded on the image processing software, designed for the experiment, by using the magic kernel up-sampling operator. Hence, the cavitation bubble in its collapse process can be recorded and stored for further analysis [80].

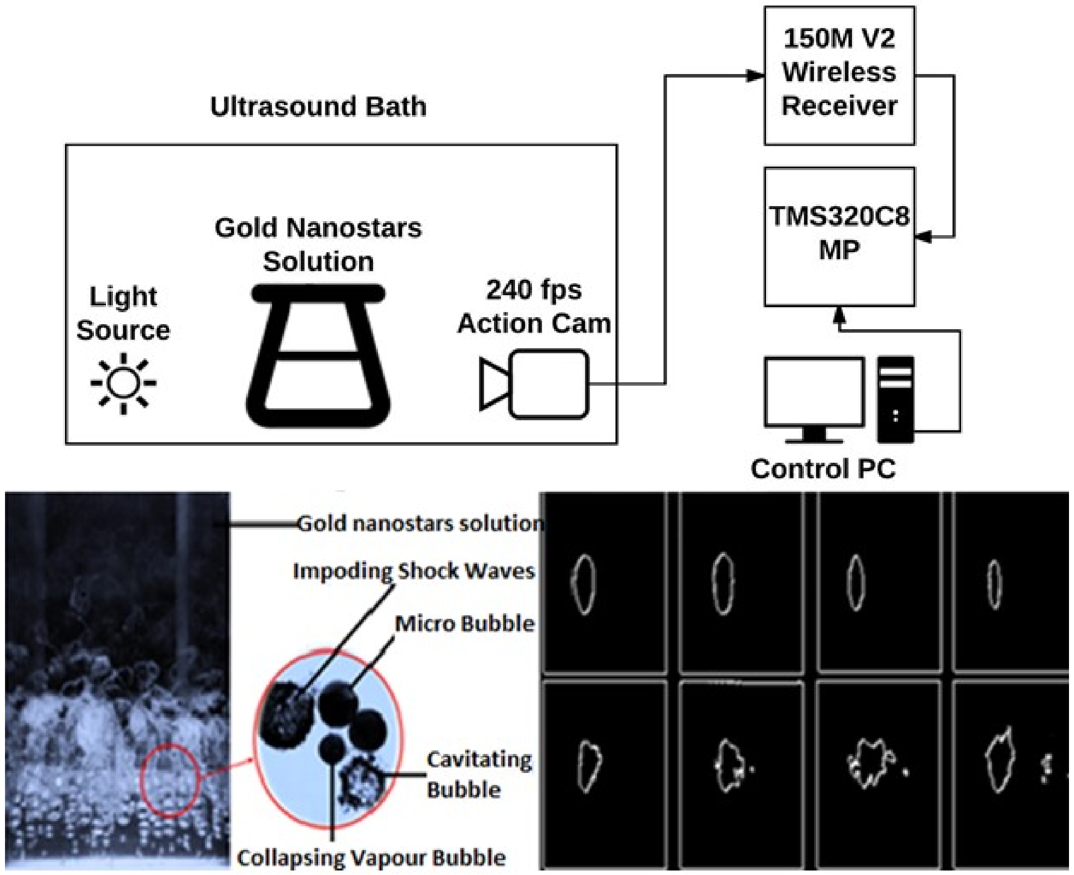


Fig 4.1: Cavitation images in the gold nanostars solution subjected to ultrasounds with a frequency of 25 kHz.

4.2.5 Characterization

The morphology of the nanoparticles was studied by using Scanning Electron Microscopy (SEM). Spectral measurements have been taken by using a LAMBDA 600 UV-Vis spectrometer in the wavelength range of 400 to 900 nm. To visualize the nanoparticle morphology by SEM, a drop of solution was casted onto a glass slide and left to evaporate.

4.3 Result and Discussion

4.3.1 The Chemistry of the Reduction of Gold Ions.

Gold nanoparticles have been synthesized by using the well-known "citrate" method [81]. Briefly, the reactions involved in this synthesis are shown below:

It is thought that the initial step of the reaction is the oxidation of the citrate ion, that results in the formation of dicarboxy acetone:

$$(^{-}OCOCH_2)_2C\ (OH)\ COO^{-} \rightarrow (-OCOCH_2)_2C = O + CO_2 + H^{+} + 2e^{-} \qquad \textbf{[Eq 4.1]}$$

Next, auric chloride is reduced to the aurous salt:

$$AuCl_3 + 2\ e^{-} \rightarrow AuCl + 2Cl^{-} \qquad \textbf{[Eq 4.2]}$$

Which will disproportionate as shown in the following reaction:

$$3AuCl \rightarrow 2Au^{0} + AuCl_3 \qquad \textbf{[Eq 4.3]}$$

The disproportionation step is facilitated by dicarboxy acetone.

To obtain monodispersed, non-agglomerated silver nanoparticles, further used for the synthesis of nanostars, the reduction in aqueous solutions of silver nitrate, was used. Because the ascorbic acid is a mild reductant, it was chosen to reduce Ag^{+} [82]. The presence of Ag (I) results in the deposition of AgCl and/or Ag (0) on the surfaces of the initial spherical AuNP cores as shown in the reaction:

Ag^+ (aq) + +$C_6H_8O_6 \rightarrow 2Ag + C_6H_6O_6 + 2H^+$ [Eq 4.4]

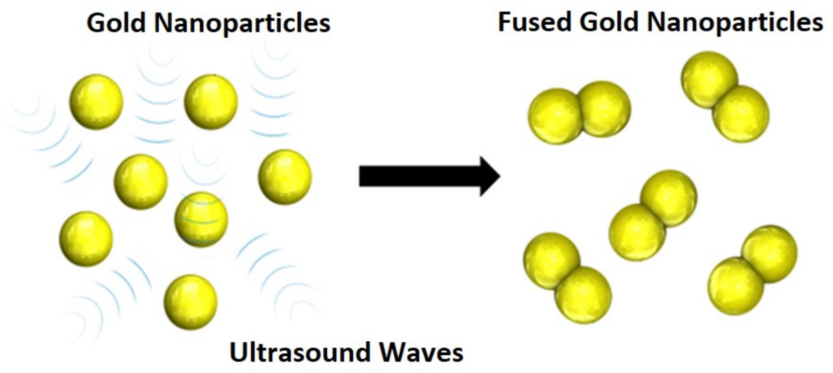
The silver atoms on the surface of spherical gold will become the starting points of the branches.

4.3.2 Effect of sonication on the morphology of Gold Nanostars (AuNSs)

The number of tips on each AuNSs is not constant, and the distribution of the shape and position of the tips varies [20]. When the volume of AuNSs is small, the core makes a significant contribution to the spectrum that will show two marked peaks, at 540, and 700 nm, respectively. As the volume of AuNSs increases, the number of tips increases as well, and the effect of the core on the spectrum decreases gradually [63] [83].

Gold nanoparticles with an average diameter larger than 40 nm melt at a temperature between 1200 K and 1300 K [1] [84]. The nearly adiabatic bubble collapse, due to cavitation, leads to almost 9 orders of magnitude of enhancement in energy per molecule, producing internal temperatures of$\sim 5 \times 10^3$ K [85, 86], sufficient for AuNSs to melt and fuse with other nanoparticles, at the contact, or through interparticle collision, inside the hot phase of a cavitation bubble.

The SEM image of AuNSs, after 20 min of sonication treatment at 25 kHz, was analyzed by using the edge detection algorithm with a magic kernel up-sampling operator as shown in Fig 4.2. It was found that the cores are fused after sonication, forming dumbbell-like particles. Therefore, it is likely that the fusion of Au NPs occurs by interparticle collision, because of the concentration and the nanoparticles size [87].



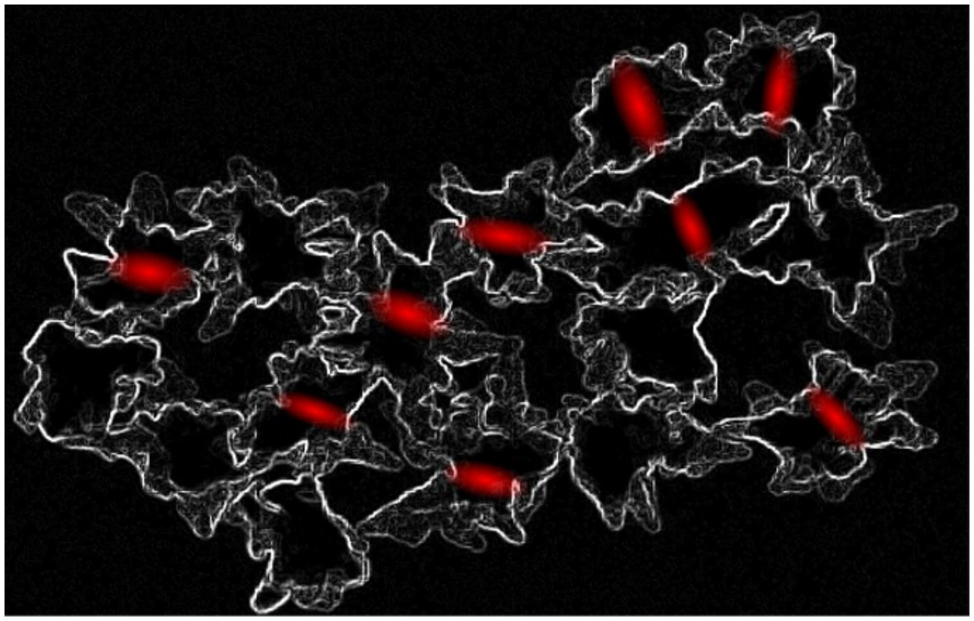
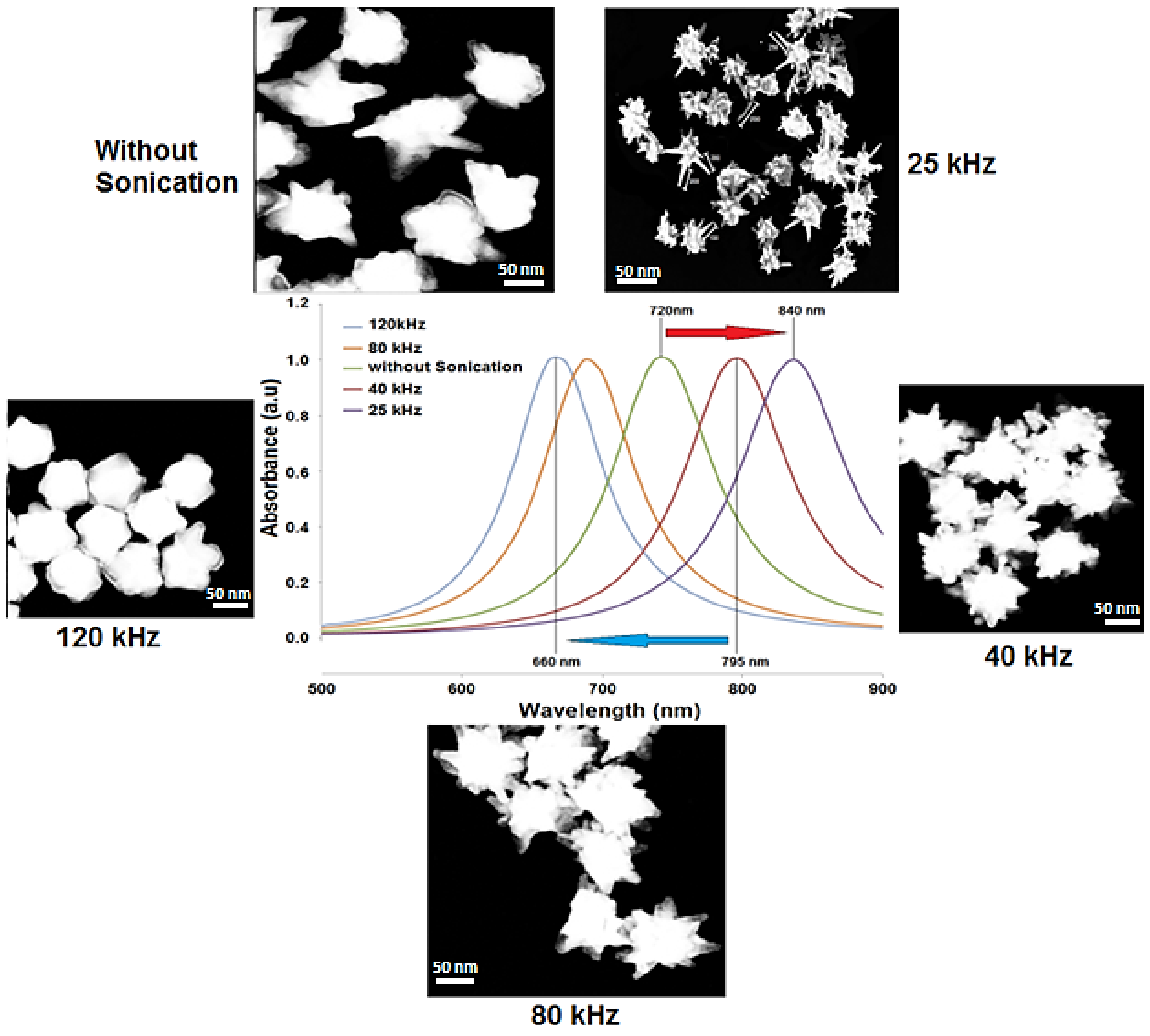
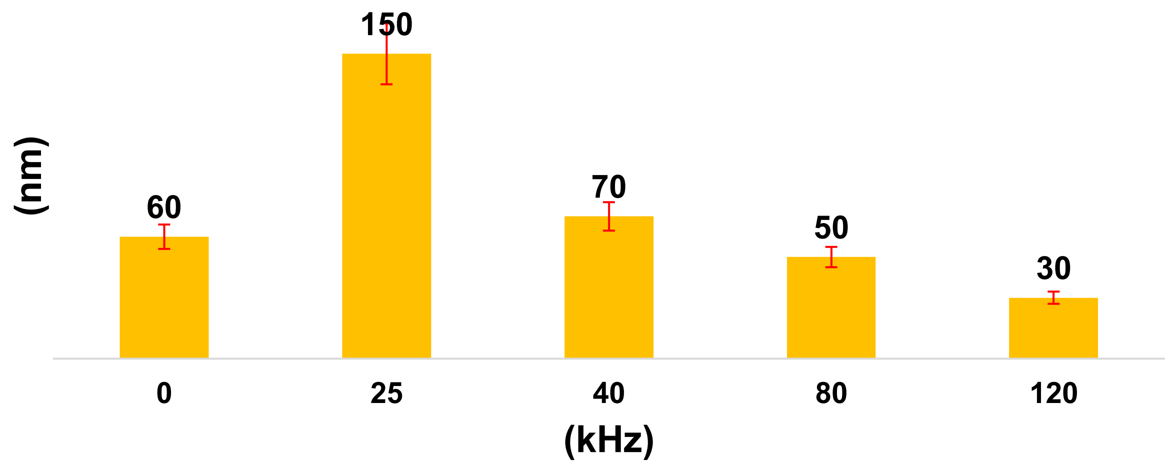


Fig 4.2: Image analysis performed on the SEM images of gold nanostars, after 20 min of sonication at 25 kHz to detect the fused cores of the gold nanostars.

A series of control experiments were carried out with the growth solutions, by using different sonication frequencies. The average particle size of the AuNSs in the growth solutions varied with the change in sonication frequency due to the formation of the dumbbell shaped nanostars as shown in Fig 4.3. As indicated by the SEM observations, the average size of the AuNSs was 50 nm, without the sonication treatment, and increased to 110 nm, when the solution was sonicated at a frequency of 25 kHz as shown in Fig 4.3 (at least 50 particles were measured to calculate the average sizes). UV-visible spectra were measured to characterize the gold nanostars, subjected to different sonication frequencies as shown in Fig 4.3. When the 25 kHz of sonication frequency was used, the LSPR band of the AuNSs experienced a gradual red-shift from 720 nm to 840 nm, corresponding to a color change, from pale blue to blue-green. When the sonication frequency was increased from 25 kHz to 120 kHz, a gradual blue shift from 840 nm to 660 nm was observed.



(a)



(b)

Fig 4.3: (a) Au LSPR band of gold nanostars as a function of the ultrasound frequency. **(b)** Average size (core + branches) of gold nanostars as a function of the sonication frequency.

Fig 4.3b shows that the size of AuNSs is critically dependent on the ultrasonic frequency. When the frequency was 25 kHz, AuNSs were observed to grow to almost 150 nm, by fusing of multiple gold nanoparticles. However, as the sonication frequency increased from 25 kHz to 120 kHz, a gradual reduction in size of the AuNSs can be observed. A possible explanation would be that, increasing the sonication frequency, the reduction process is accelerated, leading to an overall smaller number of ions in the solution, and therefore, a lower ionic strength at which aggregation would not occur.

Fig 4.4 shows the effect of ultrasonic energy (sonication time) on the size of AuNSs, by using a conventional ultrasonic bath with a constant frequency (25 kHz).

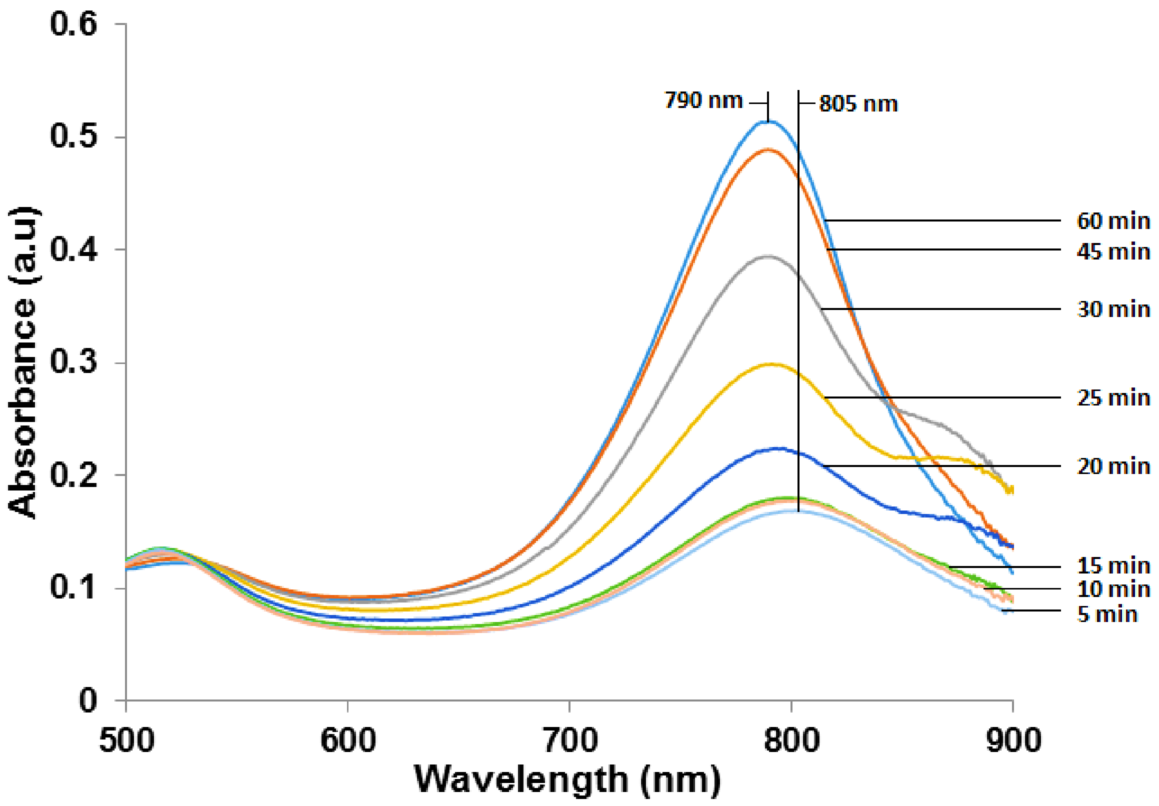
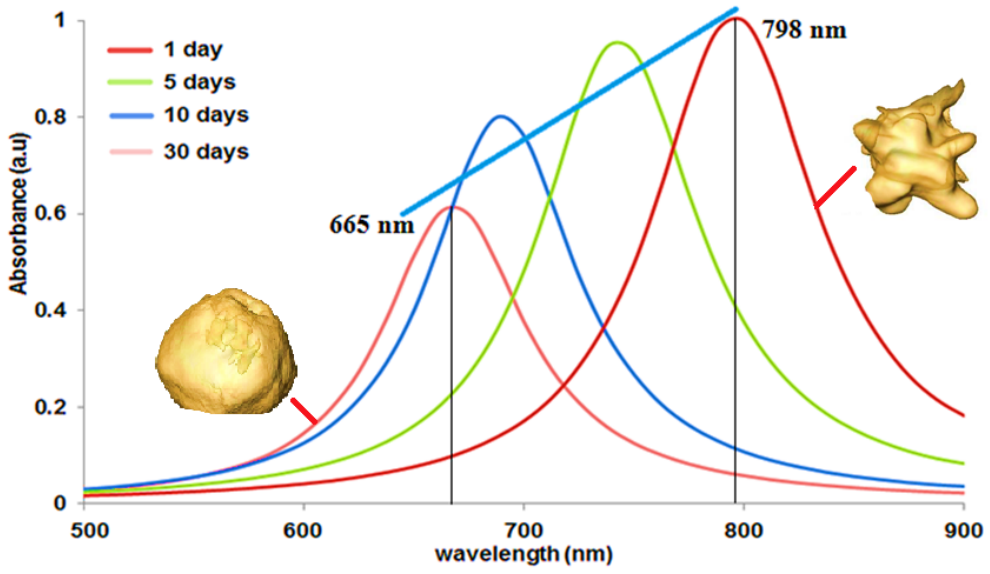


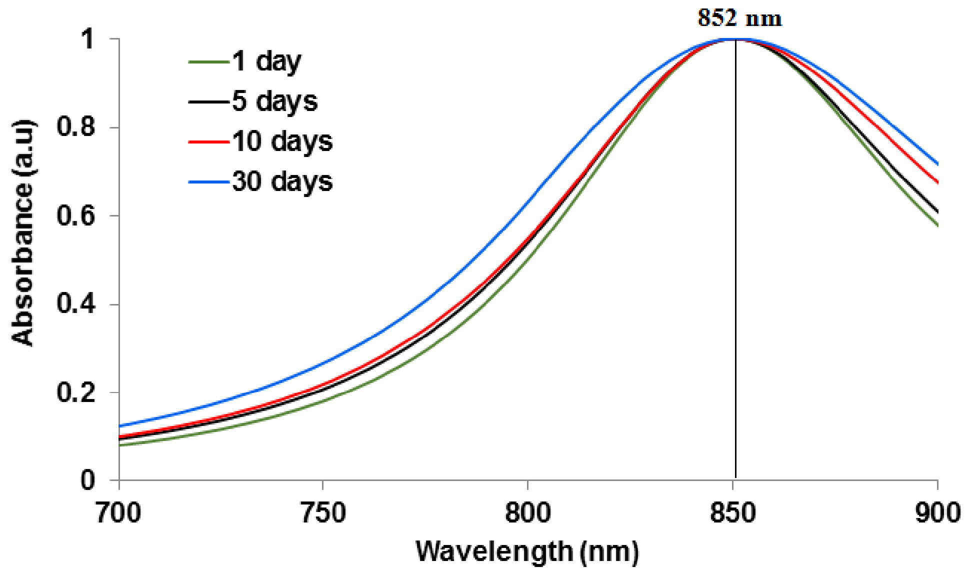
Fig 4.4: Temporal evolution of UV–Vis absorption spectra of AuNSs samples at 25 kHz

Fig 4.4 shows that, at a given frequency, the size of nanoparticles is not significantly dependent on the sonication time. The spectra indicated a blue shifted LSPR band, from 805 nm to 790 nm, showing that the prolonged exposure to ultrasounds, results in the formation of smaller particles. The sonication of an aqueous solution for a prolonged period generates a higher number of cavitation bubbles. It is possible that, at higher acoustic power levels, a greater number of radicals are generated per bubble, due to the larger size (R_{max}) of the cavitation bubbles. [89, 90]. That makes it impossible to control the growth stage of the crystallization process, leading to an uncontrolled and rapid reduction. Even though there would be larger AuNSs present, this would be for an extremely short period, and, at the end of the sonication process, the solution would contain only small particles.

The stability of the as synthesized AuNSs was studied by comparing the effect of sonication on the optical and morphological features of the AuNSs. Fig 4.5a and 5b show the LSPR band of the AuNSs with, and without, sonochemical treatment. Fig 4.5a shows that, without sonication, a blue shift from 810 nm to 670 nm is observed after only three days, showing that the untreated AuNSs undergo aging, as indicated by the strong blue shift of the LSPR band. Fig 4.5b shows that sonicated AuNSs are stable. The sonochemical treated AuNSs show a significant improvement of stability, possibly, due to the increase in the size of AuNSs that facilitates the increase in the aggregation barrier.



(a)



(b)

Fig 4.5: (a) Temporal evolution of UV–Vis absorption spectra of AuNSs samples, without sonication treatment (until 30 days). **(b)** Temporal evolution of the UV–Vis absorption spectra of AuNSs samples sonicated at 25 kHz.

A direct comparison of yields between different synthesis protocols is difficult because of the lack of available analytical techniques [91, 92]. However, based on the LSPR results, it can be assumed that the AuNSs synthesis, followed by ultrasonication treatment, led to well-formed nanostars, with a yield of about 14% [67]. Therefore, it is reasonable to say that the suspended gold nanostars are stable for more than one month. Because of the sonication, nanostars are larger, the surface plasmon resonance mode is shifted to the near infrared which makes them more suitable for biological applications.

4.4 CONCLUSION

In this study, it was demonstrated for the first time that the sonication frequency has a significant effect on the size and morphology of gold nanostars. The formation and stability of the gold AuNSs treated ultrasonically can be summarized as follows. At the initial stage of the reaction, many the small gold particles with a diameter around 30 nm are produced by reduction of gold nanoparticles ions with a sodium citrate solution. The seed-mediated method, followed by the sonication treatment developed in this study, offers new opportunities to synthesize aqueous suspensions of monodispersed gold nanostars with prolonged stability. The result indicates that at a sonication frequency of 25 kHz, the maximum average particle size of 120 nm is attained, due to the large diameter of the cavitation bubble which, under collapse, generates extreme conditions of pressure and temperature. Therefore, the LSPR extinction spectra of the gold nanostars, centered within the range of 650 to 900 nm, is tunable via the change in the ultrasound frequency. The optical characteristics of gold nanostars solution after ultrasound treatment appeared to be substantially more stable than those of non-treated ones because of the slower aggregation of NSs. The LSPR extinction spectra indicates that there is no further shift in the LSPR band after sonicating the solution at 25 kHz beyond 20 min, indicating an exponential relation between the optical property of gold nanostars and the duration of the ultrasound treatment. The use of sonication technique to stabilize gold nanostars, without any additional chemical stabilizer, is both technologically and scientifically important. It can, hence, be expected to be able to produce by this method, a large volume of consistent quality gold nanostars.

CHAPTER 5

Sensitivity and Stability enhancement of Poly (vinyl alcohol)-nanostars composites for Localized Surface Plasmon Resonance sensing applications

5.1 Introduction

Recently, water-soluble PVA–nanoparticle composite materials are of great interest in terms of their potential applications in biomedical, electronic, and optical materials due to their hybrid properties derived from several components [93]. Whether in solution or bulk, these polymer composites offer a unique response to stimuli (pH, temperature, ionic strength, light and force). Such characteristics are induced by the interactions of the nanoparticles and the polymer [94]. Such frameworks have been broadly examined over the previous decade following synthetic advances [95]. Although there are numerous cases of ligand-adjustment of nanoparticles, there are few examples of polymer-stabilized metal nanoparticles [96].

In general, we note two diverse methodologies used to date. The first procedure comprises of the in-situ preparation of the nanoparticles in the matrix. This is implemented either by dissolving the metal salts in the polymer matrix [97] or by the dissipation of metals on the heated polymer surface [98]. The second method is a less common procedure which involves polymerizing the matrix around the nanoparticles [99]. An improved methodology would include adapting first and second methods by blending of pre-synthesized nanoparticles into the polymer matrix. This gives full synthetic control over both the nanoparticles and the network and has the potential for creating a wide assortment of composite materials.

Among a variety of possible nanoparticle morphologies, colloidal gold nanostars (AuNS) comprising a central core from which multiple sharp branches protrude, have been noted as highly efficient surface-enhanced Raman scattering (SERS) substrates due to their tun- able localized surface plasmon resonances (LSPR) and nanoantenna effects by controlling core size, along with tip length and sharpness. [100 – 106] However, these AuNS typically display poor stability, resulting in changes in the particle's morphology (reshaping) and corresponding LSPR blueshifts, which has also been reported for other nanostructures with sharp tips. [107, 108]

So far, the sensing experiments with gold-polymer platforms have been performed by using mostly composites of gold nanospheres and PDMS polymer [97]. The refractive index sensitivity found by Shiohara et al. was lower than that previously found by our group for nanostars embedded in porous PDMS [98]. In addition, because of the strong hydrophobicity of PDMS, more appropriate PDMS surface modifications are needed to improve the surface wettability facilitating fluid delivery. It is well known that, even after a plasma treatment, PDMS will slowly revert to its hydrophobic character [99].

To overcome these difficulties, we reasoned that it is necessary to design the gold nanostars to make them compatible with the polymer matrix. We speculated that gold nanostars whose polymer ligand is chemically the same as the matrix would be more thermodynamically favorable to their incorporation [100].

In this work, considerable efforts have been devoted to developing a high sensitive Localized Surface Plasmon Resonance (LSPR) based platform by the synthesis of novel gold nanostars synthesized using seed method assisted by sonochemical method and their successful embedding them into the pores of the hydrophilic Poly vinyl alcohol (PVA) polymer matrix. The effect of sonication frequency and temperature on Localized Surface Plasmon Resonance (LSPR) of AuNSs-PVA composite is also investigated here.

5.2 Materials and Methods

5.2.1 Materials

The hydrochloric acid solution ($HAuCl_4$), silver nitrate ($AgNO_3$), sodium citrate ($Na_3C_6H_5O_7$), ascorbic acid ($C_6H_8O_6$) and PVA (C_2H_4O) (MW 89,000-98,000 g/mol) were purchased from Sigma-Aldrich. Gold chloride (hydrogen tetrachloroaurate) ($HAuCl_4 \cdot 4H_2O$) was acquired from Alfa Aesar Chemical. Sonication irradiation was carried out using a horn type sonicator (Type: CCH5838D-25LB PZT-4, frequency: 25 kHz, diameter of tip: 19 mm, Resonance Impedance (Ω): 10 – 25, Static Capacity (pF) ± 10%: 5400, Input power (Watt): 100W). A Perkin Elmer Lambda 650 UV-Vis spectrometer was used for all the spectral measurements.

5.2.2 Preparation of the Poly- (Vinyl Alcohol) (PVA) solution

The PVA aqueous solution was prepared by slowly adding 4 g of PVA powder in 100 ml of cold water to avoid formations of lumps, and heated. At a temperature ranging from 85˚C to 98˚C until the PVA is fully dissolved. The polymer solution is then further heated and held at 115°C for 15 min to remove air bubbles. Once cooled to room temperature, the solution was stored for further usage.

5.2.3 Preparation of gold nanostars- PVA composite

The gold nanostar solution was synthesized using seed-mediated method [88] protocol was extended further to enable size control and tuning capabilities of the longitudinal plasmon peak in the NIR region from around 655 to over 750 nm of the stars. The nanostars solution sonicated at 25 kHz for 20 min. The solution is then stored at 4 °C to stop the aggregation process and allow the growth of the branches.

The gold nanostars - PVA composite was prepared by adding10 mL of gold nanostars solution to 100 mL PVA aqueous solution. The mixture was stirred vigorously and then subjected to vacuum for removing the traces of air from the sample. Finally, the homogenous solution was poured into a mold and let it dry at room temperature. The schematic, showing the process of the preparation of composite and embedding of the AuNSs aggregates into the pore network of the PVA polymer is shown in Fig 5.1.

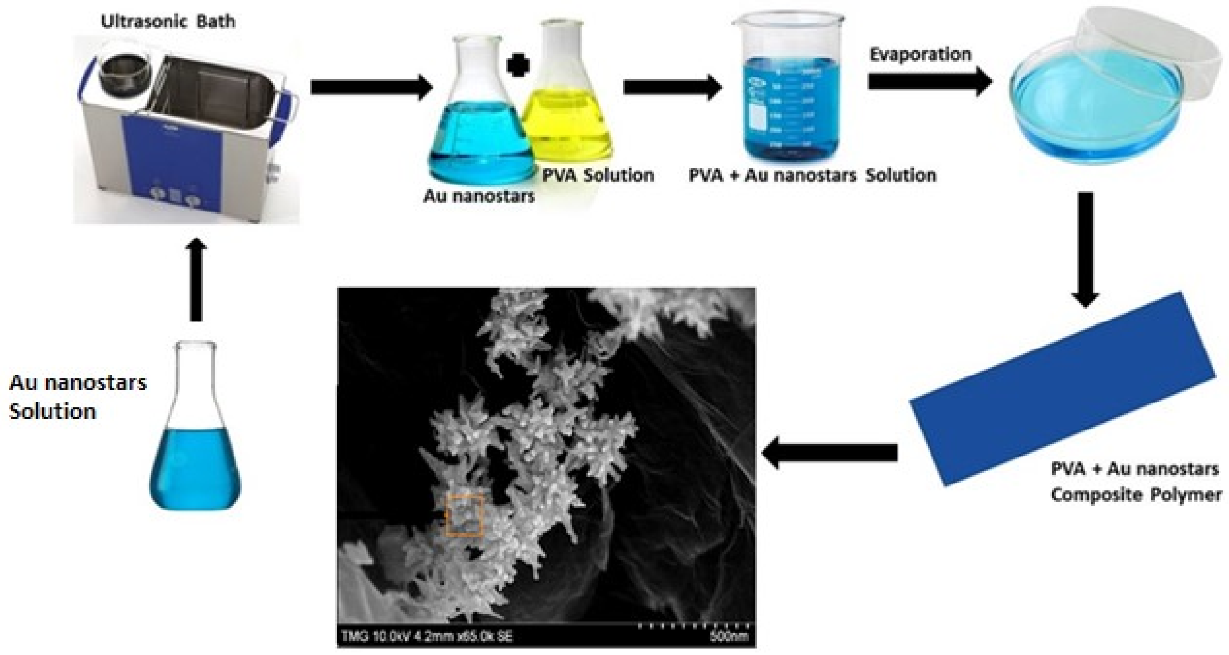

Ultrasonic Bath
Au nanostars
PVA Solution
PVA + Au nanostars Solution
Evaporation
Au nanostars Solution
PVA + Au nanostars Composite Polymer
TMG 10.0kV 4.2mm x65.0k SE
500nm

Fig 5.1: Schematic showing the preparation of gold nanostars-PVA composite platform

5.3 Refractive Index Sensitivity Measurement

The AuNS-PVA nanocomposite samples are introduced into different dielectric media of varying refractive indices as listed in Table I. The sensitivity of the composite to the surrounding dielectric medium is calculated as the ratio of the Au-LSPR band shift (Δλ) between the solvent and de-ionized water to the change in the refractive induces (Δn) of the solvent to de-ionized water [102]:

Sensitivity (σ) = (Δλ)/Δn **[Eq. 5.1]**

$$\sigma = (\Delta\lambda)/\Delta n \quad \text{[Eq. 5.1]}$$

Where

$$\Delta\lambda = \lambda_S - \lambda_W \quad \text{[Eq. 5.2]}$$

$$\Delta n = n_S - n_W \quad \text{[Eq. 5.3]}$$

Where, Δλ is the difference in the plasmonic shift, λ_S is the peak wavelength of the Au LSPR band where the sample is immersed in a solvent of a known refractive index and λ_W is the position of the band of the sample immersed in de-ionized water, Δn is the difference between the refractive indices of the solvent and that of DI water, n_s is the refractive index of the solvent, and n_w is the refractive index of water.

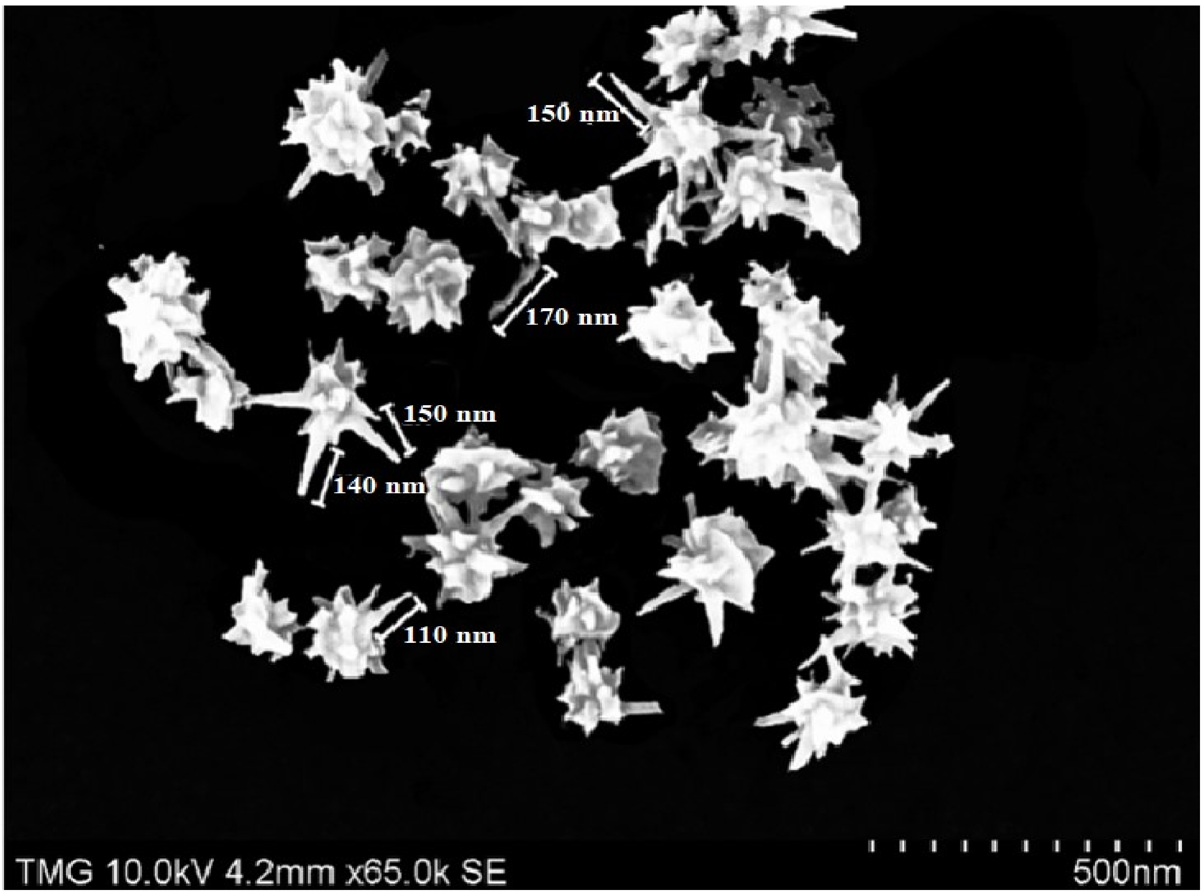
Table 5.1 Refractive indices of dielectric media used for sensitivity tests at 25 °C

Media	**Refractive Index (n)**
Deionized Water	1.330
Ethanol	1.361
Acetone	1.360

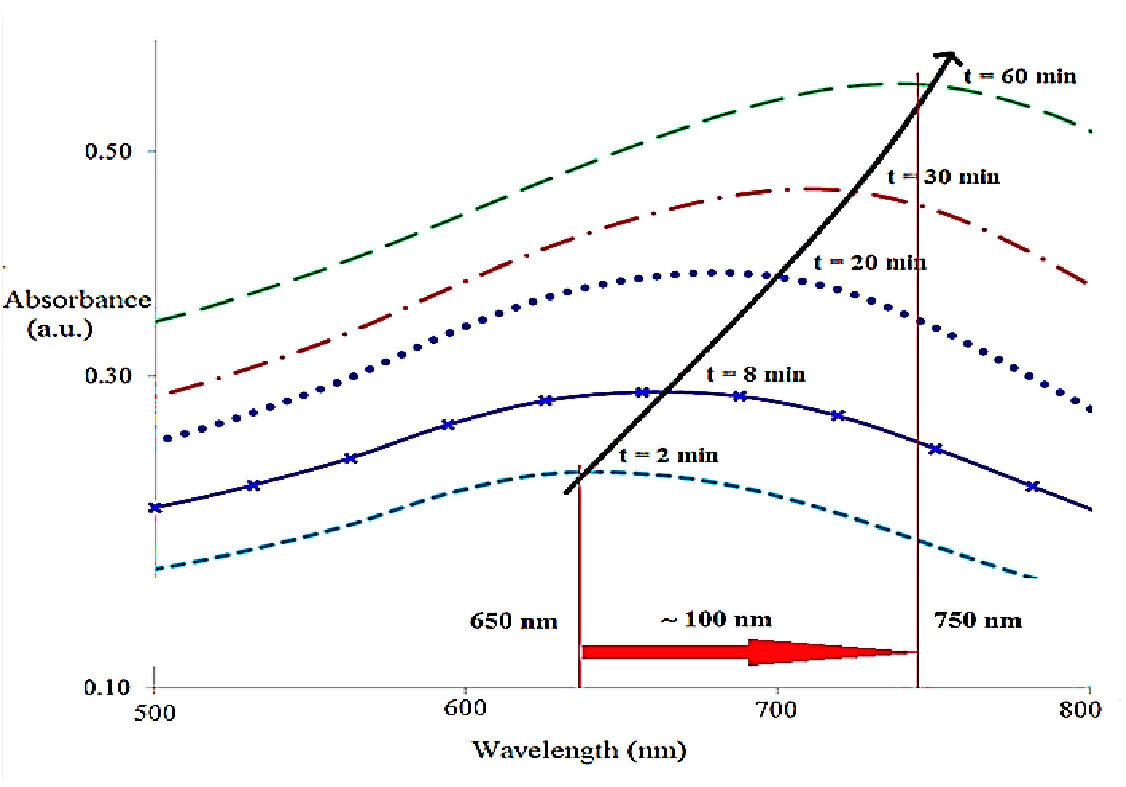
N,N-Dimethylformamide	1.430
Dimethylacetamide	1.437
Chloroform	1.440

5.4 Results

5.4.1 Temporal evolution of gold nanostars during synthesis.

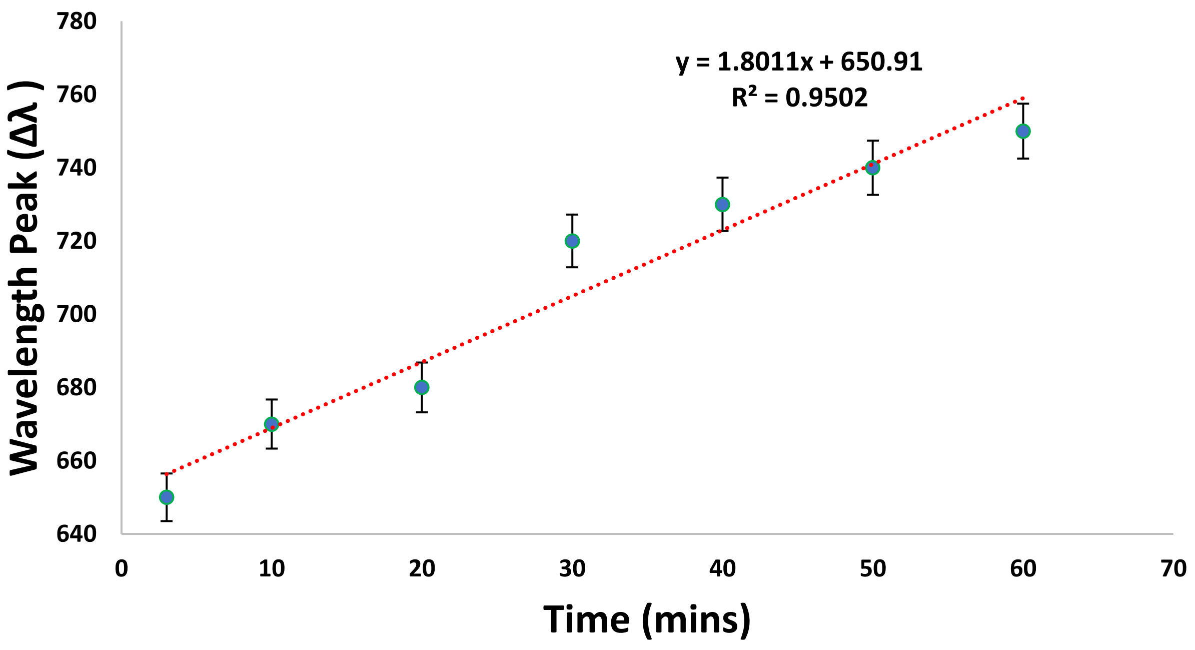


(a)



t = 60 min
t = 30 min
t = 20 min
t = 8 min
t = 2 min
0.50
Absorbance (a.u.)
0.30
0.10
650 nm
~ 100 nm
750 nm
500
600
700
800
Wavelength (nm)

(b)



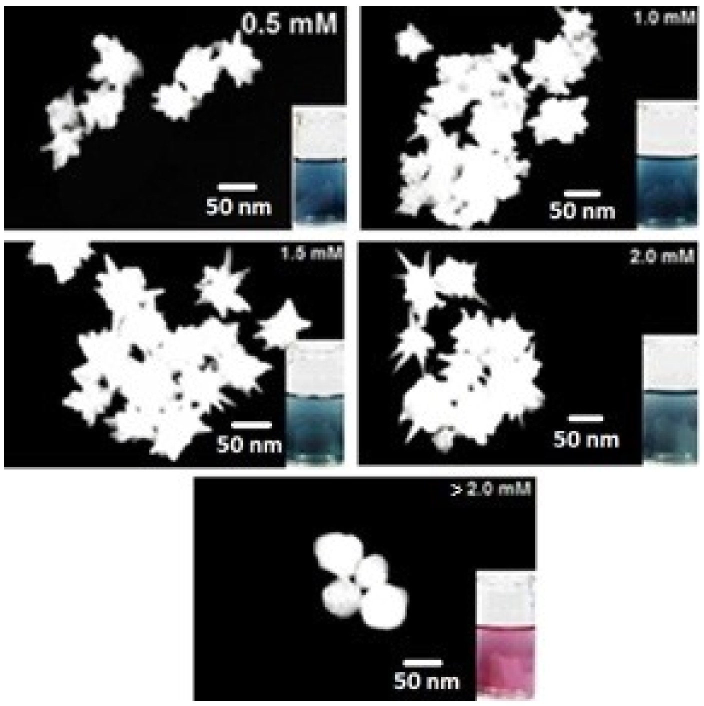
y = 1.8011x + 650.91
R² = 0.9502
Wavelength Peak (Δλ)
780
760
740
720
700
680
660
640
0
10
20
30
40
50
60
70
Time (mins)

(c)

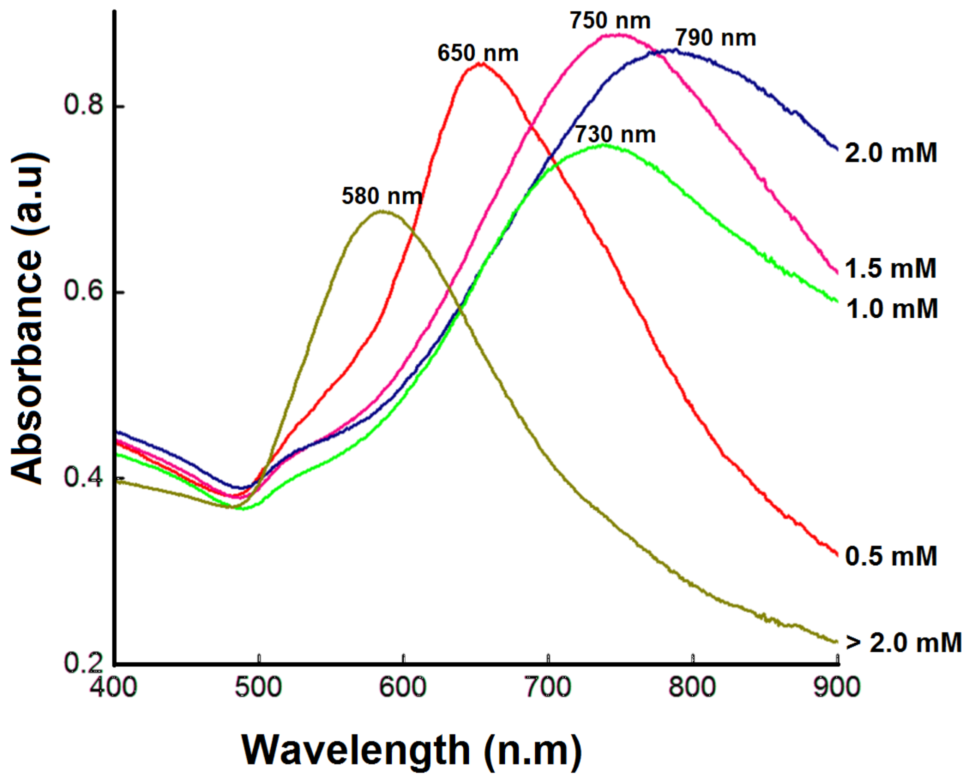
Fig 5.2: (a) SEM image of the ultrasound-stabilized gold nanostars deposited on a glass slide from their aqueous solution. **(b)** Spectral study of the growth of nanostars with time (t = 2 min to t = 60 min (The concentration of silver nitrate used in this experiment was 2 mM), (Sample set, n = 10 samples) **(c)** Change in LSPR peak wavelength as a function of the time.

The growth mechanism of branches in a nanostars is important to establish a relation between morphology of the stars and optical resonance. The yield of long-branched nanostars is very high, as shown by SEM image as shown in Fig 5.2a. The size of the nanostars ranges from 100 nm to 250 nm, and the number of tips varies from 15 to 20. To investigate the kinetics of formation of stars, the solution was sampled at specific time intervals, and the spectra were recorded as shown in Fig 5.2b which represents the location of the peak of plasmon band as the spectra evolves in time, suggesting that the branches grows continuously until synthesis completion, which occurs approximately around 60 min. The non-linear growth of the branches can be inferred from the non-linear LSPR shift with time. At the initiation of growth plasmon band appears around λ = 650 nm very early in the synthesis, at t = 2 min, and rapidly red-shifts to approximately λ = 700 nm at t = 20min, following which it slowly progresses further towards the NIR, reaching λ=750 nm at t = 60 min. The evolution of the plasmonic bands implies that the AuNSs branches start growing from the core surface shortly following nucleation and increase in length rapidly in the first 8 min as shown Figure 5.1b. The process is followed by further nucleation and growth of the branches. The core and branches grow simultaneously throughout the synthesis. Their fastest growth rate corresponds to the beginning of the synthesis, followed by a steady growth until synthesis saturation [103].

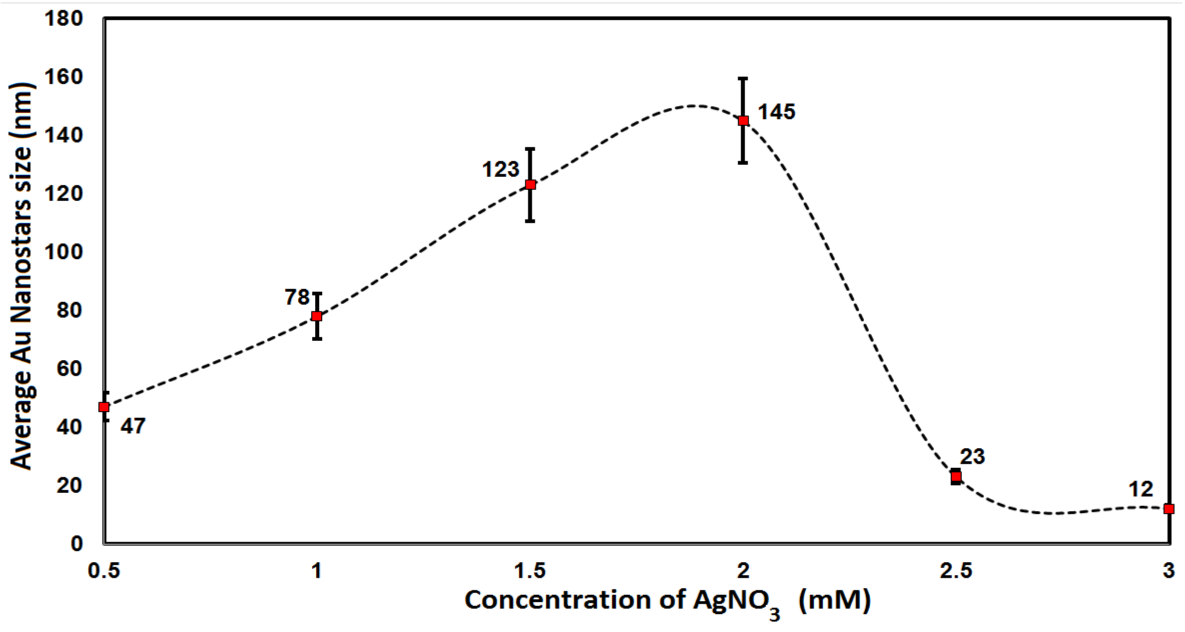
The morphology of the Au nanostars depends significantly on the nucleation kinetics, controlled by the concentration of silver nitrate ($AgNO_3$). In the growth solution, ascorbic acid acts as a mild reducing agent for silver ions. The silver atoms, formed by reduction, will deposit on the gold seeds and become the starting points of the branches.



(a)



(b)



(c)

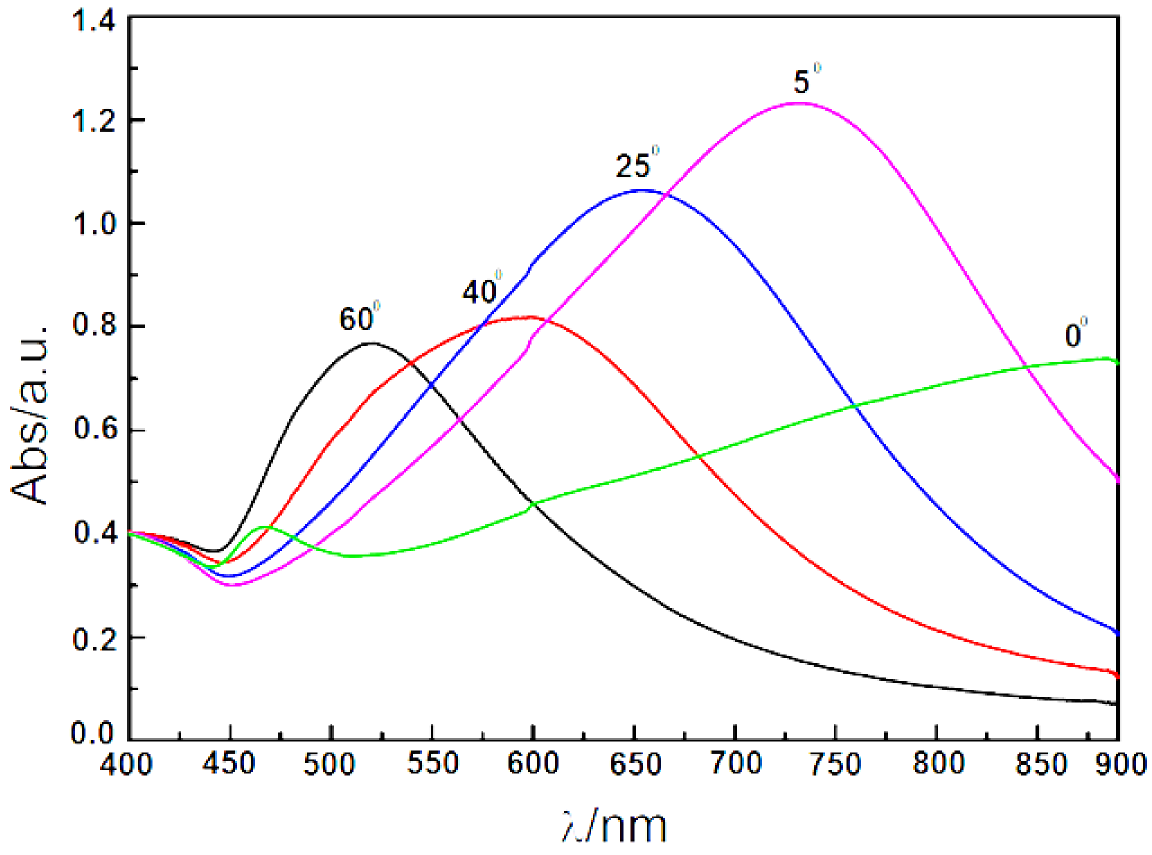
Fig 5.3: (a) SEM images of the Au nanostars corresponding to different concentrations of $AgNO_3$. **(b)** Tuning the absorption of the LSPD band of nanostars in the presence of silver nitrate solutions

of various concentrations. **(c)** Average Au nanostars size (nm) vs concentration of AgNO3 (mM). (Sample set, n = 10 samples, SD = 4)

As seen in Fig 5.3, the concentration of the $AgNO_3$ solution used for the synthesis plays a significant role in the branch formation [104]. The figure shows that the morphology of AuNSs can be controlled by varying the concentration of the $AgNO_3$ solution as shown in Fig 5.3a. Increasing the concentration of $AgNO_3$ from 0.5mM to 2mM, a gradual shift in the position of the nanostar's plasmon band, from 650 nm to780 nm, is noticed but, if the concentration of the $AgNO_3$ increases over 2 mM, the LSPD band shifts back from 780 nm to 590 nm. The evolution of the plasmonic bands implies that a careful tuning of Ag^+ concentration is important for the generation of branches. AuNSs with high Ag+ concentration (>2 mM) led to no branches on the surface of AuNP with significant nonspecific growth on the tip body. Solution of low $AgNO_3$ concentration (< 0.5 mM) led to very short branch (length: 15 – 20 nm). We found that the medium concentration of $AgNO_3$ (2 mM) yielded nanostars with sufficiently long branches (length = 100 – 150 nm), as shown in the Fig 5.3b and Fig 5.3c. The possibility to tune the Au plasmon band by changing the concentration of a reagent is important and necessary for a significant number of biological and energy applications.

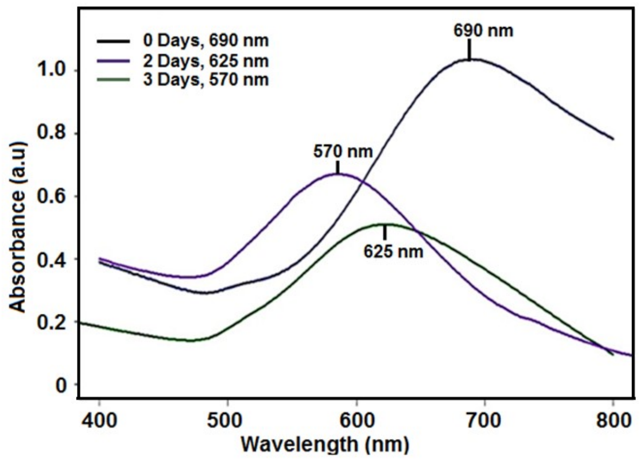
5.4.2 Effect of ultrasounds and temperature on growth kinetics and stabilization

Au nanoparticles (AuNPs) have the general tendency to form large clusters to minimize their surface energy, commonly referred to as aggregates, thereby often losing their nanoscale properties. It is assumed that energy is transform during bubble growth and collapse in sonication reduction which helps to supply energy required for AuNSs synthesis [105].

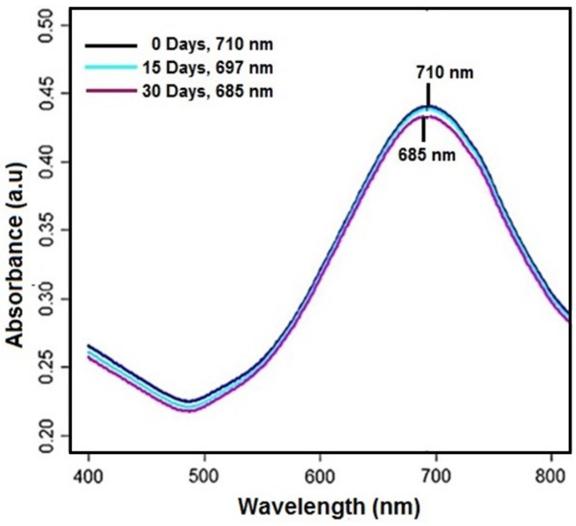


Abs/a.u.
λ/nm
60⁰
40⁰
25⁰
5⁰
0⁰

(a)

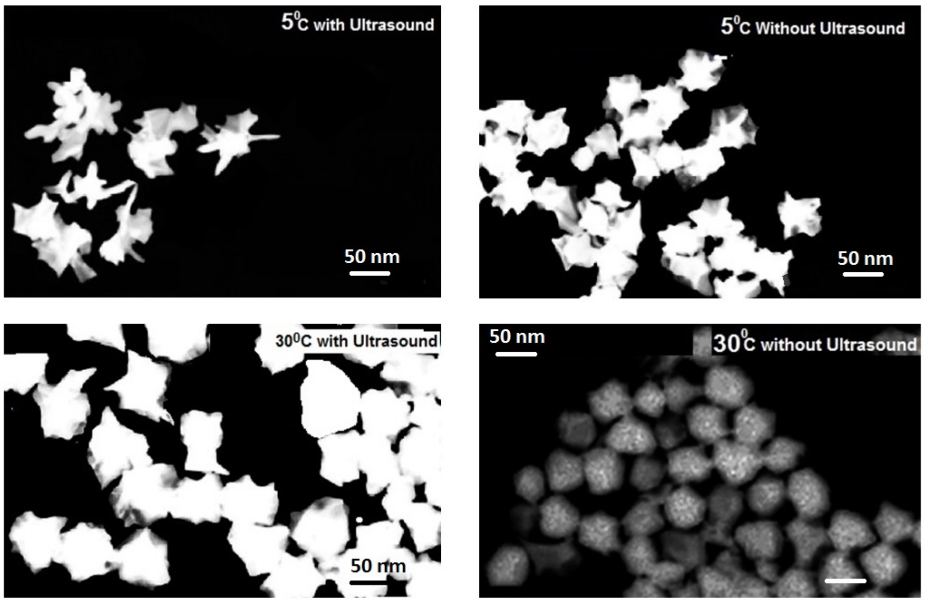


0 Days, 690 nm
2 Days, 625 nm
3 Days, 570 nm
690 nm
570 nm
625 nm
Absorbance (a.u)
Wavelength (nm)



0 Days, 710 nm
15 Days, 697 nm
30 Days, 685 nm
710 nm
685 nm
Absorbance (a.u)
Wavelength (nm)

(b) **(c)**



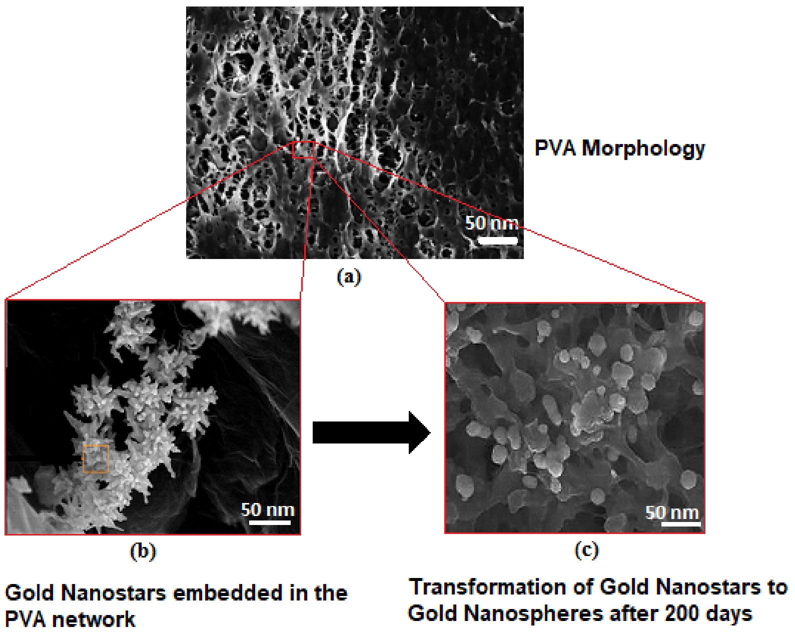
Temperature (⁰C)	Gold nanostars average size (nm)	
	Without sonication	Sonication (@25kHz)
5	75-90	125-150
30	45-60	70-100

(d)

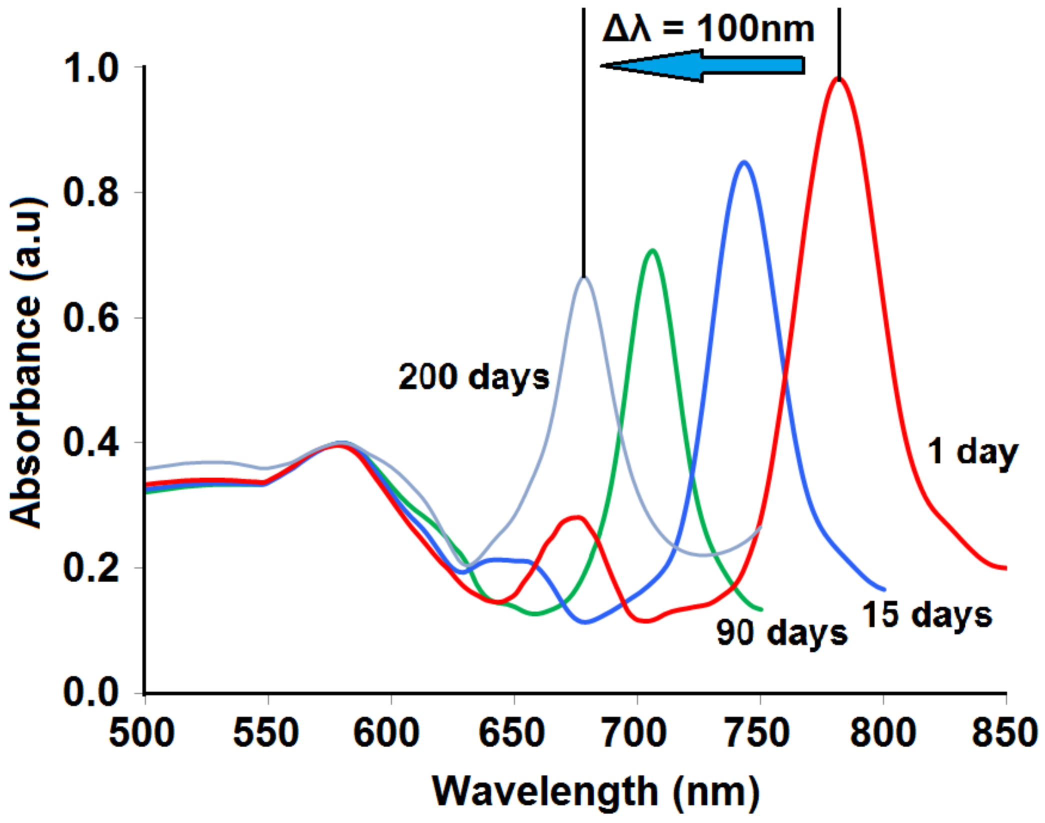
Fig 5.4: (a) Visible-NIR spectra of Au nanostars as a function of the post-synthesis temperature at a constant ultrasound frequency of 20 kHz. **(b)** Absorbance spectra of nanostars, recorded on the same day, after two days, and after 3days without using ultrasounds **(c)** Absorbance spectra of nanostars recorded on day 0, after 6 days, and after 30 days after using ultrasounds. **(d)** SEM image of the Au nanostars obtained at different temperatures, as indicated on the label, at constant ultrasound frequency of 20 kHz and the effect of ultrasound and temperature on the branch length. Data points (n) = 6

The AuNSs were synthesized at 25 °C and series of experiments were performed by changing the temperature of the AuNSs solution and with and without sonication treatment. To determine the effect of sonication assisted by temperature, the AuNSs solution was sonicated at 25kHz and stored at various temperatures and the effect was determined by visible-NIR absorbance spectra. The AuNSs LSPR was found to be shifted from 530 nm to 740 nm when the temperature decreased from 60 ºC to 5 ºC as showed in Fig 5.4a. Fig 5.4b and Fig 5.4c demonstrate the effect of sonication on of AuNSs using visible-NIR spectra. Herein, the Sonication-treated AuNSs showed a blue shift in absorption band from 710 nm to only 685 nm over 30 days, whereas for the non-treated AuNSs the blue shift in absorption band was from 690 nm to 570 nm after 3 days. At the same time, the color of the sample turned to purple showing the transformation of nanostars into nanospheres. The effect of sonication and temperature on morphology of AuNSs is showed by Fig 5.4d. The AuNSs stored at 5^0C were highly anisotropic, with relatively long branches and planar appendices, when compared to AuNSs synthesized at 30^0C, which seem to have shorter branches.

5.4.3 Formation of the Nanostars – PVA polymer composite



PVA Morphology
50 nm
(a)
50 nm
50 nm
(b)
(c)
Gold Nanostars embedded in the PVA network
Transformation of Gold Nanostars to Gold Nanospheres after 200 days



(d)

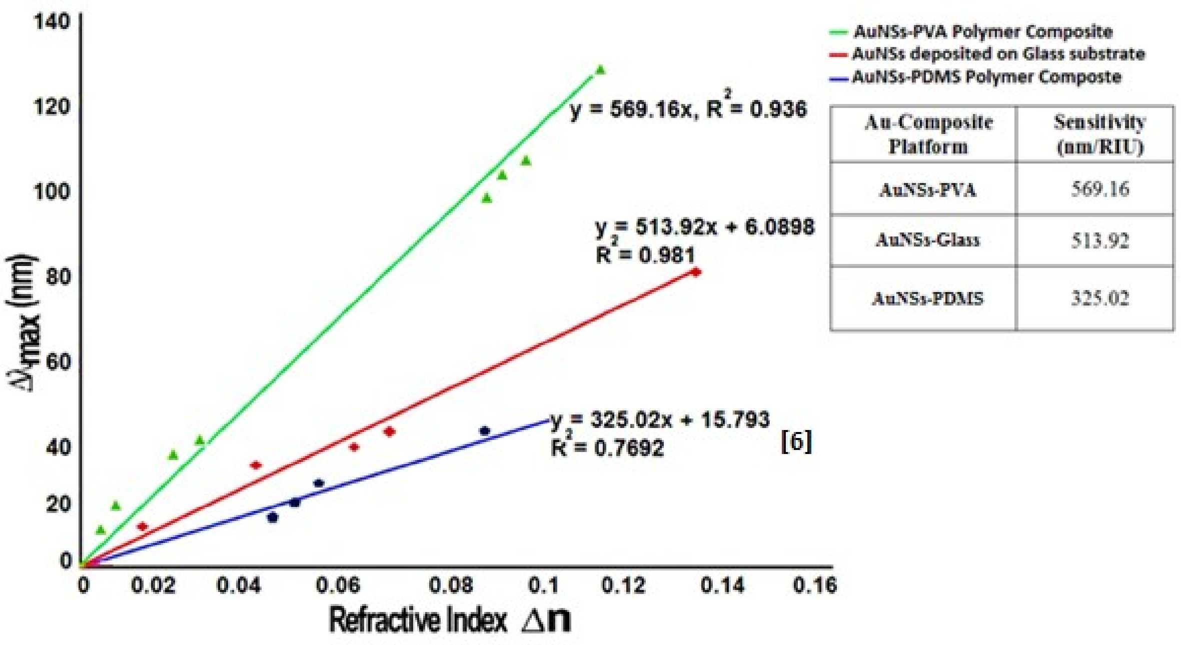
Fig 5.5: (a) SEM image of the PVA polymer network. **(b)** SEM image of Au nanostars aggregates embedded in the PVA matrix. **(c)** The slow transformation of nanostars to nanospheres. **(d)** Spectra of Au nanostars – PVA composite over a period of 200 days showing the stability of the embedded nanostars aggregates. (Sample set (n) = 6)

Investigation of PVA polymer film microstructure indicated that the polymer has an internal 3D network with a large number of micropores throughout the polymer matrix. Fig 5.5a shows a network structure that can act as a host for the Au nanostar aggregates shown in the Fig 5.5b. The LSPR spectra of the corresponding AuNSs – PVA polymer composite over time is shown in Fig 5.5c. AuNSs – PVA polymer composite film contains the well dispersed nanostars in polymer matrix. Considering the porous framework of the polymer as well as the method we used to incorporate the nanostars, the resultant composite can be considered a host-guest inclusion complex. A gradual blue shift is seen in the spectra of Au nanostars – PVA composite as shown in Fig 5.5d. The shift has been accounted by the very slow transformation of the nanostars into the more thermodynamically stable nanospheres over a year as seen in the Fig 5.5b.

5.5 Chemical Sensing of Surrounding Environment

The LSPR is extremely sensitive to the environment surrounding the AuNPs. Therefore, the LSPR extinction curve is monitored as a function of changes in the local dielectric environment which is related to the refractive index. This method can also be applied to sensing of biological molecules such as proteins and antibodies. The refractive-index sensitivity of the AuNSs LSPR was demonstrated by exposing the AuNSs-PVA polymer to different solvents having refractive indices listed in Table I. The change in LSPR band ($\Delta\lambda$) versus refractive index of the local environment (Δn), for a solvent is plotted in Fig. 5.6, demonstrating a linear relationship between ($\Delta\lambda$) and (Δn).

The AuNSs-Glass platform was prepared by depositing 100 µL of AuNSs on the glass substrate and letting it dry at 25 °C. For AuNSs-PDMS platform, the PDMS Elastomer was mixed thoroughly with the curing agent in the weight ratio of 10:1 and then degassed under vacuum to remove entrapped air bubbles. It was then casted on glass slides and cured under vacuum at room temperature. The mixture was then incubated at 60 °C for 12 hours for the polymer to form. The PDMS polymer is then dipped in the AuNSs solution for 3 hrs and dried at 25 °C. Overall, the refractive-index sensitivity for AuNSs-PVA polymer composite was found to be 569 (nm/RIU) which is higher refractive-index sensitivity than the other platforms as shown in Fig 5.6.



Au-Composite Platform	Sensitivity (nm/RIU)
AuNSs-PVA	569.16
AuNSs-Glass	513.92
AuNSs-PDMS	325.02

Fig 5.6: Sensitivity of the AuNSs on different composite platform (Sample set (n) = 6)

It is assumed that during the sensing process, the dielectric medium (the solvent) enters the porous network of the polymer and reaches the long branches of the AuNSs. Therefore, the overall sensitivity of AuNSs-PVA polymer is significantly higher indicating that the EM field extends far enough, away from the particle surface to be useful for sensing experiments.

5.6 Conclusion

Gold nanostars have been prepared through a seed-assisted method and stabilized by controlling the temperature after the ultrasound treatment. It is found that if the nanostars stored at 5 0C after the ultrasound treatment, their stability can be extended from 2-3 days to 30 days. The concentration of silver nitrate was optimized to increase in the branch length. The morphology of the particles was assessed by SEM, while the optical properties were characterized with UV/vis spectroscopy. Subsequently, a host-guest complex has been prepared, by embedding the stabilized Au nanostars into the porous network of the hydrated PVA. Due to the strong van der Waals interactions between individual stars, they form aggregates inside the pores. It is assumed that during the sensing process, the solvent enters the PVA network and contacts the long branches of the stars. In the case of sensing with Au NS – PVA composites, the sensing entities are not the individual stars but their aggregates. Further investigations are necessary to understand the fundamental mechanism of plasmonic sensing through aggregates.

CHAPTER 6

Gold Nanostars polymer composite platforms for Localized Surface Plasmon Resonance Based Early Cancer Diagnosis

6.1 Introduction

Exosomes are vesicles of sizes ranging from 30 to 100 nm, with an important role in cell communication, predominant for the maturation of tumour microenvironment and progression of cancer [106]. ExoCarta, an exosome database, highlights the contents identified in multiple exosomes organisms [107], over 4900 mRNAs, 41,800 proteins, and 2800 miRNAs [108], with sites in several subcellular compartments [109]. The specific composition of exosomes appears to depend on the type of cell or tissue and may differ depending on the physiological state as shown in Fig 6.1 [110, 111, 112].

The extraordinary features of exosomes are as follows: (i) the composition varies according to the original cell and exosomes derived from cancer cells, usually reflects the tumor stage [113, 114, 115], (ii) exosomes are stable in circulation, they are found in body fluids, including blood, saliva, breast milk, and urine, indicating that circulating exosomes can be a biological marker, suitable for diagnosis and prognosis of cancer [116], (iii) they can change the phenotype of a recipient cell, responsible for the maturation of the tumor microenvironment and progression of cancer [117, 118]. Understanding the role of exosomes in tumorigenesis creates a new era in the diagnosis and treatment of cancer, using exosomes circulating as tumour biomarkers [119, 120] and targeting exosomes derived from cancer cells [121].

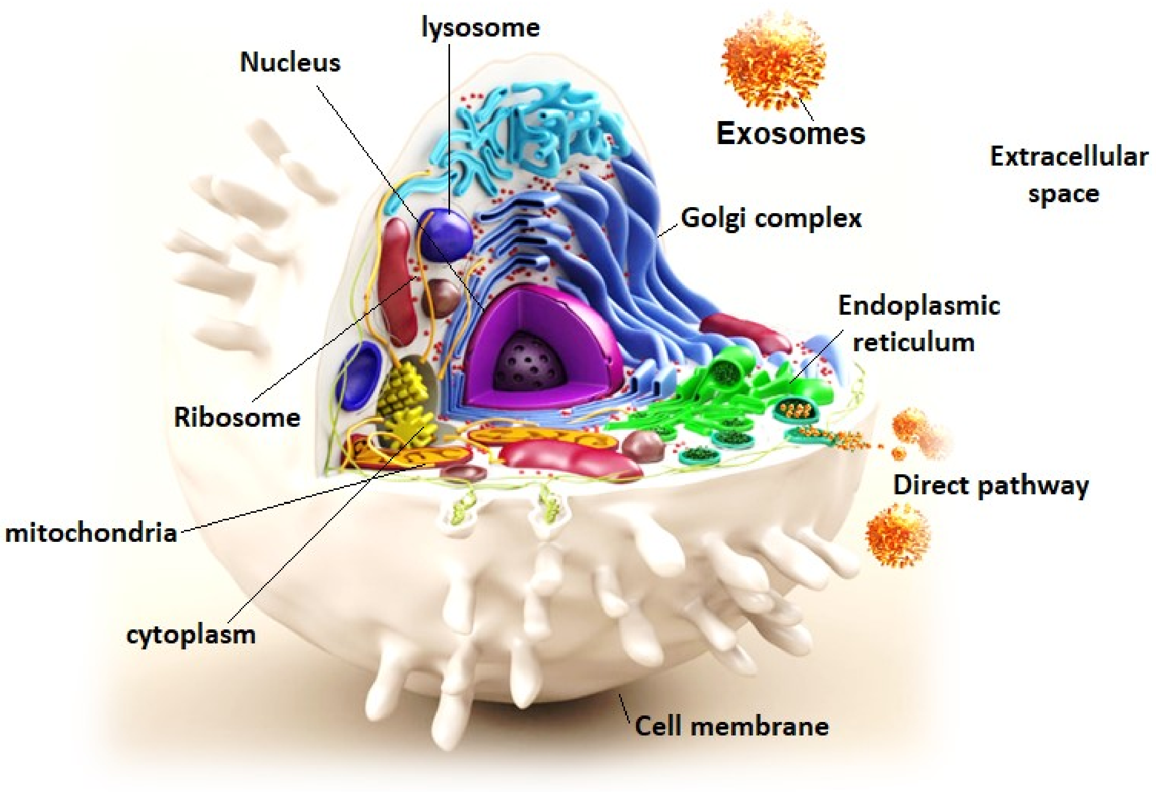


Fig 6.1: Biogenesis of exosomes into the extracellular space

Over the past few years, nanotechnology has paved the way for the development of a plethora of new diagnostic and therapeutic platforms [122]. Examples of Nano carriers are liposomes, polymeric nanoparticles, viral vectors, and more recently gold nanoparticles [123]. Gold nanoparticles (AuNP) have unique physical and optical properties, making them a powerful tool for diagnosis, and therapy [124, 125]. AuNPs can be modulated in shape, and composition, as well as their size (1-150 nm), and surface area [126, 127, 128]. Since nanoscale particles are large enough to contain a variety of drug molecules, thus, they can allow the development of new treatment strategies [129, 130]. Among the known gold nanoparticles, gold nanostars (AuNSs) (multi-pod, branched morphologies) are a relatively new class of nanoparticles, having anisotropic morphology and a wide range of potential applications due to their optical properties, including localized surface plasmon resonance through the Near-IR spectral region, particularly for biomedical applications [131, 132, 133].

Gold Nanostars (AuNSs) can be covalently or electrostatically conjugated to a variety of biomolecules, including nucleic acids, proteins, peptides, antibodies, etc., conferring gold nanostars for targeting capabilities. However, the application of AuNSs for targeting requires their stability in solutions with high protein and salt concentration. AuNSs can target exosomes via functionalization with exosome-binding lectins and antibodies, aiming at exosome capture. To increase the half-life circulation AuNSs, generally they are functionalized with polymers such as polyethylene glycol (PEG), which enhance their hydrophilic character and, consequently, their colloidal stability, biocompatibility, and bio-distribution. The PEG layer on the surface of AuNSs generates an inert hydrophilic surface, which increases the stability of AuNSs at high concentrations of salt and biological media, preventing their morphological change. Similarly, functionalization with PEG prevents non-specific electrostatic adsorption of biomolecules, including proteins such as opsonin, which circulate in plasma proteins that mark antigens for phagocytosis. Therefore, AuNSs's coverage with PEG prevents recognition by the mononuclear phagocytic system (MPS) (which would prevent AuNSs interaction with exosomes), then increases its half-life in the blood and bio-distribution. Biotin-PEG-Vn96 has a polyethylene glycol linker between the peptide and biotin moieties, providing flexibility to Vn96 to access and capture exosomes. PEGs can be bonded with other functional groups at the AuNSs surface or can serve as linkers for the functionalization with other functional biomolecules as shown in Fig 6.2a [134 – 139].

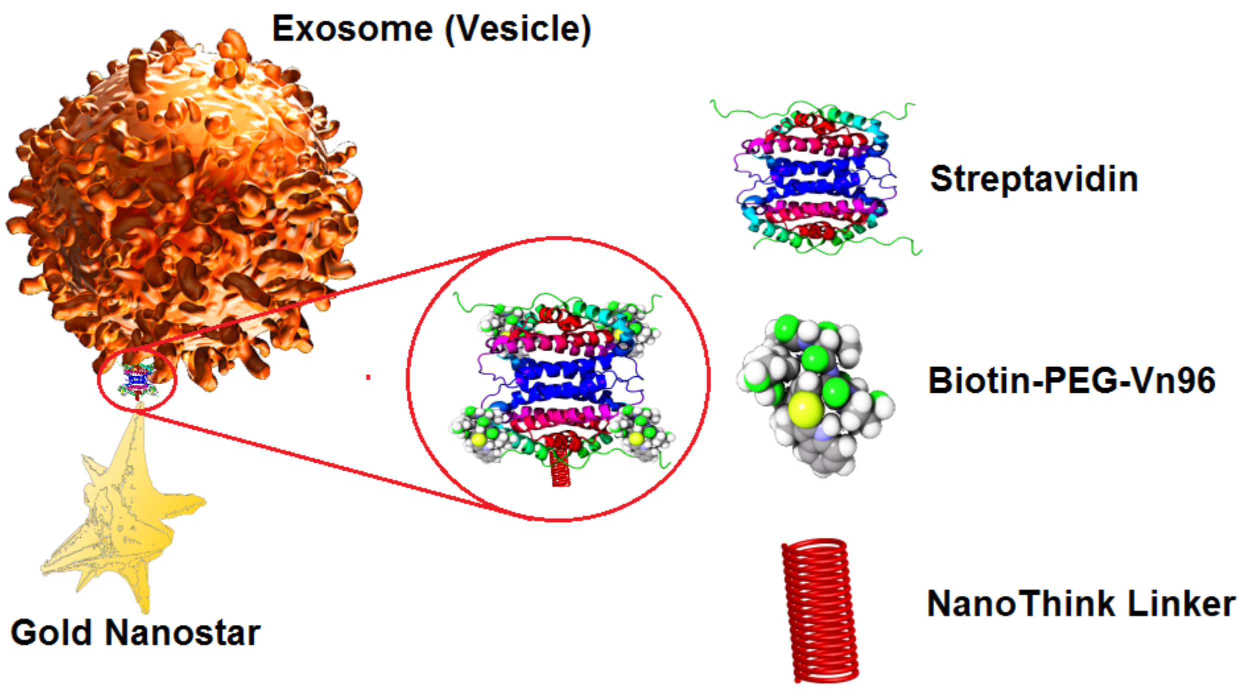
Surface plasmon resonance (SPR) is the collective oscillation of valence electrons in a metal, excited by the incident light. The LSPR condition depends on the size, shape of the metal nanoparticles and the refractive index of the environmental media surrounding the metal nanoparticles.

Recently, a localized surface plasmon sensor has been proposed with the use of colloidal gold monolayers deposited on a glass substrate. [140, 141, 142]. In this paper, we employed polydimethylsiloxane (PDMS) polymer as a substrate for AuNSs. Because of strong hydrophobicity of PDMS, a more appropriate PDMS surface modifications treatment was developed using PVA to form a composite polymer substrate for immobilization of gold nanostars.

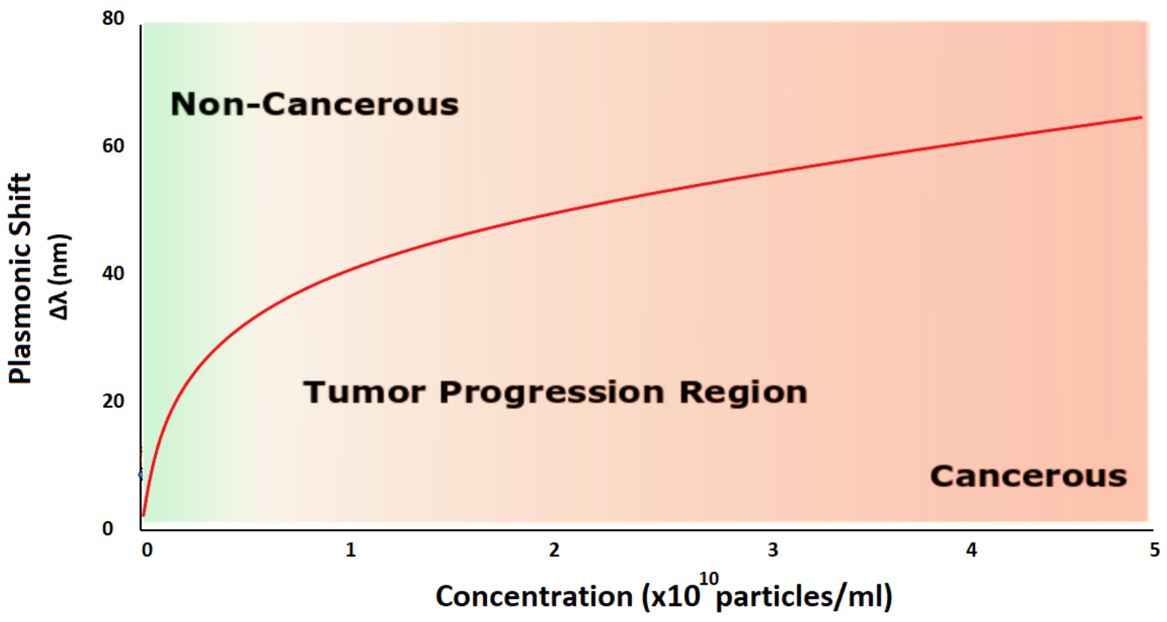
Surface plasmon resonance (SPR) based methods have been developed for the detection of Extra-Vesicular (EV) RNAs. In order to capture exosomes, the surface of gold nanostars has to be functionalized with one or more different types of antibodies against membrane proteins. Commercial SPR instruments were able to screen the membrane proteins and determine the concentration of exosomes in various cultured cell lines. However, due to large surface/volume ratio of gold nanostars, the affinity of target interaction is high and for this reason, the Localized Surface Plasmon Resonance (LSPR), based on noble metal nanostars proved to be a more straightforward way to detect and isolate exosomes [143, 144].

The concentration of exosomes present in the body fluids increases in a cancer patient compared to a healthy patient. For example, in the case of a glioblastoma (GBM) cancer cell study [145], the concentration of exosomes was found to be approximately 50 times higher than that of a healthy patient. Therefore, the concentration of exosomes, detected by measuring the LSPR plasmonic shift, will increase during the progression of cancer as shown in Fig 6.2b. This figure shows, in a general way, that the plasmonic shift for the exosomes released by cancerous cells increases when compared to healthy cells.

Here we address novel strategies that can be used to target MCF-7 exosomes derived from cancer cells, using gold nanostars, immobilized in PDMS/PVA substrate as a platform.



(a)



(b)

Fig 6.2: (a) Gold Nanostar (AuNS) functionalization for targeting Exosomes (not to scale). **(b)** Schematic of the relationship between plasmonic shift and concentration of exosomes during cancer progression.

6.2 Materials and Methods

6.2.1 Materials

The hydrochloric acid solution ($HAuCl_4$), silver nitrate ($AgNO_3$), sodium citrate ($Na_3C_6H_5O_7$), ascorbic acid ($C_6H_8O_6$) and PVA (C_2H_4O) (MW 89,000-98,000g/mol) were purchased from Sigma-Aldrich. Gold chloride (hydrogen tetrachloroaurate) ($HAuCl_4 \cdot 4H_2O$) was acquired from Alfa Aesar Chemical. De-ionized (DI) water with a resistivity of 18MΩ, used in all the experiments was obtained from the NANOpure ultrapure water system (Barnstead). The 11-mercaptoundecanoic acid in ethanol (Nano Thinks Acid 11), phosphate buffered saline (PBS) were obtained from Sigma-Aldrich, Canada. PBS tablets were dissolved in DI water at 0.1M concentration with a pH of 7.2. Streptavidin (SA) was purchased from IBA GmBH, Vn96-linker-biotin and MCF7 exosomes were supplied by the Atlantic Cancer Research Institute (ACRI), Moncton, N.B. Canada.

6.2.2 Fabrication of PVA/PDMS Composite

2 grams, 4 grams and 6 grams of NaOH and PVA in 1:1 ratio was added to 100 mL of DI water and stirred at room temperature for 10 min forming there different sample set named as Solution 1, Solution 2 and Solution 3. The temperature was then gradually increased to 100 °C, and the solution was stirred for another 40 min. Finally, the temperature was reduced to 80 °C to avoid lumping. DI water was added to compensate for any losses due to water evaporation.

The PDMS prepolymer and curing agent were thoroughly mixed in a 10:1 ratio. The mixture was poured in a petri dish, thoroughly degassed in vacuum and cured overnight in the oven at 65 °C.

To perform the surface modification treatment of PDMS was accomplished in the following steps:

Step 1: The PDMS chips (50×20×5 mm) were thoroughly degreased in IPA, blown dry and further dried in an oven at 115 °C for at least 40 min.

Step 2: The PDMS chips were immersed in Solution 1, Solution 2 and Solution 3 for 15 min at 80 °C,

Step 3: The chips were removed from the solution and blown dried and heated on a hotplate at 110 °C for 15 min.

We found that this process significantly increased the adsorption of NaOH-PVA onto the PDMS surfaces. A single layer of NaOH-PVA was sufficient to produce stable droplets. The process was repeated three times to get the desired hydrophilicity.

6.2.3 Contact angle measurement

The evaluation of surface hydrophilicity of the PVA-treated PDMS was performed via contact angle measurements from the contact area of liquid droplet spread on a solid substrate. Ten different samples were used from each sample set of PVA-treated PDMS. Deionized water (100 μL) was always dropped in the center of the PVA/PDMS sample, and the affinity for the surface was measured using a circular adjustment method. The average contact angle was extrapolated from three different samples. Measurements were made for each sample before treatment, immediately after treatment and also 30 days after treatment. All samples were stored in room temperature around 25 °C and humidity of 30-35%, continously monitored using Arduino weather station.

All the experiments were imaged using a python scripted frame grabber written explicitly for this experiment. The droplet size was measured using a Sony alpha 200 DSLR camera with 200mm macro lens with a shutter speed of 1/160 sec @ f/8 focus, ISO 100 as shown in Fig 6.3. The size was calculated, using a software, based on edge detection algorithm using Python programming language (Python Software Foundation, Wilmington, DE, USA), developed explicitly for this experiment. The monodispersity of the droplets was extrapolated from the Gaussian fitting of the histogram as the standard deviation of the average value (coefficient of variation).

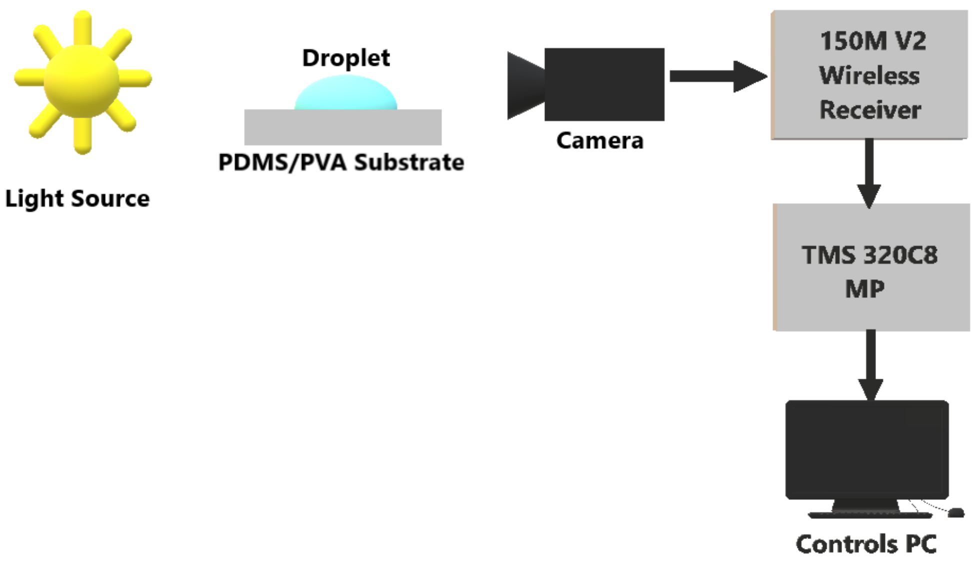


Fig 6.3: Experimental setup to measure the contact angle

6.2.4 Fabrication of Gold nanostars (AuNSs) embedded PVA/PDMS

The AuNSs embedded PVA-PDMS polymer composite was fabricated from gold nanostars solution deposited on a PVA/PDMS composite by the thermal convection method. Gold nanostars (AuNSs) were synthesized by a seed-mediated method. [146]

The PVA/PDMS composite substrates were cleaned with DI water and then rinsed with acetone, dried with 2-propanol (IPA) and air. Then, the substrate was heated in an oven at 100 °C for 1 hour to remove any moisture and contaminants, before the deposition process.

The PVA/PDMS composite substrates were immersed at an angle of approximately 30° in a beaker containing the gold nanostar solution and kept at room temperature around 25 °C until the solution is evaporated, depositing the multilayers of gold nanostars on the PVA/PDMS substrate as shown in Fig 6.4.

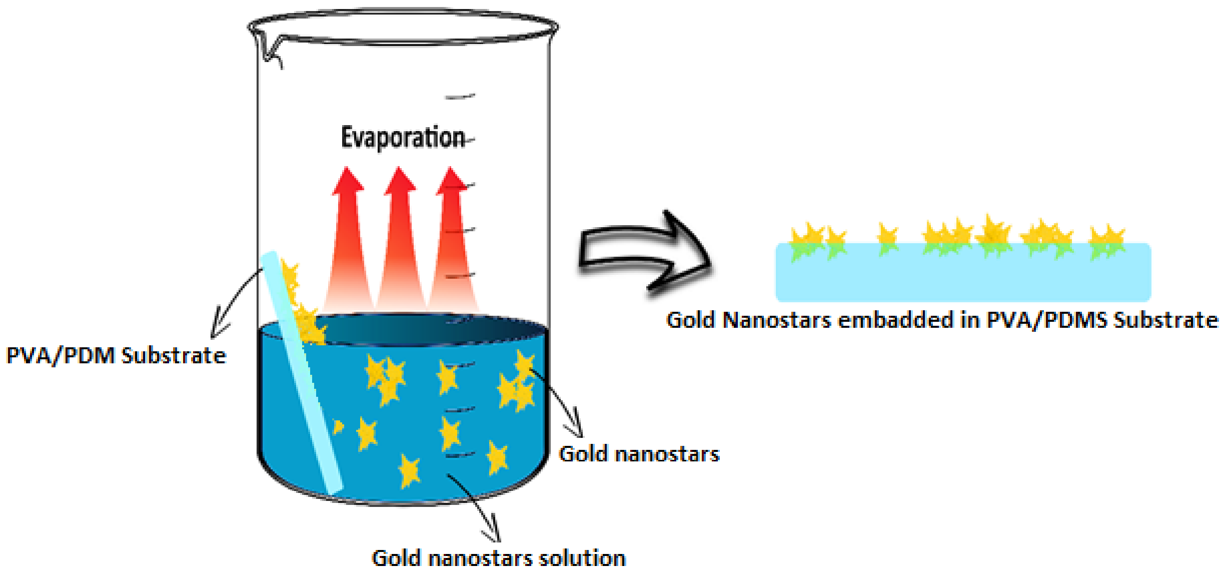


Fig 6.4: Schematic of the AuNSs deposition on PVA/PDMS substrate

6.2.5 Refractive index sensitivity measurements

The AuNS-PVA/PDMS composite or polymer composite samples are introduced into different dielectric media of varying refractive indices as listed in Table 1. The sensitivity (S) of the composite to the surrounding dielectric medium is calculated as the ratio of the Au-LSPR band shift ($\Delta\lambda$) between the solvent and de-ionized water to the change in the refractive index of the solvent to de-ionized water (Δn) [146]. Sensitivity (S) = $\frac{\Delta\lambda}{\Delta n}$ where $\Delta\lambda = \lambda_S - \lambda_W$ and $\Delta n = n_S - n_W$.

Where $\Delta\lambda$ is the change in the wavelength of the absorbance peak, λ_S is the wavelength of the Au LSPR band corresponding to the sample immersed in a solvent with a known refractive index and λ_W is the position of the band of the sample in de-ionized water, Δn is the difference between the refractive indices of the solvent and that of DI water, n_s is the refractive index of the solvent, and n_w is the refractive index of water.

Table 6.1. Refractive indices of dielectric media used for sensitivity tests

Media	**Refractive Index (n)**
Deionized Water	1.330
Ethanol	1.361
Acetone	1.360
N,N-Dimethylformamide	1.430
Dimethylacetamide	1.437
Chloroform	1.440

A Perkin Elmer Lambda 650 UV-Vis spectrometer was used for all the spectral measurements.

6.2.6 Functionalization of the Gold nanostars (AuNSs) embedded PVA/PDMS

The adsorption of linkers on nanoparticles has been widely investigated over the last decade [147]. The functionalization of AuNSs was accomplished using a three-component assembly as shown in Fig 6.5, which consists of the following:

Step 1: The absorption spectrum of AuNSs embedded in the PVA/PDMS was measured, and then 100 μL of the NanoThink 11 solution was deposited on the substrate and incubated for 3 hours and, then, allowed to dry. When dried, the absorption spectrum was measured again, using the Perkin-Elmer spectrophotometer (Lambda 650).

Step 2: In the next stage, 100 μl of different concentrations of Streptavidin (SA) solution were deposited on the various sample substrates and incubated for overnight to dry, followed by the spectral measurement.

Step 3: The last step of functionalization, involved depositing of 100μl of the different concentrations of the Biotin-PEG-Vn96 solution on the various substrates and incubating them overnight to dry, followed by spectral measurement.

Exosomes targeting: Tunable Resistive Pulse Sensing (TRPS) technology was used to measure the particle/ml of culture media. The concentration of exosomes in the culture media was found to be 1.33x10^{10} particles/ml. Varying concentrations of MCF 7 exosomes were deposited over different substrates and incubated overnight. Once the samples were dry, they were washed three to four times using 0.5% Blotting-Grade Blocker Non-Fat Dry Milk, to block the unreacted exosomes. The minimum concentration of exosomes was less than 0.066x10^{10} particles/ml, and maximum concentration was 4.65x10^{10} particles/ml.

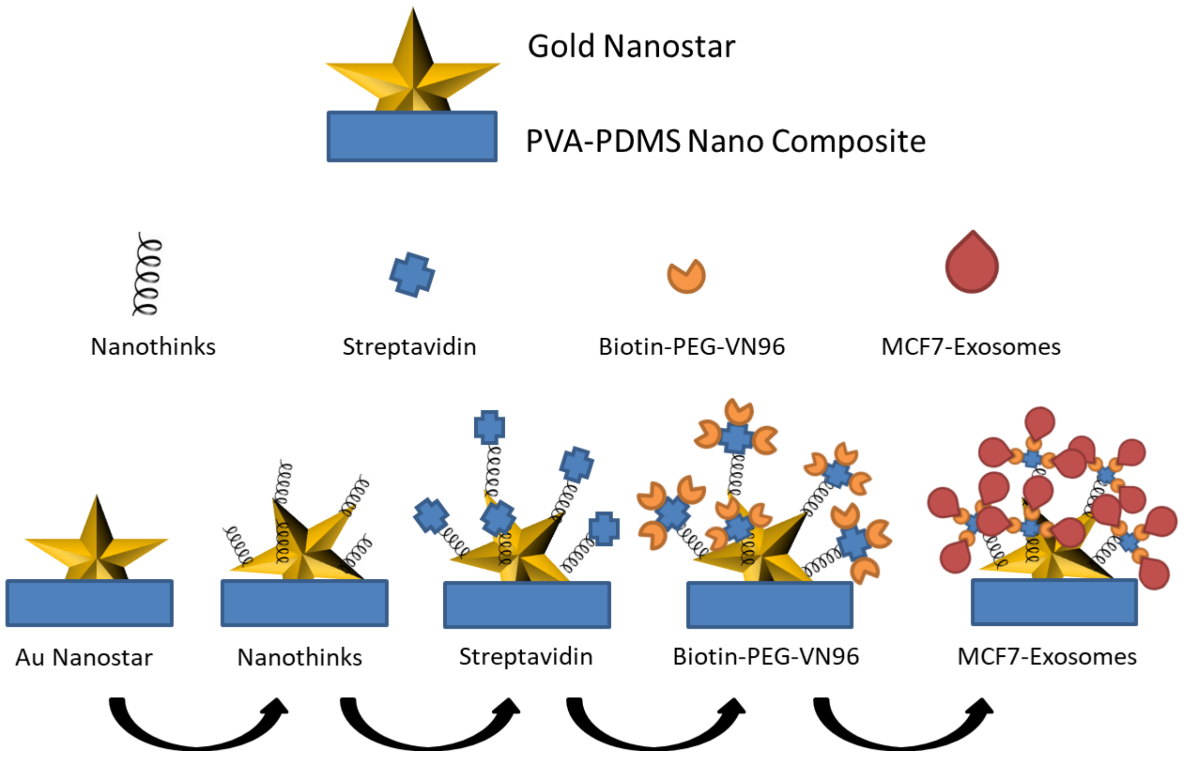


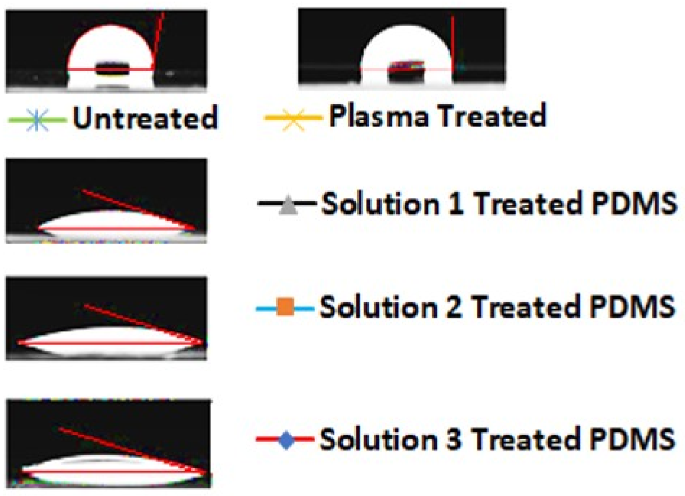
Fig 6.5: Biosensing protocol and their corresponding absorbance bands

6.3 Results and Discussion

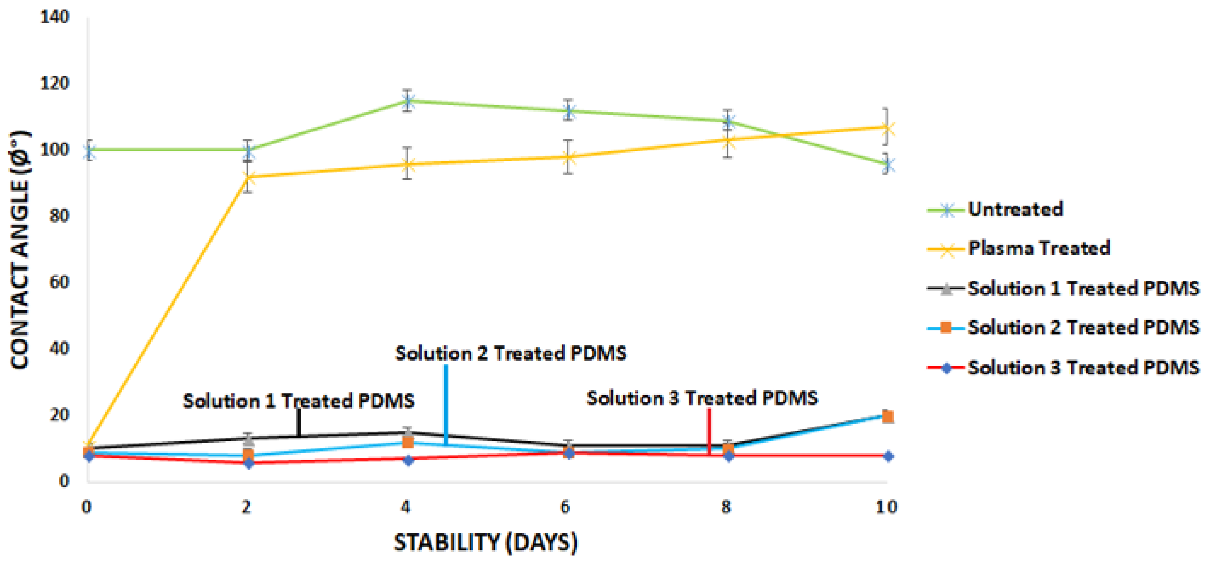
6.3.1. Surface wettability study of PVA/PDMS

The first step of the treatment was the generation of radical species of hydroxyls groups on the surface of PDMS, and this species will then form a thermal covalent bond with the PVA molecule. To do so, NaOH was used with PVA to form the radical species of C-OH [148]. A series of contact angle measurements were conducted to study the effect of NaOH-PVA deposition on the PDMS surface as shown in Fig 6.6a. The results showed that significantly lower water-air contact angles of PDMS treated with NaOH-PVA were obtained compared to native PDMS as shown in Figure 6.6b.

A long-term stable and sustained hydrophilicity of the PDMS surface was attained by combining the NaOH and PVA treatment with the plasma oxidized PDMS. When only the plasma activation was used, the sample tended to regain its surface hydrophobicity. The average contact angle noted was 8.5 ± 0.5° immediately after plasma oxidation and increased to 92.6 ± 0.3° 2 days later. The level of induced hydrophilicity was evaluated for PDMS treated with Solution 1, Solution 2 and Solution 3 and concluded that the average contact angles were 11.0 ± 4° for PDMS treated with Solution 2 and 8 ± 5° for PDMS treated with Solution 3 and this difference was retained for 9 days. PDMS treated with solution 3 was used for all subsequent experiments.



(a)



(b)

Fig 6.6: Contact angle measurements of PDMS surfaces as a function of time. **(a)** Affinity of a 10 µL water-in-air droplet for a PDMS surface under various surface treatment conditions **(b)** Untreated, Plasma treated, PVA-treated, NaOH-PVA treated PDMS. (Sample set, n = 10 samples. Standard Deviation = 4)

6.3.2 Refractive index sensing of surrounding environment

The LSPR is extremely sensitive to the environment surrounding the AuNPs. Therefore, the LSPR extinction curve is monitored as a function of changes in the local dielectric environment which is related to the refractive index as shown in Figure 6.7. This method can also be applied to sensing of biological molecules such as proteins and antibodies. The refractive-index sensitivity of the AuNSs LSPR was demonstrated by exposing the AuNSs-PVA/PDMS to different solvents having refractive indices listed in Table 1. The change in LSPR band (Δλ) versus refractive index of the local environment (Δn), for a solvent as shown in Figure 6.5, demonstrating a linear relationship between (Δλ) and (Δn).

Overall, the refractive-index sensitivity for AuNSs-PVA/PDMS was found to be 503.99 (nm/RIU) which is a much higher-refractive-index sensitivity compared to other platforms.

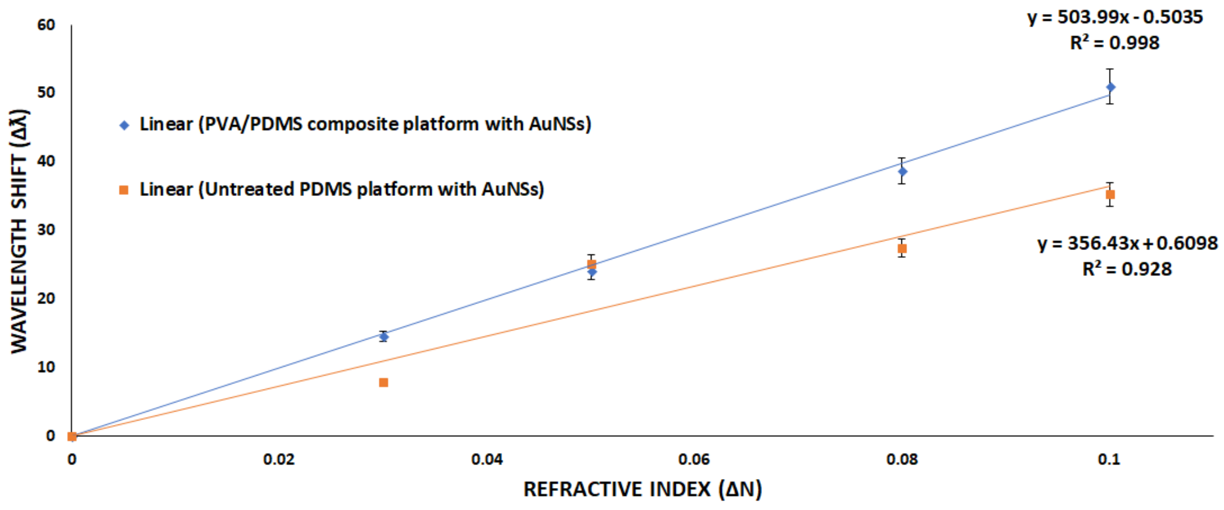


Fig 6.7: Sensitivity of the AuNS on the different composite platforms (Sample set, n = 10 samples. Standard Deviation = 5)

6.3.3 Biosensing Protocol

As shown in the experimental section, after each step, the AuNSs LSPR band was measured. Fig 6.8 shows the absorbance spectra of AuNSs, before and after each step of the protocol. AuNSs exhibit a localized SPR peak at 620 nm that shifted to 701.70 and 770.02 nm upon adsorption of Biotin-PEG-Vn96 onto the nanoparticle surface and subsequent conjugation with exosomes solution (0.066×10^{10} to 4.65×10^{10} particles/ml), respectively. In this case $\Delta\lambda^*$, plasmonic differnece is the relative shiht for the each step.

Table 6.2: Concentrations and volume of the chemical compounds used in the biosensing application with their corresponding LSPR shift Δλ*

Chemical compound	**Concentration Used**	**Avg. Δλ* (nm)**
NanoThink 11	5mM	25
Streptavidin	49 µg/mL	34
Biotin-PEG-Vn96	15µg/mL	29
Exosomes (MCF 7)	(0.003, 0.066, 0.133, 0.66, 1.33, 1.995, 2.66, 3.99, 4.65) x10^{10} particles/ml	68

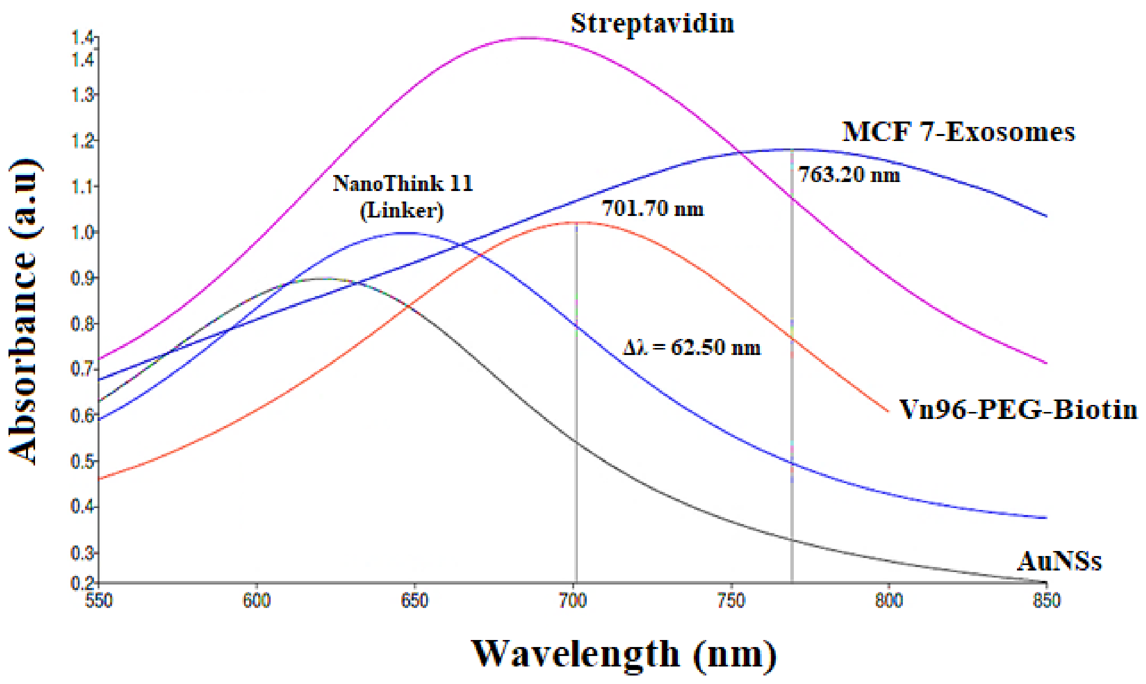


Fig 6.8: Absorption spectra of functionalized gold nanostars (AuNSs) on a PVA-treated PDMS sample platform after each step. (Sample set, n = 10 samples. Standard Deviation = 3)

6.3.4 Interaction of streptavidin with AuNSs

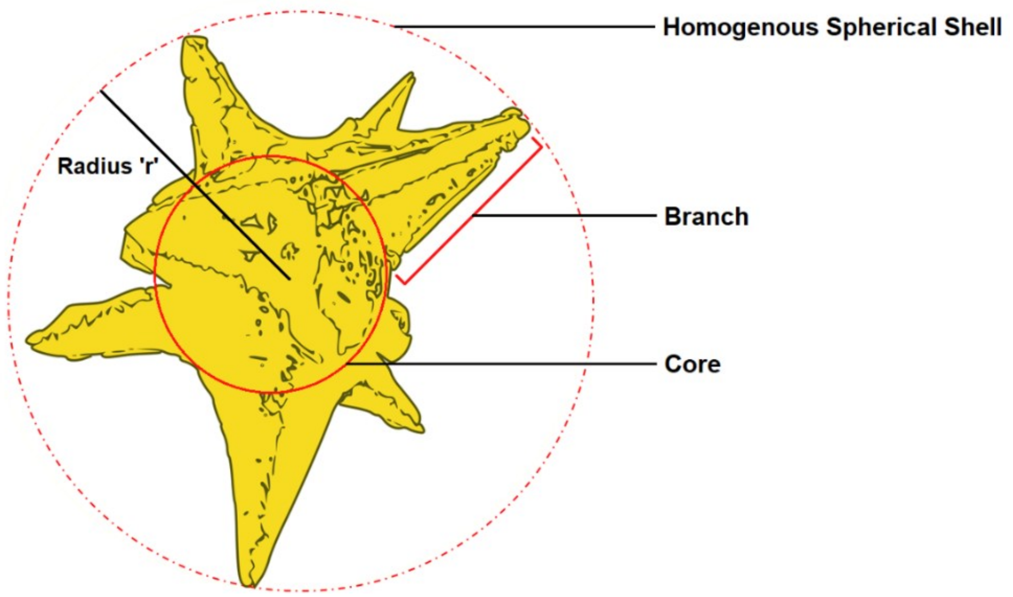


Fig 6.9: AuNSs depicted as a sphere with homogenous spherical shell

Monodispersed AuNSs is depicted as a sphere with homogenous spherical shell with an average spherical diameter of 40 ± 0.4 nm and branch length of 25 ± 10 nm will have a surface area availability per nanoparticle of about 13096.73 nm^2 as shown in Fig 6.9. Considering that streptavidin has a hydrodynamic diameter of 5 nm and an estimated surface area of 25 nm^2, a maximum of about 524 streptavidin molecules per nanoparticle can contribute to the formation of an adsorbed monolayer [149].

The 40 nm shift of the AuNS LSPR band, after the deposition of streptavidin can be used to predict the number of streptavidin molecules adsorbed per unit area (assuming again that AuNSs - Streptavidin can be depicted as a sphere with a homogenous spherical shell). On this basis, the spectral shift ($\Delta\lambda$) is given by [149]:

$$\boldsymbol{\Delta\lambda} = \frac{\lambda_p^2\,(\epsilon_S - \epsilon_W)}{\lambda_{max,AuNSs}(1+2\propto_S(1-g))}$$ **[Eq 6.1]**

Where, λ_p is bulk metal plasmon wavelength (131 nm for gold [43], λ_{max}, AuNSs is the wavelength of maximum absorption for AuNSs (625 nm, as shown in Fig 6.8), g is the fraction of nanoparticle that can be considered a shell, and $\propto_S = (\epsilon_S - \epsilon_W)/(\varepsilon s + 2\epsilon_W)$. g can be obtained from Equation 1 with $\Delta\lambda = 40$ nm by assuming a refractive index (n) of 1.334 for the medium (water) [44] and a refractive index of 1.47 for Streptavidin [150] ($\epsilon_W = n_{water}^2$ and $\epsilon_S = n_{SA}^2$).

The coating thickness (S) can be calculated from the shell fraction (g) and core diameter (d) [149]:

$$\boldsymbol{S} = \frac{d}{2}\left((1-g)^{-\frac{1}{3}} - 1\right)$$ **[Eq 6.2]**

The mass of strepavidin absorbed per unit area (coverage, ξ) is of interest as well as the thickness, which is used to calculated using the equation [149]:

$$\boldsymbol{\xi} = s\,\frac{n_{SA} - n_{water}}{d_n/d_c}$$ **[Eq 6.3]**

By using the value 0.212 cm^3 g^{-1} for the refractive-index increment (d_n/d_c) of Streptavidin [150] a coverage of 19 mg m^{-2} of streptavidin, corresponding to about 414 streptavidin molecules per nanostar, is obtained.

6.3.5. Interactions of Biotin-PEG-Vn96 with streptavidin bounded AuNSs

The streptavidin–biotin interaction has a very high affinity ($K_a \sim 10^{15}$ M^{-1}) under various conditions [151] and will serve as a very good model for the LSPR nanosensor. Streptavidin is a tetrameric protein with four subunits arrayed symmetrically, and each unit has a biotin-binding affinity. Therefore it can be expected that four of the biotin binding pockets are accessible by Biotin-PEG-Vn96 per surface-adsorbed streptavidin molecule. On this basis, it can be estimated to be around 828 Biotin-PEG-Vn96 molecules were immobilized per AuNS.

The experiment involved treating the streptavidin functionalized AuNSs with 3.0, 7.5, 15, 20 (µg/mL) concentration of biotin-PEG-Vn96 solution. After the incubation, the glass slide was washed with DI water for several times to remove unbound biotin-PEG-Vn96.

A red shift in the LSPR band was noticed in Fig 6.8 after depositing the streptavidin solution.

To understand the importance of the functionalization of AuNSs, it is necessary to understand the kinetics of the association between streptavidin and biotin. These two biomolecules have exceptionally high affinity with an binding constant $K_A \sim 10^{15}$ $Lmol^{-1}$ [152].

If the concentration of the streptavidin is [S] and the Biotin-PEG-Vn96 concentration is [B], therefore concentration of the formed complex is [SB]. The interactions between streptavidin and Biotin-PEG-Vn96 can be described as [151]:

$$[\boldsymbol{S}] + [\boldsymbol{B}] \Leftrightarrow [\boldsymbol{SB}] \qquad \textbf{[Eq 6.4]}$$

Where the association and dissociation happen at the same time but at the different rates.

The rate of association is given as k_a[S][B], where k_a is the association rate constant with the unit $M^{-1}s^{-1}$ while the rate of dissociation is given as k_d[SB], where k_d is the dissociation rate constant with the unit s^{-1}. Therefore, the rate of formation of complexes at a given time t is given by [153]:

$$\frac{d[SB]}{dt} = k_a[S][B] - k_d[SB] \qquad \textbf{[Eq 6.5]}$$

The concentrations of the already formed complexes and remaining streptavidin are related as [S] = [S_0] - [SB], where [S_0] is the concentration of streptavidin before introducing the biotin. Therefore, the above equation can be written as [153]:

$$\frac{d[SB]}{dt} = k_a[B]([S_0] - [SB]) - k_d[SB] \qquad \textbf{[Eq 6.6]}$$

This implies that the concentration [S] is a constant value. There let [S] = C_S, therefore [154]

$$\lambda_C = m_c[SB] \qquad \textbf{[Eq 6.7]}$$

Where λ_C is the shift in the wavelength and m_c is the sensitivity coefficient. The maximum response occurs when all the streptavidin is bonded to biotin-PEG-Vn96 in the association phase. Therefore, the relation between the kinetic and equilibrium constants is as described in equation, which corresponds the maximum wavelength shift λ_{max} [154]:

$$\lambda_C = \lambda_{max}\left(1 + \left(\frac{k_d}{C_s k_a}\right)\right) \qquad \textbf{[Eq 6.8]}$$

Knowing the λ_C, λ_{max} values from Fig 6.6 and $k_d = 2.4\text{x}10^{-6}$ S^{-1} [46] it is found that the association rate constant is $4.5\text{x}10^{7}$ M^{-1} s^{-1}, which shows that streptavidin and biotin bond can be considered as strong binding.

And at steady state [153]:

$$\boldsymbol{k_a[S][B] = k_d[SB] \Rightarrow \frac{[SB]}{[S][B]} = \frac{k_a}{K_d}} \textbf{ and } \boldsymbol{[SB] = [S_0] = \frac{\lambda_{max}}{m_c}} \qquad \textbf{[Eq 6.9]}$$

Therefore,

$$\boldsymbol{\frac{\frac{\lambda_{max}}{m_c}}{[S][B]} = \frac{k_a}{k_d} \Rightarrow [B] = \frac{k_d\left(\frac{\lambda_{max}}{m_c}\right)}{[S]k_a} = 653\ M} \qquad \textbf{[Eq 6.10]}$$

Where [S] is the molecule of streptavidin calculated in section 3.4, [S] = 414 *M*.

Therefore, the number of Biotin attached per streptavidin is given by

$$\boldsymbol{\frac{[B]}{[S]} = 2.4} \qquad \textbf{[Eq 6.11]}$$

Meaning that two or three Biotin-PEG-Vn96 is bonded to one streptavidin molecule. Therefore, maximum of about 1572 Biotin-PEG-Vn96 molecules per nanoparticle can contribute to the formation of an adsorbed monolayer.

6.3.6 Interaction of MCF-7 Exosomes with functionalized AuNSs

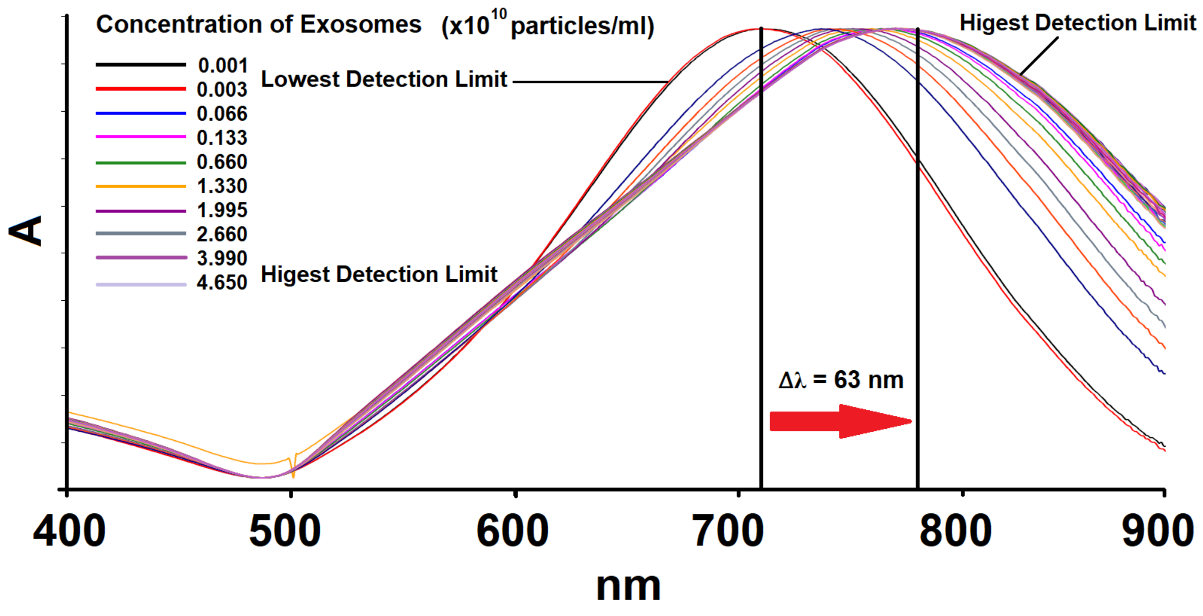
The functionalization is experimentally studied by depositing different concentration of Exosomes solution on the functionalized AuNSs-PVA/PDMS substrate. The measured reflection spectra are plotted in Fig 6.10a. The LSPR wavelength shift as a function of concentration (particles/mL) along with the corresponding sensitivity. A red shift from 703 nm to 766 nm was noted in resonance wavelength when a solution of 3.99×10^{10} particles/ml of exosomes was deposited. The concentration is a key biosensing parameter that determines the sensitivity with which very small wavelength changes can be measured, by considering the sharpness of the resonance.

In order to demonstrate the limitation of the biosensing for targeting the exosomes, the corresponding values for the LSPR shifts were noted for the lowest concentration as 710 nm for 0.001×10^{10} and 0.003×10^{10} particles/ml and the highest concentration as 766 nm for 3.99×10^{10} and 4.65×10^{10} particles/ml.

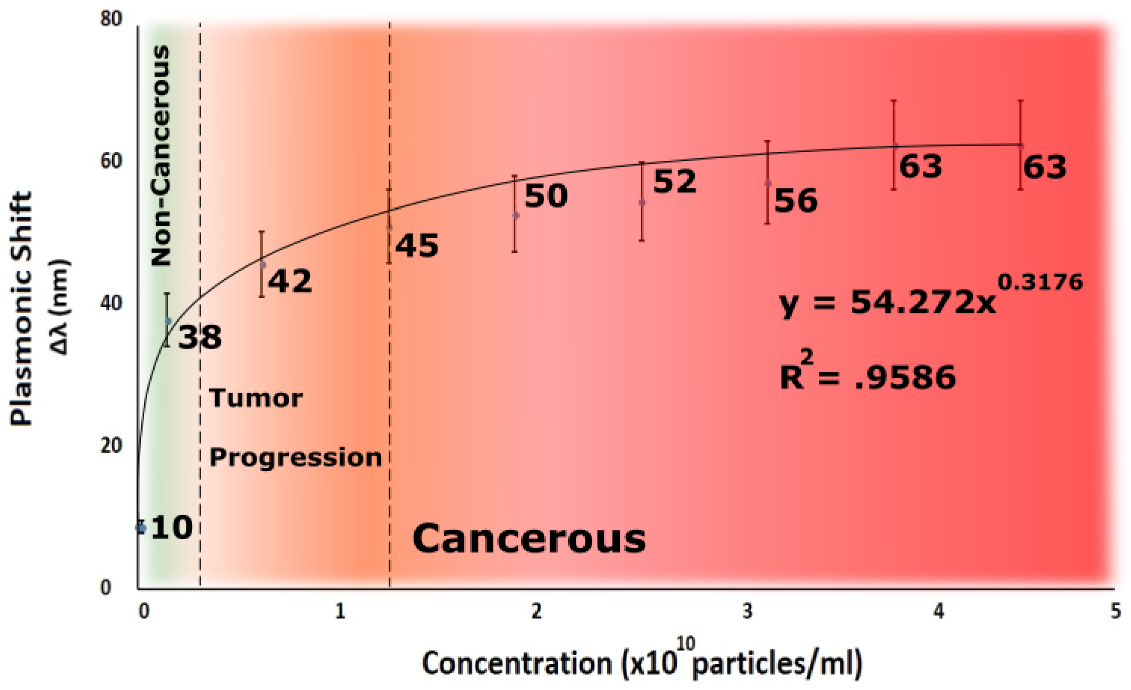
When exosomes are captured by biotin-PEG-Vn96, the interactions could result in deformability due to their elastic nature. Therefore, the protein – protein binding of exosomes to the biotin-PEG-Vn96 molecules will result in a non-linear behavior. The dependency curve of the system is defined as $S = \Delta\lambda/(\text{particles/mL})$, where $\Delta\lambda$ is the LSPR spectral shift and (particles/mL) is the concentration of exosomes deposited on the substrate as shown in Fig 6.10b. The dependency curve allows the estimation of concentration in terms of the number of exosomes in a sample. The equation of the curve reflects a non-linear trend in the form of Lx^M where x corresponds to the number of exosomes to biotin-PEG-Vn96 interactions. It can be observed from the experimental results that the LSPR shift of the exosomes has a non-liner dependency on binding to the biotin-PEG-Vn96 and the curve is significantly not yet saturated, which clearly explains that the capacity of nanostars platform to capture exosomes is higher than the standard gold nanoparticles based platform.

The culture media used in the experiment belongs to fully developed cancerous condition taken from patients. Therefore, 1.33×10^{10} particles/ml correspondes to the concentration of exosomes in a fully developed condition and 0.066×10^{10} particles/ml concentration of exosomes of this sample would corresponds to a non-cancerous situation. Hence, the tested range of concentrations of exosomes covers non-cancerous to finally cancerous condition.

Hence, the gold nanostars embedded PVA/PDMS polymer platform based on LSPR interactions with the optimized biosensing protocol proves to be the effective way to detect and capture high concentration of exosomes corresponding to cancerous conditions. Therefore, the developed nanostars based platform can effectively detect early stages of cancer as well as progress leading to very early diagnosis.



(a)



(b)

Figure 6.10: (a) Concentration variant adsorption spectrum of Exosomes deposited on Biotin-PEG-Vn96 bonded AuNSs-PVA/PDMS substrate. **(b)** Calibration cure of average LSPR shift for various concentrations of Exosomes (Sample set, n = 10 samples. Standard Deviation = 4)

6.4 Conclusion

We demonstrated a simple, one-step surface modification method to create hydrophilic PDMS substrate, which surpasses several existing techniques. The proposed surface modification was achieved by exposing the PDMS surface to NaOH-PVA in DI water. Our experimental data show that the PVA-treated PDMS surfaces retain their hydrophilicity for more than 10 days highlighting the ability of the proposed method to deliver off-the-shelf products that can be maintained without compromising the performance of the microfluidic device. The AuNSs-PVA/PDMS substrate exhibit a strong LSPR at wavelengths around 625 nm. The efficiency of this platform was demonstrated with sensitivity factor estimated as 503.99 nm RIU^{-1}.

We believe that, the success of this LSPR nanosensor is directly related to the isolation of target biomolecules. The attachment of MCF 7 on the functionalized Gold nanostars leads to a detectable shift in localized surface plasmon resonance, which is used here to study the extent and the thereby the efficiency of the functionalized Gold nanostars in targeting MCF 7 exosomes. The biosensing platform can be employed for a wide range of exosomes concentrations: from highly diluted concentration of 0.066×10^{10} particles/ml with a LSPR at wavelengths around 704 nm to the more concentrated amount of 3.99×10^{10} particles/ml with strong LSPR at wavelengths around 763 nm.

In summary, this AuNSs-PVA/PDMS substrate has the ability to detect the extremely small refractive index changes, allowing the detection of lower and higher concentration of biomolecules in extremely diluted solutions.

Chapter 7

Contributions, Conclusions and Future Work

7.1 Contribution

As solutions to the difficulties encountered during my attempt to accomplish the objectives stated in this thesis, this work has:

1) Characterizes optically an AuNS by computational simulations of absorbance spectra. The numerical model was built up by decomposing an AuNS into parameterized structural elements to allow for the systematic study of its optical properties. As a result, this structure was shown to have highly tunable fundamental SPR, which can be achieved by adjusting the geometry, orientation and number of their branches. The most effective way of tuning the resonant excitation of an AuNS is determined.

2) The synthesis process of the gold nanostars holds the key to understand the morphology and shape tuning ability of gold nanostars.
   a) For the first time, detail studies where conducted to understand the effect of concentration of silver nitrate and temperature on the growth and aggregation kinetics of the gold nanostars.

3) The traditional methods use non-biocompatible chemical to stabilize the gold nanostars making the use of gold nanostars for biosensing impossible. As solutions to the difficulties encountered, novel methods were innovated to stabilize and increase the size of the gold nanostars for biosensing application.

a) Developed novel method to increase the stability of gold nanostars by modifying the pH of gold nanostars solution. The nanostars remained monodispersed in the growth solution for more than two weeks in room temperature.
b) Developed novel Sono-chemistry method to increase the shape and stability of the gold nanostars in the growth solution. Ultrasonication at 25 kHz was used to fuse nanostars together causing the increase in the particle size and increasing the aggression energy in the solution therefore increasing the aggression limit of the nanostars. The nanostars remained monodispersed in the growth solution for more than two weeks in room temperature.
c) Innovated a host-guest complex to increase the stability of the gold nanostars by creating PVA embedded gold nanostars composite. The gold nanostars retained they optical properties for more than 200 days.

4) Innovated a standard preparation for surface modification of PDMS using NaOH-PVA solution to increase the hydrophilicity of PDMS for biofunctionalization application. The results of this method prolong the hydrophilicity of the PDMS surface over 10 days when compare to standard plasma treatment, where the hydrophilicity of PDMS last only for few minutes.

5) Current scientific research focuses more on gold nanospheres as the primary nanoparticle for biosensing application. For the first time, a high sensitivity biosensing platform using gold nanostars as a primary nanoparticle to detect the progress of cancer using MCF7 exosomes as a bioindicator. The biosensing platform can be employed for a wide range of exosomes concentrations: from highly diluted concentration of 0.066x1010 particles/ml

with a LSPR at wavelengths around 704 nm to the more concentrated amount of 3.99x1010 particles/ml with strong LSPR at wavelengths around 763 nm.

6) A lot of low cost apparatus were invented during the research to accelerate the progress and quality of the research.
   a) Low cost high speed cavitation imaging system.
   b) Subpixel rendering methods to increase the quality of the SEM images by roster to vector conversation methods.
   c) Low cost water droplet imaging system.
   d) Low cost ultrasound generator system.

7.2 Conclusion

Chapter 2 characterizes optically an AuNS through FDTD-analyses. It was shown that the geometry of the branches is a key tuning factor of the absorbance peak of AuNSs. This can be shifted both in intensity and position, depending on the length and aperture of the branches. To adjust the overall optical response of the AuNSs for maximal sensitivity within a well-defined region of the optical spectrum, the most effective means of fine-tuning may be desired prepared the theoretical and simulation background necessary to conclude that the LSPR fine-tuning by adjusting the parameter of the branch is much more effective than adjusting the length. However, since the nucleation and aggregation process involving molecule formation and assembly cannot be controlled. Therefore the knowledge of the optical properties depending on the above geometrical parameters is of high importance in tailoring AuNSs of desired optical properties, such as maximal sensitivity in a convenient region of observation.

Chapter 3 presented the method used for achieving stability of gold nanostars increased with increase in the pH of nanostars solution to 11.0. The AuNS retained their original optical properties and shape even after two weeks at room temperature. The SEM image shows the cause of increase in the size of the nanostars to be that of increase in the branch length of gold nanostars. Upon further investigation of the shape-tuning effects of base buffer at pH 11, it is concluded that unlike the transformation of nanostars to nanospheres which has been reported in many literature, the nanostars transformed from nanospheres cannot be reversed to nanostars again.

Chapter 4 demonstrated for the first time that the sonication frequency has a significant effect on the size and morphology of gold nanostars. At the initial stage of the reaction, many the small gold particles with a diameter around 30 nm are produced by reduction of gold nanoparticles ions with a sodium citrate solution. The seed-mediated method, followed by the sonication treatment developed in this study, offers new opportunities to synthesize aqueous suspensions of monodispersed gold nanostars with prolonged stability. The result indicates that at a sonication frequency of 25 kHz, the maximum average particle size of 120 nm is attained, due to the large diameter of the cavitation bubble which, under collapse, generates extreme conditions of pressure and temperature. Therefore, the LSPR extinction spectra of the gold nanostars, centered within the range of 650 to 900 nm, is tunable via the change in the ultrasound frequency. The optical characteristics of gold nanostars solution after ultrasound treatment appeared to be substantially more stable than those of non-treated ones because of the slower aggregation of NSs. The use of sonication technique to stabilize gold nanostars, without any additional chemical stabilizer, is both technologically and scientifically important. It can, hence, be expected to be able to produce by this method, a large volume of consistent quality gold nanostars.

Chapter 5 presented a host-guest complex has been prepared, by embedding the stabilized Au nanostars into the porous network of the hydrated PVA. Due to the strong van der Waals interactions between individual stars, they form aggregates inside the pores. It is assumed that during the sensing process, the solvent enters the PVA network and contacts the long branches of the stars. In the case of sensing with Au NS – PVA composites, the sensing entities are not the individual stars but their aggregates

Chapter 6 proposed surface modification was achieved by exposing the PDMS surface to NaOH-PVA in DI water. The NaOH creates the hydroxyl group (C-OH) in the presence of PVA when heated, which allows the hydrogen bonding between the PVA molecule and the PDMS surface, leading to a permanently hydrophilized surface. Our experimental data show that the PVA-treated PDMS surfaces retain their hydrophilicity more than 10 days highlighting the ability of the proposed method to deliver off-the-shelf products that can be maintained without compromising the performance of the microfluidic device.

The Streptavidin coated AuNSs were modified with biotinylated PEG-Vn96 to develop an innovative targeting approach. We believe that, the success of this LSPR nanosensor is directly related to the isolation of target biomolecules. The attachment of MCF 7 on the functionalized Gold nanostars leads to a detectable shift in localized surface plasmon resonance, which was used here to study the extent and the thereby the efficiency of the functionalized Gold nanostars in targeting MCF 7 exosomes. The AuNSs-PVA/PDMS substrate exhibit a strong LSPR at wavelengths around 625 nm. The shift in LSPR wavelength upon deposition of MCF 7 exosomes showed a linear response with a correlation coefficient of R = 0.95. From the slope of the linear fit, the sensitivity factor was estimated as 512.37 nm RIU^{-1}. The binding events between the immobilized biotin and streptavidin molecules were investigated in a streptavidin concentration range of 2.0 - 20 (µg/mL).

Furthermore, the biosensing platform can be employed for a wide range of exosomes concentrations: from highly diluted concentration of 0.066×10^{10} particles/ml with a LSPR at wavelengths around 704 nm to the more concentrated amount of 3.99×10^{10} particles/ml with strong LSPR at wavelengths around 763 nm. In summary, this AuNSs-PVA/PDMS substrate has the ability to detect the extremely small refractive index changes, allowing the detection of lower and higher concentration of biomolecules in extremely diluted solutions.

7.3 Refereed Journal papers

1. **R. Kumar**, S. Badilescu and M. Packirisamy "Modeling the Effects of Morphology on Plasmonic Behaviour of Gold Nanostars" submitted to: Plasmonics, July, 2018, to be submitted.

2. **R. Kumar**, S. Badilescu and M. Packirisamy "Tuning of Morphology and Stability of Gold Nanostars through pH Adjustment" submitted to: Journal of Nanoscience and Nanotechnology, May, 2018, accepted and under publication process.

3. **R. Kumar**, S. Badilescu and M. Packirisamy "Sonochemical Stabilization of Gold Nanostars: Effect of Sonication Frequency on the Size, Shape and Stability of Gold Nanostars" AdvNanoBioM&D: 2017: 1(4):172-181.

4. **R. Kumar**, S. Badilescu and M. Packirisamy "Sensitivity and Stability enhancement of Poly (vinyl alcohol)-nanostars composites for Localized Surface Plasmon Resonance

sensing applications" submitted to: Photonics and Nanostructures - Fundametals and Applications, July, 2018, currently under review.

5. **R. Kumar**, S. Badilescu and M. Packirisamy "Gold Nanostars polymer composite platforms for Localized Surface Plasmon Resonance Based Early Cancer Diagnosis" Journal of Biophotonics, July, 2018, to be submitted.

7.4 Proceedings papers and Conference presentations

1. **R. Kumar**, S. Badilescu and M. Packirisamy "Ex-Situ Synthesized and Diffusion Driven Highly Anchored Gold Nano-Stars in Polymer Composites" Photonics North 2016 Abstract Reference Number: 255-SLiU-192.

2. **R. Kumar**, S. Badilescu and M. Packirisamy "Influence of the Potential of pH on the Morphology and Stability of the Gold Nanostars solution" International Conference of Theoretical and Applied Nanoscience and Nanotechnology (TANN'17), ID:124.

7.5 Suggestions for further research

1. The FDTD simulation were conducted only with sphere with two branch models due to system memory limitation. Simulation with more than two branch models can be simulated now using cluster server installed with RSoft software.

2. Understanding the effect of branch morphology is the key to understand the optical properties of gold nanostars. Using the RSoft software on cluster, multiple branches with different branch length needs to be studies to exactly corelate the outcome of the synthesized gold nanostars.

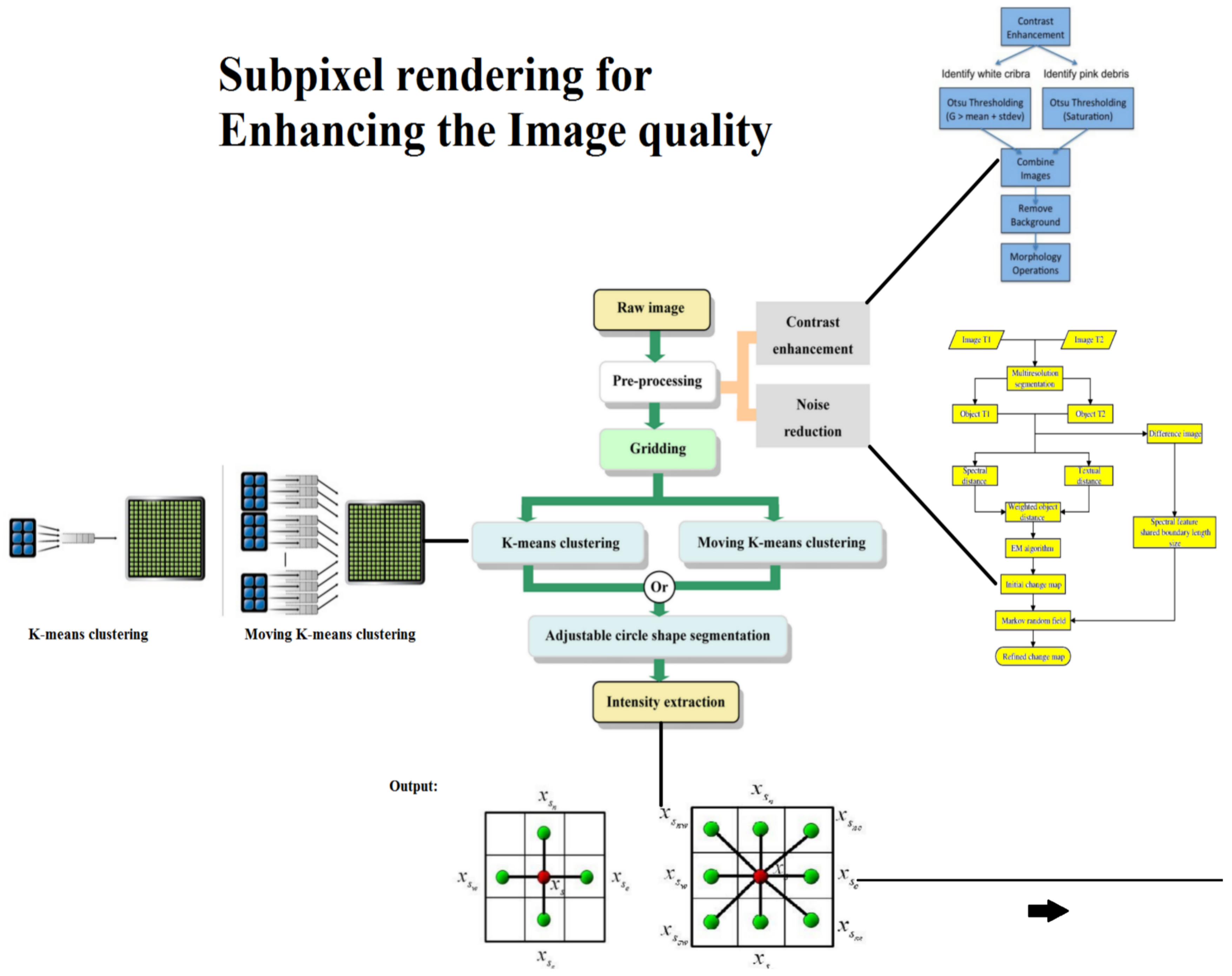
3. Gold nanostars are currently not stable when conducting the liquid biopsy. To increase the stability of the gold nanostars for the liquid biopsy application, research needs to be conducted on gold nanostars suspended in PB buffer or another biocompatible neutral buffer.

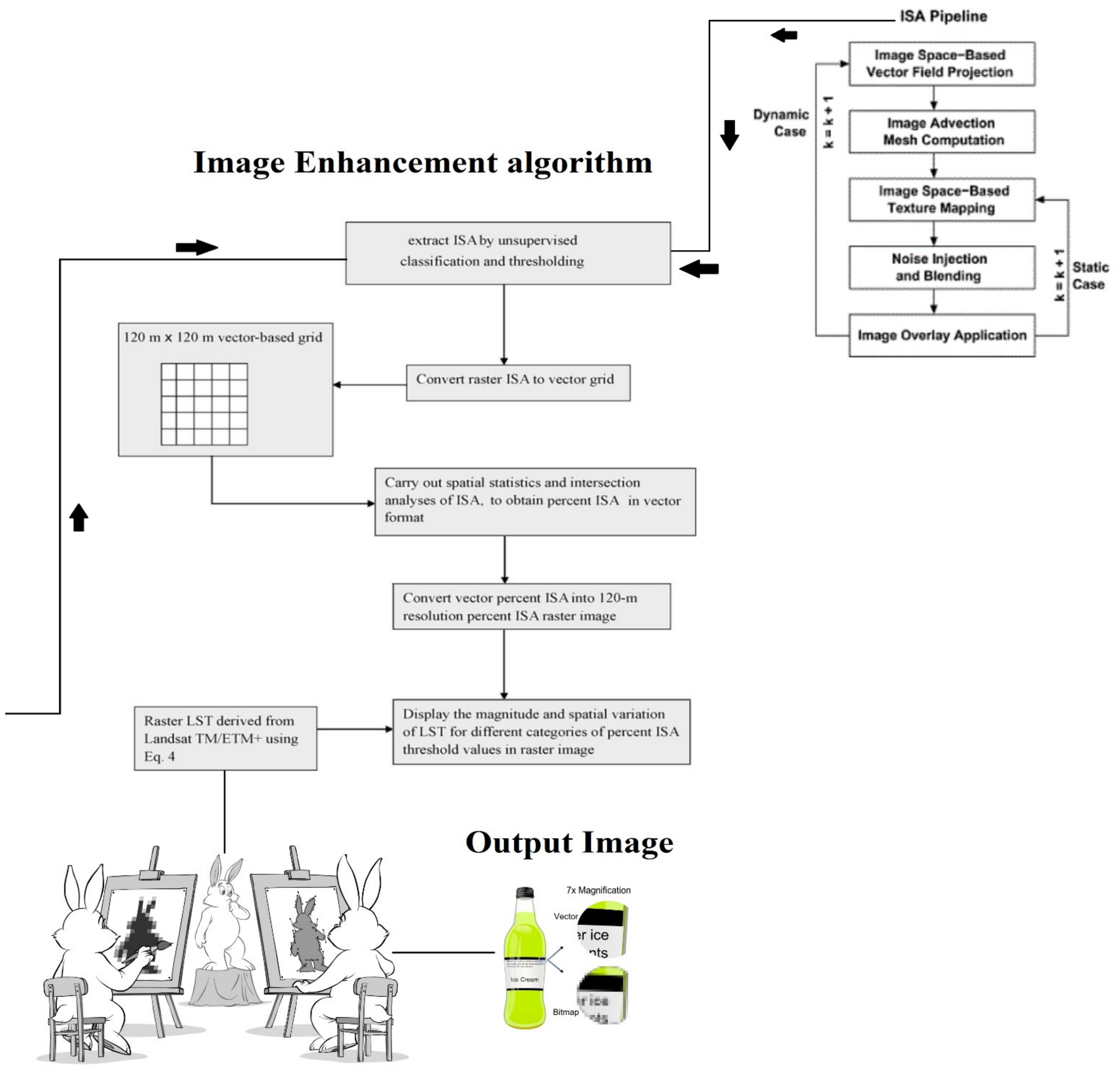
4. To increase the application area of the gold nanostars based biosensing platform, research should be conducted microfluidic application of the platform.

Appendix A

Subpixel Rendering for Image Enhancement

Subpixel rendering for Enhancing the Image quality




ISA Pipeline
Image Space–Based Vector Field Projection
Image Advection Mesh Computation
Image Space–Based Texture Mapping
Noise Injection and Blending
Image Overlay Application
Dynamic Case
k = k + 1
Static Case
k = k + 1
Image Enhancement algorithm
extract ISA by unsupervised classification and thresholding
Convert raster ISA to vector grid
120 m × 120 m vector-based grid
Carry out spatial statistics and intersection analyses of ISA, to obtain percent ISA in vector format
Convert vector percent ISA into 120-m resolution percent ISA raster image
Display the magnitude and spatial variation of LST for different categories of percent ISA threshold values in raster image
Raster LST derived from Landsat TM/ETM+ using Eq. 4
Output Image
7x Magnification
Vector
Bitmap
Ice Cream

The k-means problem is solved using Lloyd's algorithm. The average complexity is given by O (k n T), were n is the number of samples and T is the number of iteration. The worst-case complexity is given by O(n^(k+2/p)) with n = n_samples, p = n_features. (D. Arthur and S. Vassilvitskii, 'How slow is the k-means method?' SoCG2006) In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available), but it falls in local minima. That's why it can be useful to restart it several times.

Code:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import argparse
import utils
import cv2
from Tkinter import *
import os
import ctypes
from PIL import ImageTk
from PIL import ImageOps
from tkFileDialog import*
import tkMessageBox
import imghdr
from PIL import Image, ImageDraw
from collections import*
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn import datasets

np.random.seed(5)

iris = datasets.load_iris()
X = iris.data
y = iris.target

estimators = [('k_means_iris_8', KMeans(n_clusters=8)),
              ('k_means_iris_3', KMeans(n_clusters=3)),
              ('k_means_iris_bad_init', KMeans(n_clusters=3, n_init=1,
                                               init='random'))]

fignum = 1
titles = ['8 clusters', '3 clusters', '3 clusters, bad initialization']
for name, est in estimators:
    fig = plt.figure(fignum, figsize=(4, 3))
    ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
    est.fit(X)
    labels = est.labels_

    ax.scatter(X[:, 3], X[:, 0], X[:, 2],
               c=labels.astype(np.float), edgecolor='k')

    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
    ax.set_xlabel('Petal width')
    ax.set_ylabel('Sepal length')
    ax.set_zlabel('Petal length')
    ax.set_title(titles[fignum - 1])
    ax.dist = 12
    fignum = fignum + 1

# Plot the ground truth
fig = plt.figure(fignum, figsize=(4, 3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

for name, label in [('Setosa', 0),
                    ('Versicolour', 1),
                    ('Virginica', 2)]:
    ax.text3D(X[y == label, 3].mean(),
              X[y == label, 0].mean(),
              X[y == label, 2].mean() + 2, name,
              horizontalalignment='center',
              bbox=dict(alpha=.2, edgecolor='w', facecolor='w'))
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X[:, 3], X[:, 0], X[:, 2], c=y, edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel('Petal width')
ax.set_ylabel('Sepal length')
ax.set_zlabel('Petal length')
ax.set_title('Ground Truth')
ax.dist = 12

fig.show()

################ DRAW ################

def drawOnImage(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=True
    drawWindow=Toplevel(canvas.data.mainWindow)
    drawWindow.title="Draw"
    drawFrame=Frame(drawWindow)
    redButton=Button(drawFrame, bg="red", width=2, \
                     command=lambda: colourChosen(drawWindow,canvas,
"red"))
    redButton.grid(row=0,column=0)
    blueButton=Button(drawFrame, bg="blue", width=2,\
                      command=lambda: colourChosen(drawWindow,canvas,
"blue"))
    blueButton.grid(row=0,column=1)
    greenButton=Button(drawFrame, bg="green",width=2, \
                       command=lambda: colourChosen(drawWindow,canvas,
"green"))
    greenButton.grid(row=0,column=2)
    magentaButton=Button(drawFrame, bg="magenta", width=2,\
                          command=lambda: colourChosen(drawWindow,canvas,
"magenta"))
    magentaButton.grid(row=1,column=0)
    cyanButton=Button(drawFrame, bg="cyan", width=2,\
                      command=lambda: colourChosen(drawWindow,canvas,
"cyan"))
    cyanButton.grid(row=1,column=1)
    yellowButton=Button(drawFrame, bg="yellow",width=2,\
                        command=lambda: colourChosen(drawWindow,canvas,
"yellow"))
    yellowButton.grid(row=1,column=2)
    orangeButton=Button(drawFrame, bg="orange", width=2,\
                        command=lambda: colourChosen(drawWindow,canvas,
"orange"))
    orangeButton.grid(row=2,column=0)
    purpleButton=Button(drawFrame, bg="purple",width=2, \
                        command=lambda: colourChosen(drawWindow,canvas,
"purple"))
    purpleButton.grid(row=2,column=1)
    brownButton=Button(drawFrame, bg="brown",width=2,\
                       command=lambda: colourChosen(drawWindow,canvas,
"brown"))
    brownButton.grid(row=2,column=2)
    blackButton=Button(drawFrame, bg="black",width=2,\
                       command=lambda: colourChosen(drawWindow,canvas,
"black"))
    blackButton.grid(row=3,column=0)
    whiteButton=Button(drawFrame, bg="white",width=2, \
                       command=lambda: colourChosen(drawWindow,canvas,
"white"))
    whiteButton.grid(row=3,column=1)
    grayButton=Button(drawFrame, bg="gray",width=2,\
                      command=lambda: colourChosen(drawWindow,canvas,
"gray"))
    grayButton.grid(row=3,column=2)
    drawFrame.pack(side=BOTTOM)


def colourChosen(drawWindow, canvas, colour):
    if canvas.data.image!=None:
        canvas.data.drawColour=colour
        canvas.data.mainWindow.bind("<B1-Motion>",\
                                    lambda event: drawDraw(event, canvas))
    drawWindow.destroy()


def drawDraw(event, canvas):
    if canvas.data.drawOn==True:
        x=int(round((event.x-
canvas.data.imageTopX)*canvas.data.imageScale))
        y=int(round((event.y-
canvas.data.imageTopY)*canvas.data.imageScale))
        draw = ImageDraw.Draw(canvas.data.image)
        draw.ellipse((x-3, y-3, x+ 3, y+3), fill=canvas.data.drawColour,\
                     outline=None)
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)

######################## FEATURES ###########################

def closeHistWindow(canvas):
    if canvas.data.image!=None:
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.histWindowClose=True

def histogram(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    histWindow=Toplevel(canvas.data.mainWindow)
    histWindow.title("Histogram")
    canvas.data.histCanvasWidth=350
    canvas.data.histCanvasHeight=475
    histCanvas = Canvas(histWindow, width=canvas.data.histCanvasWidth, \
                        height=canvas.data.histCanvasHeight)
    histCanvas.pack()
    # provide sliders to the user to manipulate red, green and blue amounts
in the image
    redSlider=Scale(histWindow, from_=-100, to=100, \
                    orient=HORIZONTAL, label="R")
    redSlider.pack()
    blueSlider=Scale(histWindow, from_=-100, to=100,\
                     orient=HORIZONTAL, label="B")
    blueSlider.pack()
    greenSlider=Scale(histWindow, from_=-100, to=100,\
                      orient=HORIZONTAL, label="G")
    greenSlider.pack()
    OkHistFrame=Frame(histWindow)
    OkHistButton=Button(OkHistFrame, text="OK", \
                        command=lambda: closeHistWindow(canvas))
    OkHistButton.grid(row=0,column=0)
    OkHistFrame.pack(side=BOTTOM)
    initialRGB=(0,0,0)
    changeColours(canvas, redSlider, blueSlider, \
                  greenSlider, histWindow, histCanvas, initialRGB)


def changeColours(canvas, redSlider, blueSlider, \
                  greenSlider, histWindow, histCanvas, previousRGB):
    if canvas.data.histWindowClose==True:
        histWindow.destroy()
        canvas.data.histWindowClose=False
    else:
        # the slider value indicates the % by which the red/green/blue
        # value of the pixels of the image need to incresed (for +ve
values)
        # or decreased (for -ve values)
        if canvas.data.image!=None and histWindow.winfo_exists() :
            R, G, B= canvas.data.image.split()
            sliderValR=redSlider.get()
            (previousR, previousG, previousB)= previousRGB
            scaleR=(sliderValR-previousR)/100.0
            R=R.point(lambda i: i+ int(round(i*scaleR)))
            sliderValG=greenSlider.get()
            scaleG=(sliderValG-previousG)/100.0
            G=G.point(lambda i: i+ int(round(i*scaleG)))
            sliderValB=blueSlider.get()
            scaleB=(sliderValB-previousB)/100.0
            B=B.point(lambda i: i+ int(round(i*scaleB)))
            canvas.data.image = Image.merge(canvas.data.image.mode, (R, G,
B))
```

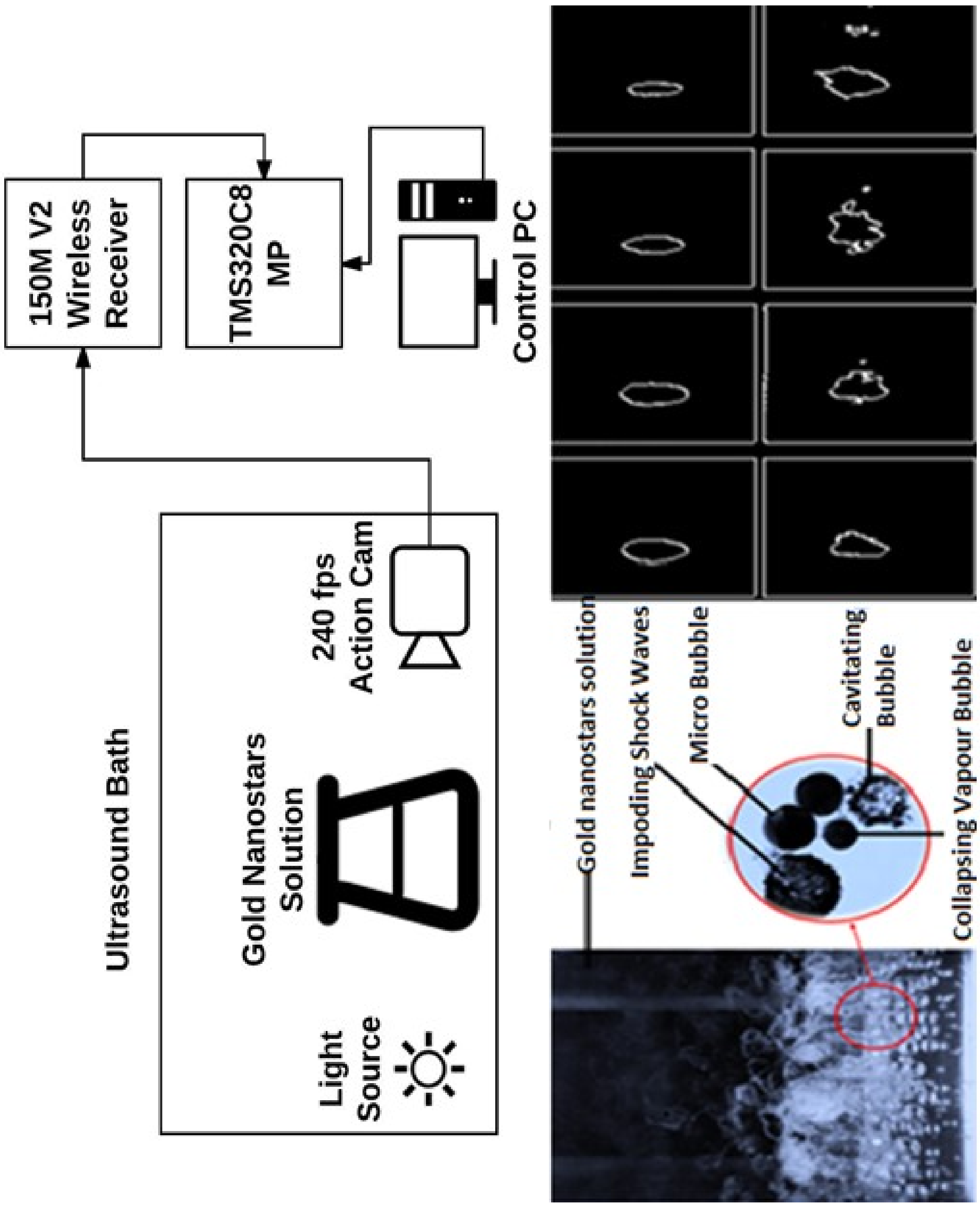
```
            canvas.data.imageForTk=makeImageForTk(canvas)
            drawImage(canvas)
            displayHistogram(canvas, histWindow, histCanvas)
            previousRGB=(sliderValR, sliderValG, sliderValB)
            canvas.after(200, lambda: changeColours(canvas, redSlider,\
                blueSlider, greenSlider,  histWindow, histCanvas,
previousRGB))

def displayHistogram(canvas,histWindow, histCanvas):
    histCanvasWidth=canvas.data.histCanvasWidth
    histCanvasHeight=canvas.data.histCanvasHeight
    margin=50
    if canvas.data.image!=None:
        histCanvas.delete(ALL)
        im=canvas.data.image
        #x-axis
        histCanvas.create_line(margin-1, histCanvasHeight-margin+1,\
                               margin-1+ 258, histCanvasHeight-margin+1)
        xmarkerStart=margin-1
        for i in xrange(0,257,64):
            xmarker="%d" % (i)
            histCanvas.create_text(xmarkerStart+i,\
                                   histCanvasHeight-margin+7, text=xmarker)
        #y-axis
        histCanvas.create_line(margin-1, \
                               histCanvasHeight-margin+1, margin-1, margin)
        ymarkerStart= histCanvasHeight-margin+1
        for i in xrange(0, histCanvasHeight-2*margin+1, 50):
            ymarker="%d" % (i)
            histCanvas.create_text(margin-1-10,\
                                   ymarkerStart-i, text=ymarker)

        R, G, B=im.histogram()[:256], im.histogram()[256:512], \
                 im.histogram()[512:768]
        for i in xrange(len(R)):
            pixelNo=R[i]
            histCanvas.create_oval(i+margin, \
                            histCanvasHeight-pixelNo/100.0-1-margin,
i+2+margin,\
                            histCanvasHeight-pixelNo/100.0+1-margin, \
                                   fill="red", outline="red")
        for i in xrange(len(G)):
            pixelNo=G[i]
            histCanvas.create_oval(i+margin, \
                            histCanvasHeight-pixelNo/100.0-1-margin,
i+2+margin,\
                            histCanvasHeight-pixelNo/100.0+1-margin, \
                                   fill="green", outline="green")
        for i in xrange(len(B)):
            pixelNo=B[i]
            histCanvas.create_oval(i+margin,\
                            histCanvasHeight-pixelNo/100.0-1-margin,
i+2+margin,\
                            histCanvasHeight-pixelNo/100.0+1-margin,\
                                   fill="blue", outline="blue")

def colourPop(canvas):
    canvas.data.cropPopToHappen=False
    canvas.data.colourPopToHappen=True
    canvas.data.drawOn=False
    tkMessageBox.showinfo(title="Colour Pop", message="Click on a part of
the image which you want in colour" , parent=canvas.data.mainWindow)
    if canvas.data.cropPopToHappen==False:
        canvas.data.mainWindow.bind("<ButtonPress-1>", lambda event:
getPixel(event, canvas))


def getPixel(event, canvas):
    # have to check if Colour Pop button is pressed or not, otherwise, the
root
    # events which point to different functions based on what button has
been
    # pressed will get mixed up
    try: # to avoid confusion between the diffrent events
        # asscoaited with crop and colourPop
        if canvas.data.colourPopToHappen==True and \
           canvas.data.cropPopToHappen==False and canvas.data.image!=None :
            data=[]
            # catch the location of the pixel selected by the user
            # multiply it by the scale to get pixel's olaction of the
            #actual image
            canvas.data.pixelx=\
            int(round((event.x-
canvas.data.imageTopX)*canvas.data.imageScale))
            canvas.data.pixely=\
            int(round((event.y-
canvas.data.imageTopY)*canvas.data.imageScale))
            pixelr, pixelg, pixelb= \
            canvas.data.image.getpixel((canvas.data.pixelx,
canvas.data.pixely))
            # the amount of deviation allowed from selected pixel's value
            tolerance=60
            for y in xrange(canvas.data.image.size[1]):
                for x in xrange(canvas.data.image.size[0]):
                    r, g, b= canvas.data.image.getpixel((x, y))
                    avg= int(round((r + g + b)/3.0))
                    # if the deviation of each pixel value > tolerance,
                    # make them gray else keep them coloured
                    if (abs(r-pixelr)>tolerance or
                        abs(g-pixelg)>tolerance or
                        abs(b-pixelb)>tolerance ):
                        R, G, B= avg, avg, avg
                    else:
                        R, G, B=r,g,b
                    data.append((R, G, B))
            canvas.data.image.putdata(data)
            save(canvas)
            canvas.data.undoQueue.append(canvas.data.image.copy())
            canvas.data.imageForTk=makeImageForTk(canvas)
            drawImage(canvas)
    except:
        pass
    canvas.data.colourPopToHappen=False


def crop(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.drawOn=False
    # have to check if crop button is pressed or not, otherwise,
    # the root events which point to
    # different functions based on what button has been pressed
    # will get mixed up
    canvas.data.cropPopToHappen=True
    tkMessageBox.showinfo(title="Crop", \
                          message="Draw cropping rectangle and press Enter"
,\
                          parent=canvas.data.mainWindow)
    if canvas.data.image!=None:
        canvas.data.mainWindow.bind("<ButtonPress-1>", \
                                    lambda event: startCrop(event, canvas))
        canvas.data.mainWindow.bind("<B1-Motion>",\
                                    lambda event: drawCrop(event, canvas))
        canvas.data.mainWindow.bind("<ButtonRelease-1>", \
                                    lambda event: endCrop(event, canvas))

def startCrop(event, canvas):
    # detects the start of the crop rectangle
    if canvas.data.endCrop==False and canvas.data.cropPopToHappen==True:
        canvas.data.startCropX=event.x
        canvas.data.startCropY=event.y

def drawCrop(event,canvas):
    # keeps extending the crop rectange as the user extends
    # his desired crop rectangle
    if canvas.data.endCrop==False and canvas.data.cropPopToHappen==True:
        canvas.data.tempCropX=event.x
        canvas.data.tempCropY=event.y
        canvas.create_rectangle(canvas.data.startCropX, \
                                canvas.data.startCropY,
                                 canvas.data.tempCropX, \
            canvas.data.tempCropY, fill="gray", stipple="gray12", width=0)

def endCrop(event, canvas):
    # set canvas.data.endCrop=True so that button pressed movements
    # are not caught anymore but set it to False when "Enter"
    # is pressed so that crop can be performed another time too
    if canvas.data.cropPopToHappen==True:
        canvas.data.endCrop=True
        canvas.data.endCropX=event.x
        canvas.data.endCropY=event.y
        canvas.create_rectangle(canvas.data.startCropX, \
                                canvas.data.startCropY,
                                 canvas.data.endCropX, \
            canvas.data.endCropY, fill="gray", stipple="gray12", width=0 )
        canvas.data.mainWindow.bind("<Return>", \
                                lambda event: performCrop(event, canvas))

def performCrop(event,canvas):
    canvas.data.image=\
    canvas.data.image.crop(\
    (int(round((canvas.data.startCropX-
canvas.data.imageTopX)*canvas.data.imageScale)),
    int(round((canvas.data.startCropY-
canvas.data.imageTopY)*canvas.data.imageScale)),
    int(round((canvas.data.endCropX-
canvas.data.imageTopX)*canvas.data.imageScale)),
    int(round((canvas.data.endCropY-
canvas.data.imageTopY)*canvas.data.imageScale))))
    canvas.data.endCrop=False
    canvas.data.cropPopToHappen=False
    save(canvas)
    canvas.data.undoQueue.append(canvas.data.image.copy())
    canvas.data.imageForTk=makeImageForTk(canvas)
    drawImage(canvas)



def rotateFinished(canvas, rotateWindow, rotateSlider, previousAngle):
    if canvas.data.rotateWindowClose==True:
        rotateWindow.destroy()
        canvas.data.rotateWindowClose=False
    else:
        if canvas.data.image!=None and rotateWindow.winfo_exists():
            canvas.data.angleSelected=rotateSlider.get()
            if canvas.data.angleSelected!= None and \
               canvas.data.angleSelected!= previousAngle:
                canvas.data.image=\
                canvas.data.image.rotate(float(canvas.data.angleSelected))
                canvas.data.imageForTk=makeImageForTk(canvas)
                drawImage(canvas)
        canvas.after(200, lambda:rotateFinished(canvas,\
                    rotateWindow, rotateSlider, canvas.data.angleSelected)
)
def closeRotateWindow(canvas):
    if canvas.data.image!=None:
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.rotateWindowClose=True

def rotate(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    rotateWindow=Toplevel(canvas.data.mainWindow)
    rotateWindow.title("Rotate")
    rotateSlider=Scale(rotateWindow, from_=0, to=360, orient=HORIZONTAL)
    rotateSlider.pack()
    OkRotateFrame=Frame(rotateWindow)
    OkRotateButton=Button(OkRotateFrame, text="OK",\
                          command=lambda: closeRotateWindow(canvas))
    OkRotateButton.grid(row=0,column=0)
    OkRotateFrame.pack(side=BOTTOM)
    rotateFinished(canvas, rotateWindow, rotateSlider, 0)

def closeBrightnessWindow(canvas):
    if canvas.data.image!=None:
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.brightnessWindowClose=True

def changeBrightness(canvas, brightnessWindow, brightnessSlider, \
                     previousVal):
    if canvas.data.brightnessWindowClose==True:
        brightnessWindow.destroy()
        canvas.data.brightnessWindowClose=False

    else:
        # increasing pixel values according to slider value increases
        #brightness we change ot according to the difference between the
        # previous value and the current slider value
        if canvas.data.image!=None and brightnessWindow.winfo_exists():
            sliderVal=brightnessSlider.get()
            scale=(sliderVal-previousVal)/100.0
            canvas.data.image=canvas.data.image.point(\
                lambda i: i+ int(round(i*scale)))
            canvas.data.imageForTk=makeImageForTk(canvas)
            drawImage(canvas)
            canvas.after(200, \
            lambda: changeBrightness(canvas, brightnessWindow, \
                                        brightnessSlider, sliderVal))
def brightness(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    brightnessWindow=Toplevel(canvas.data.mainWindow)
    brightnessWindow.title("Brightness")
    brightnessSlider=Scale(brightnessWindow, from_=-100, to=100,\
                           orient=HORIZONTAL)
    brightnessSlider.pack()
    OkBrightnessFrame=Frame(brightnessWindow)
    OkBrightnessButton=Button(OkBrightnessFrame, text="OK", \
                              command=lambda:
closeBrightnessWindow(canvas))
    OkBrightnessButton.grid(row=0,column=0)
    OkBrightnessFrame.pack(side=BOTTOM)
    changeBrightness(canvas, brightnessWindow, brightnessSlider,0)
    brightnessSlider.set(0)

def reset(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    ### change back to original image
    if canvas.data.image!=None:
        canvas.data.image=canvas.data.originalImage.copy()
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)

def mirror(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    if canvas.data.image!=None:
        canvas.data.image=ImageOps.mirror(canvas.data.image)
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)

def flip(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    if canvas.data.image!=None:
        canvas.data.image=ImageOps.flip(canvas.data.image)
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)


def transpose(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    # I treated the image as a continuous list of pixel values row-wise
    # and simply excnaged the rows and the coloums
    # in oder to make it rotate clockewise, I reversed all the rows
    if canvas.data.image!=None:
        imageData=list(canvas.data.image.getdata())
        newData=[]
        newimg=Image.new(canvas.data.image.mode,\
                (canvas.data.image.size[1], canvas.data.image.size[0]))
        for i in xrange(canvas.data.image.size[0]):
            addrow=[]
            for j in xrange(i, len(imageData), canvas.data.image.size[0]):
                addrow.append(imageData[j])
            addrow.reverse()
            newData+=addrow
        newimg.putdata(newData)
        canvas.data.image=newimg.copy()
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)

############### FILTERS #####################
    def covertGray(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    #### The existing method to convert to a grayscale image converts the
####
    ####         image mode, so I used my own function to convert
####
    # value of each channel of a pixel is set to the average of the
original
    # values of the channels
    if canvas.data.image!=None:
        data=[]
        for col in xrange(canvas.data.image.size[1]):
            for row in xrange(canvas.data.image.size[0]):
                r, g, b= canvas.data.image.getpixel((row, col))
                avg= int(round((r + g + b)/3.0))
                R, G, B= avg, avg, avg
                data.append((R, G, B))
        canvas.data.image.putdata(data)
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)

def sepia(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    # this method first converts the image to B&W and then adds the
    # same amount of red and green to every pixel
    if canvas.data.image!=None:
        sepiaData=[]
        for col in xrange(canvas.data.image.size[1]):
            for row in xrange(canvas.data.image.size[0]):
                r, g, b= canvas.data.image.getpixel((row, col))
                avg= int(round((r + g + b)/3.0))
                R, G, B= avg+100, avg+50, avg
                sepiaData.append((R, G, B))
        canvas.data.image.putdata(sepiaData)
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)

def invert(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    if canvas.data.image!=None:
        canvas.data.image=ImageOps.invert(canvas.data.image)
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)

def solarize(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    solarizeWindow=Toplevel(canvas.data.mainWindow)
    solarizeWindow.title("Solarize")
    solarizeSlider=Scale(solarizeWindow, from_=0, to=255,
orient=HORIZONTAL)
    solarizeSlider.pack()
    OkSolarizeFrame=Frame(solarizeWindow)
    OkSolarizeButton=Button(OkSolarizeFrame, text="OK",\
                            command=lambda: closeSolarizeWindow(canvas))
    OkSolarizeButton.grid(row=0,column=0)
    OkSolarizeFrame.pack(side=BOTTOM)
    ### beacsue intial silderVal=0
    performSolarize(canvas, solarizeWindow, solarizeSlider, 255)


def performSolarize(canvas, solarizeWindow, solarizeSlider,
previousThreshold):
    if canvas.data.solarizeWindowClose==True:
        solarizeWindow.destroy()
        canvas.data.solarizeWindowClose=False

    else:
        # the  slider denotes the % of solarization thta the user wants,
        # so the threshold (above which pixels are inverted) is inversely
        # related to the slider value
        if solarizeWindow.winfo_exists():
            sliderVal=solarizeSlider.get()
            threshold_=255-sliderVal
            if canvas.data.image!=None and threshold_!=previousThreshold:
                canvas.data.image=ImageOps.solarize(canvas.data.image,\
                                                    threshold=threshold_)
                canvas.data.imageForTk=makeImageForTk(canvas)
                drawImage(canvas)
            canvas.after(200, lambda: performSolarize(canvas, \
                                solarizeWindow, solarizeSlider,
threshold_))

def closeSolarizeWindow(canvas):
    if canvas.data.image!=None:
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.solarizeWindowClose=True

def posterize(canvas):
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.drawOn=False
    # we basically reduce the range of colurs from 256 to 5 bits
    # and so, assign a single new value to each colour value
    # in each succesive range
    posterData=[]
    if canvas.data.image!=None:
        for col in xrange(canvas.data.imageSize[1]):
            for row in xrange(canvas.data.imageSize[0]):
                r, g, b= canvas.data.image.getpixel((row, col))
                if r in xrange(32):
                    R=0
                elif r in xrange(32, 96):
                    R=64
                elif r in xrange(96, 160):
                    R=128
                elif r in xrange(160, 224):
                    R=192
                elif r in xrange(224,256):
                    R=255
                if g in xrange(32):
                    G=0
                elif g in xrange(32, 96):
                    G=64
                elif g in xrange(96, 160):
                    G=128
                elif r in xrange(160, 224):
                    g=192
                elif r in xrange(224,256):
                    G=255
                if b in xrange(32):
                    B=0
                elif b in xrange(32, 96):
                    B=64
                elif b in xrange(96, 160):
                    B=128
                elif b in xrange(160, 224):
                    B=192
                elif b in xrange(224,256):
                    B=255
                posterData.append((R, G, B))
        canvas.data.image.putdata(posterData)
        save(canvas)
        canvas.data.undoQueue.append(canvas.data.image.copy())
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)
################ EDIT MENU FUNCTIONS ############################
def keyPressed(canvas, event):
    if event.keysym=="z":
        undo(canvas)
    elif event.keysym=="y":
        redo(canvas)
# we use deques so as to make Undo and Redo more efficient and avoid
# memory space isuues
# after each change, we append the new version of the image to
# the Undo queue
def undo(canvas):
    if len(canvas.data.undoQueue)>0:
        # the last element of the Undo Deque is the
        # current version of the image
        lastImage=canvas.data.undoQueue.pop()
        # we would want the current version if wehit redo after undo
        canvas.data.redoQueue.appendleft(lastImage)
    if len(canvas.data.undoQueue)>0:
        # the previous version of the image
        canvas.data.image=canvas.data.undoQueue[-1]
    save(canvas)
    canvas.data.imageForTk=makeImageForTk(canvas)
    drawImage(canvas)

def redo(canvas):
    if len(canvas.data.redoQueue)>0:
        canvas.data.image=canvas.data.redoQueue[0]
    save(canvas)
    if len(canvas.data.redoQueue)>0:
        # we remove this version from the Redo Deque beacuase it
        # has become our current image
        lastImage=canvas.data.redoQueue.popleft()
        canvas.data.undoQueue.append(lastImage)
    canvas.data.imageForTk=makeImageForTk(canvas)
    drawImage(canvas)

############# MENU COMMANDS ################
def saveAs(canvas):
    # ask where the user wants to save the file
    if canvas.data.image!=None:
        filename=asksaveasfilename(defaultextension=".jpg")
        im=canvas.data.image
        im.save(filename)

def save(canvas):
    if canvas.data.image!=None:
        im=canvas.data.image
        im.save(canvas.data.imageLocation)

def newImage(canvas):
    imageName=askopenfilename()
    filetype=""
    #make sure it's an image file
    try: filetype=imghdr.what(imageName)
    except:
        tkMessageBox.showinfo(title="Image File",\
        message="Choose an Image File!" , parent=canvas.data.mainWindow)
    # restrict filetypes to .jpg, .bmp, etc.
    if filetype in ['jpeg', 'bmp', 'png', 'tiff']:
        canvas.data.imageLocation=imageName
        im= Image.open(imageName)
        canvas.data.image=im
        canvas.data.originalImage=im.copy()
        canvas.data.undoQueue.append(im.copy())
        canvas.data.imageSize=im.size #Original Image dimensions
        canvas.data.imageForTk=makeImageForTk(canvas)
        drawImage(canvas)
    else:
        tkMessageBox.showinfo(title="Image File",\
        message="Choose an Image File!" , parent=canvas.data.mainWindow)
```

```
######## CREATE A VERSION OF IMAGE TO BE DISPLAYED ON THE CANVAS #########
def makeImageForTk(canvas):
    im=canvas.data.image
    if canvas.data.image!=None:
        # Beacuse after cropping the now 'image' might have diffrent
        # dimensional ratios
        imageWidth=canvas.data.image.size[0]
        imageHeight=canvas.data.image.size[1]
        #To make biggest version of the image fit inside the canvas
        if imageWidth>imageHeight:
            resizedImage=im.resize((canvas.data.width,\

int(round(float(imageHeight)*canvas.data.width/imageWidth))))
            # store the scale so as to use it later
            canvas.data.imageScale=float(imageWidth)/canvas.data.width
        else:

resizedImage=im.resize((int(round(float(imageWidth)*canvas.data.height/imag
eHeight)),\
                                    canvas.data.height))
            canvas.data.imageScale=float(imageHeight)/canvas.data.height
        # we may need to refer to ther resized image atttributes again
        canvas.data.resizedIm=resizedImage
        return ImageTk.PhotoImage(resizedImage)

def drawImage(canvas):
    if canvas.data.image!=None:
        # make the canvas center and the image center the same
        canvas.create_image(canvas.data.width/2.0-
canvas.data.resizedIm.size[0]/2.0,
                        canvas.data.height/2.0-
canvas.data.resizedIm.size[1]/2.0,
                            anchor=NW, image=canvas.data.imageForTk)
        canvas.data.imageTopX=int(round(canvas.data.width/2.0-
canvas.data.resizedIm.size[0]/2.0))
        canvas.data.imageTopY=int(round(canvas.data.height/2.0-
canvas.data.resizedIm.size[1]/2.0))

############# DESKTOP BK ##############
## Please comment this function out if you use this on any OS apart from
Windows

def desktopBk(canvas):
    if canvas.data.image!=None:
        new=canvas.data.image.copy()
        # Windows desktop photos are supposed to be bitmap images
        newLocation=os.path.dirname(\
            canvas.data.imageLocation)+"/desktopPhoto.bmp"
        new.save(newLocation)
        SPI_SETDESKWALLPAPER = 20
        ctypes.windll.user32.SystemParametersInfoA(SPI_SETDESKWALLPAPER, 0,
str(newLocation), 0)
########### INITIALIZE ##############
def init(root, canvas):
    buttonsInit(root, canvas)
    menuInit(root, canvas)
    canvas.data.image=None
    canvas.data.angleSelected=None
    canvas.data.rotateWindowClose=False
    canvas.data.brightnessWindowClose=False
    canvas.data.brightnessLevel=None
    canvas.data.histWindowClose=False
    canvas.data.solarizeWindowClose=False
    canvas.data.posterizeWindowClose=False
    canvas.data.colourPopToHappen=False
    canvas.data.cropPopToHappen=False
    canvas.data.endCrop=False
    canvas.data.drawOn=True

    canvas.data.undoQueue=deque([], 10)
    canvas.data.redoQueue=deque([], 10)
    canvas.pack()

def buttonsInit(root, canvas):
    backgroundColour="white"
    buttonWidth=14
    buttonHeight=2
    toolKitFrame=Frame(root)
    cropButton=Button(toolKitFrame, text="Crop",\
                      background=backgroundColour ,\
                      width=buttonWidth, height=buttonHeight, \
                      command=lambda:crop(canvas))
    cropButton.grid(row=0,column=0)
    rotateButton=Button(toolKitFrame, text="Rotate",\
                        background=backgroundColour, \
                        width=buttonWidth,height=buttonHeight, \
                        command=lambda: rotate(canvas))
    rotateButton.grid(row=1,column=0)
    brightnessButton=Button(toolKitFrame, text="Brightness",\
                            background=backgroundColour ,\
                            width=buttonWidth, height=buttonHeight,\
                            command=lambda: brightness(canvas))
    brightnessButton.grid(row=2,column=0)
    histogramButton=Button(toolKitFrame, text="Histogram",\
                           background=backgroundColour ,\
                           width=buttonWidth,height=buttonHeight, \
                           command=lambda: histogram(canvas))
    histogramButton.grid(row=3,column=0)
    colourPopButton=Button(toolKitFrame, text="Colour Pop",\
                           background=backgroundColour, \
                           width=buttonWidth,height=buttonHeight, \
                           command=lambda: colourPop(canvas))
    colourPopButton.grid(row=4,column=0)
    mirrorButton=Button(toolKitFrame, text="Mirror",\
                        background=backgroundColour, \
                        width=buttonWidth,height=buttonHeight, \
                        command=lambda: mirror(canvas))
    mirrorButton.grid(row=5,column=0)
    flipButton=Button(toolKitFrame, text="Flip",\
                      background=backgroundColour ,\
                      width=buttonWidth,height=buttonHeight, \
                      command=lambda: flip(canvas))
    flipButton.grid(row=6,column=0)
    transposeButton=Button(toolKitFrame, text="Transpose",\
                           background=backgroundColour, width=buttonWidth,\
                           height=buttonHeight,command=lambda:
transpose(canvas))
    transposeButton.grid(row=7,column=0)
    drawButton=Button(toolKitFrame, text="Draw",\
                      background=backgroundColour ,width=buttonWidth,\
                      height=buttonHeight,command=lambda:
drawOnImage(canvas))
```

```
    drawButton.grid(row=8,column=0)
    resetButton=Button(toolKitFrame, text="Reset",\
                       background=backgroundColour ,width=buttonWidth,\
                       height=buttonHeight, command=lambda: reset(canvas))
    resetButton.grid(row=9,column=0)
    #Please comment this button out if you use this on any OS apart from
Windows
    desktopButton=Button(toolKitFrame, text="Make Desktop Bk",\
                          background=backgroundColour,height=buttonHeight,\
                          width=buttonWidth,command=lambda:
desktopBk(canvas))
    desktopButton.grid(row=10,column=0)
    toolKitFrame.pack(side=LEFT)

def menuInit(root, canvas):
    menubar=Menu(root)
    menubar.add_command(label="New", command=lambda:newImage(canvas))
    menubar.add_command(label="Save", command=lambda:save(canvas))
    menubar.add_command(label="Save As", command=lambda:saveAs(canvas))
    ## Edit pull-down Menu
    editmenu = Menu(menubar, tearoff=0)
    editmenu.add_command(label="Undo   Z", command=lambda:undo(canvas))
    editmenu.add_command(label="Redo   Y", command=lambda:redo(canvas))
    menubar.add_cascade(label="Edit", menu=editmenu)
    root.config(menu=menubar)
    ## Filter pull-down Menu
    filtermenu = Menu(menubar, tearoff=0)
    filtermenu.add_command(label="Black and White", \
                           command=lambda:covertGray(canvas))
    filtermenu.add_command(label="Sepia",\
                           command=lambda:sepia(canvas))
    filtermenu.add_command(label="Invert", \
                           command=lambda:invert(canvas))
    filtermenu.add_command(label="Solarize", \
                           command=lambda:solarize(canvas))
    filtermenu.add_command(label="Posterize", \
                           command=lambda:posterize(canvas))
    menubar.add_cascade(label="Filter", menu=filtermenu)
    root.config(menu=menubar)



def run():
    # create the root and the canvas
    root = Tk()
    root.title("Image Editor")
    canvasWidth=500
    canvasHeight=500
    canvas = Canvas(root, width=canvasWidth, height=canvasHeight, \
                    background="gray")
    # Set up canvas data and call init
    class Struct: pass
    canvas.data = Struct()
    canvas.data.width=canvasWidth
    canvas.data.height=canvasHeight
    canvas.data.mainWindow=root
    init(root, canvas)
    root.bind("<Key>", lambda event:keyPressed(canvas, event))
    # and launch the app
    root.mainloop()  # This call BLOCKS (so your program waits)
run()
```

APPENDIX B

Low cost cavitation image capturing experimental setup


Ultrasound Bath
Light Source
Gold Nanostars Solution
240 fps Action Cam
150M V2 Wireless Receiver
TMS320C8 MP
Control PC
Gold nanostars solution
Impoding Shock Waves
Micro Bubble
Cavitating Bubble
Collapsing Vapour Bubble

Code:

```
import os
import glob
import shutil
import sys
import logging
from threading import Thread
import time
import datetime

mypath = os.path.abspath(__file__)   # Find the full path of this python
script
baseDir = mypath[0:mypath.rfind("/")+1]  # get the path location only
(excluding script name)
baseFileName = mypath[mypath.rfind("/")+1:mypath.rfind(".")]
progName = os.path.basename(__file__)
# Color data for OpenCV lines and text
cvWhite = (255, 255, 255)
cvBlack = (0, 0, 0)
cvBlue = (255, 0, 0)
cvGreen = (0, 255, 0)
cvRed = (0, 0, 255)
# Check for variable file to import and error out if not found.
configFilePath = baseDir + "config.py"
if not os.path.exists(configFilePath):
    print("ERROR : Missing config.py file - Could not find Configuration
file %s"
          % configFilePath)
    import urllib2
    config_url = "https://raw.github.com/pageauc/speed-
camera/master/config.py"
    print("INFO  : Attempting to Download config.py file from %s" %
config_url)
    try:
        wgetfile = urllib2.urlopen(config_url)
    except:
        print("ERROR : Download of config.py Failed")
        print("        Try Rerunning the speed-install.sh Again.")
        print("        or")
        print("        Perform GitHub curl install per Readme.md")
        print("        and Try Again.")
        print("        Exiting %s" % progName)
        sys.exit(1)
    f = open('config.py', 'wb')
    f.write(wgetfile.read())
    f.close()
# Read Configuration variables from config.py file
from config import *
from search_config import search_dest_path
if pluginEnable:     # Check and verify plugin and load variable overlay
    pluginDir = os.path.join(baseDir, "plugins")
    if pluginName.endswith('.py'):   # Check if there is a .py at the end
of pluginName variable
        pluginName = pluginName[:-3]    # Remove .py extensiion
    pluginPath = os.path.join(pluginDir, pluginName + '.py')
    print("INFO  : pluginEnabled - loading pluginName %s" % pluginPath)
    if not os.path.isdir(pluginDir):
        print("ERROR : plugin Directory Not Found at %s" % pluginDir)
        print("        Suggest you Rerun github curl install script to
install plugins")
```

```
        print("        https://github.com/pageauc/pi-timolo/wiki/How-to-
Install-or-Upgrade#quick-install")
        print("        Exiting %s Due to Error" % progName)
        sys.exit(1)
    elif not os.path.exists(pluginPath):
        print("ERROR : File Not Found pluginName %s" % pluginPath)
        print("        Check Spelling of pluginName Value in %s" %
configFilePath)
        print("        ------- Valid Names -------")
        validPlugin = glob.glob(pluginDir + "/*py")
        validPlugin.sort()
        for entry in validPlugin:
            pluginFile = os.path.basename(entry)
            plugin = pluginFile.rsplit('.', 1)[0]
            if not ((plugin == "__init__") or (plugin == "current")):
                print("        %s"  % plugin)
        print("        ------- End of List -------")
        print("        Note: pluginName Should Not have .py Ending.")
        print("INFO  : or Rerun github curl install command.  See github
wiki")
        print("        https://github.com/pageauc/speed-camera/wiki/How-to-
Install-or-Upgrade#quick-install")
        print("        Exiting %s Due to Error" % progName)
        sys.exit(1)
    else:
        pluginCurrent = os.path.join(pluginDir, "current.py")
        try:    # Copy image file to recent folder
            print("INFO  : Copy %s to %s" % (pluginPath, pluginCurrent))
            shutil.copy(pluginPath, pluginCurrent)
        except OSError as err:
            print('ERROR : Copy Failed from %s to %s - %s'
                  % (pluginPath, pluginCurrent, err))
            print("        Check permissions, disk space, Etc.")
            print("        Exiting %s Due to Error" % progName)
            sys.exit(1)
        print("INFO  : Import Plugin %s" % pluginPath)
        sys.path.insert(0, pluginDir)    # add plugin directory to program
PATH
        from plugins.current import *
        try:
            if os.path.exists(pluginCurrent):
                os.remove(pluginCurrent)
            pluginCurrentpyc = os.path.join(pluginDir, "current.pyc")
            if os.path.exists(pluginCurrentpyc):
                os.remove(pluginCurrentpyc)
        except OSError as err:
            print("WARN  : Failed To Remove File %s - %s"
                  % (pluginCurrentpyc, err))
            print("        Exiting %s Due to Error" % progName)
else:
    print("INFO  : No Plugins Enabled per pluginEnable=%s" % pluginEnable)

# Now that variables are imported from config.py Setup Logging
if loggingToFile:
    logging.basicConfig(level=logging.DEBUG,
                        format='%(asctime)s %(levelname)-8s %(funcName)-10s
%(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S',
                        filename=logFilePath,
                        filemode='w')
elif verbose:
    logging.basicConfig(level=logging.DEBUG,
                        format='%(asctime)s %(levelname)-8s %(funcName)-10s
%(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S')
    logging.info("Logging to Console per Variable verbose=True")
else:
    logging.basicConfig(level=logging.CRITICAL,
                        format='%(asctime)s %(levelname)-8s %(funcName)-10s
%(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S')
    logging.info("Logging Disabled per Variable verbose=False")
# import the necessary packages
# -----------------------------
try:  #Add this check in case running on non RPI platform using web cam
    from picamera.array import PiRGBArray
    from picamera import PiCamera
except ImportError:
    WEBCAM = True

import subprocess
if not WEBCAM:
    # Check that pi camera module is installed and enabled
    camResult = subprocess.check_output("vcgencmd get_camera", shell=True)
    camResult = camResult.decode("utf-8")
    camResult = camResult.replace("\n", "")
    if (camResult.find("0")) >= 0:   # -1 is zero not found. Cam OK
        logging.error("Pi Camera Module Not Found %s", camResult)
        logging.error("if supported=0 Enable Camera using command sudo
raspi-config")
        logging.error("if detected=0 Check Pi Camera Module is Installed
Correctly")
        logging.error("Exiting %s", progName)
        sys.exit(1)
    else:
        logging.info("Camera Module is Enabled and Connected %s",
camResult)
try:   # Check to see if opencv is installed
    import cv2
except ImportError:
    logging.error("Could not import cv2 library")
    if sys.version_info > (2, 9):
        logging.error("python3 failed to import cv2")
        logging.error("Try installing opencv for python3")
        logging.error("For RPI See https://github.com/pageauc/opencv3-
setup")
    else:
        logging.error("python2 failed to import cv2")
        logging.error("Try RPI Install per command")
    logging.error("INFO  : Exiting %s", progName)
    sys.exit(1)
# fix possible invalid values
if WINDOW_BIGGER < 1:
    WINDOW_BIGGER = 1
if image_bigger < 1:
    image_bigger = 1
# System Settings
image_width = int(CAMERA_WIDTH * image_bigger)   # Set width of trigger
point image to save
image_height = int(CAMERA_HEIGHT * image_bigger) # Set height of trigger
point image to save
# Calculate conversion from camera pixel width to actual speed.
```

```
px_to_kph = float(cal_obj_mm/cal_obj_px * 0.0036)
quote = '"'  # Used for creating quote delimited log file of speed data
if SPEED_MPH:
    speed_units = "mph"
    speed_conv = 0.621371 * px_to_kph
else:
    speed_units = "kph"
    speed_conv = px_to_kph

#---------------------------------------------------------------------------
---------------------
class PiVideoStream:
    def __init__(self, resolution=(CAMERA_WIDTH, CAMERA_HEIGHT),
framerate=CAMERA_FRAMERATE,
                 rotation=0, hflip=CAMERA_HFLIP, vflip=CAMERA_VFLIP):
        # initialize the camera and stream
        try:
            self.camera = PiCamera()
        except:
            logging.error("PiCamera Already in Use by Another Process")
            logging.error("Exit %s", progName)
            sys.exit(1)
        self.camera.resolution = resolution
        self.camera.rotation = rotation
        self.camera.framerate = framerate
        self.camera.hflip = hflip
        self.camera.vflip = vflip
        self.rawCapture = PiRGBArray(self.camera, size=resolution)
        self.stream = self.camera.capture_continuous(self.rawCapture,
                                                     format="bgr",
                                                     use_video_port=True)
        # initialize the frame and the variable used to indicate
        # if the thread should be stopped
        self.frame = None
        self.stopped = False

    def start(self):
        # start the thread to read frames from the video stream
        t = Thread(target=self.update, args=())
        t.daemon = True
        t.start()
        return self

    def update(self):
        # keep looping infinitely until the thread is stopped
        for f in self.stream:
            # grab the frame from the stream and clear the stream in
            # preparation for the next frame
            self.frame = f.array
            self.rawCapture.truncate(0)

            # if the thread indicator variable is set, stop the thread
            # and resource camera resources
            if self.stopped:
                self.stream.close()
                self.rawCapture.close()
                self.camera.close()
                return

    def read(self):
        # return the frame most recently read
        return self.frame

    def stop(self):
        # indicate that the thread should be stopped
        self.stopped = True

#----------------------------------------------------------------------------
---------------------
class WebcamVideoStream:
    def __init__(self, CAM_SRC=WEBCAM_SRC, CAM_WIDTH=WEBCAM_WIDTH,
CAM_HEIGHT=WEBCAM_HEIGHT):
        # initialize the video camera stream and read the first frame
        # from the stream
        self.stream = CAM_SRC
        self.stream = cv2.VideoCapture(CAM_SRC)
        self.stream.set(3, CAM_WIDTH)
        self.stream.set(4, CAM_HEIGHT)
        (self.grabbed, self.frame) = self.stream.read()
        # initialize the variable used to indicate if the thread should
        # be stopped
        self.stopped = False

    def start(self):
        # start the thread to read frames from the video stream
        t = Thread(target=self.update, args=())
        t.daemon = True
        t.start()
        return self

    def update(self):
        # keep looping infinitely until the thread is stopped
        while True:
            # if the thread indicator variable is set, stop the thread
            if self.stopped:
                return
            # otherwise, read the next frame from the stream
            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
        # return the frame most recently read
        return self.frame

    def stop(self):
        # indicate that the thread should be stopped
        self.stopped = True

#----------------------------------------------------------------------------
--------------------
def get_fps(start_time, frame_count):
    # Calculate and display frames per second processing
    if frame_count >= 1000:
        duration = float(time.time() - start_time)
        FPS = float(frame_count / duration)
        logging.info("%.2f fps Last %i Frames", FPS, frame_count)
        frame_count = 0
        start_time = time.time()
    else:
        frame_count += 1
    return start_time, frame_count
```

```
#-----------------------------------------------------------------------------
---------------------
def show_settings():
    """Initialize and Display program variable settings from config.py"""
    cwd = os.getcwd()
    html_path = "media/html"
    if not os.path.isdir(image_path):
        logging.info("Creating Image Storage Folder %s", image_path)
        os.makedirs(image_path)
    os.chdir(image_path)
    os.chdir(cwd)
    if imageRecentMax > 0:
        if not os.path.isdir(imageRecentDir):
            logging.info("Create Recent Folder %s", imageRecentDir)
            try:
                os.makedirs(imageRecentDir)
            except OSError as err:
                logging.error('Failed to Create Folder %s - %s',
imageRecentDir, err)
    if not os.path.isdir(search_dest_path):
        logging.info("Creating Search Folder %s", search_dest_path)
        os.makedirs(search_dest_path)
    if not os.path.isdir(html_path):
        logging.info("Creating html Folder %s", html_path)
        os.makedirs(html_path)
    os.chdir(cwd)
    if verbose:
        print("-----------------------------------------------------------------
--------------------")
        print("Note: To Send Full Output to File Use command")
        print("python -u ./%s | tee -a log.txt" % progName)
        print("Set log_data_to_file=True to Send speed_Data to CSV File
%s.log"
              % baseFileName)
        print("--------------------------------- Settings -----------------
--------------------")
        print("")
        print("Plugins ......... pluginEnable=%s  pluginName=%s"
              % (pluginEnable, pluginName))
        print("Message Display . verbose=%s  display_fps=%s calibrate=%s"
              % (verbose, display_fps, calibrate))
        print("                  show_out_range=%s" % show_out_range)
        print("Logging ......... Log_data_to_CSV=%s  log_filename=%s.csv
(CSV format)"
              % (log_data_to_CSV, baseFileName))
        print("                  loggingToFile=%s  logFilePath=%s"
              % (loggingToFile, logFilePath))
        print("                  Log if max_speed_over > %i %s"
              % (max_speed_over, speed_units))
        print("Speed Trigger ... If  track_len_trig > %i px" %
track_len_trig)
        print("Exclude Events .. If  x_diff_min < %i or x_diff_max > %i px"
              % (x_diff_min, x_diff_max))
        print("                  If  y_upper < %i or y_lower > %i px"
              % (y_upper, y_lower))
        print("                  or  x_left < %i or x_right > %i px"
              % (x_left, x_right))
        print("                  If  max_speed_over < %i %s"
              % (max_speed_over, speed_units))
        print("                  If  event_timeout > %i seconds Start New
Track"
```

```
             % (event_timeout))
        print("                  track_timeout=%i sec wait after Track
Ends"
              " (avoid retrack of same object)"
              % (track_timeout))
        print("Speed Photo ..... Size=%ix%i px  image_bigger=%i"
              "  rotation=%i  VFlip=%s  HFlip=%s "
              % (image_width, image_height, image_bigger,
                 CAMERA_ROTATION, CAMERA_VFLIP, CAMERA_HFLIP))
        print("                  image_path=%s  image_Prefix=%s"
              % (image_path, image_prefix))
        print("                  image_font_size=%i px high
image_text_bottom=%s"
              % (image_font_size, image_text_bottom))
        print("Motion Settings . Size=%ix%i px  px_to_kph=%f
speed_units=%s"
              % (CAMERA_WIDTH, CAMERA_HEIGHT, px_to_kph, speed_units))
        print("OpenCV Settings . MIN_AREA=%i sq-px  BLUR_SIZE=%i"
              "  THRESHOLD_SENSITIVITY=%i  CIRCLE_SIZE=%i px"
              % (MIN_AREA, BLUR_SIZE, THRESHOLD_SENSITIVITY, CIRCLE_SIZE))
        print("                  WINDOW_BIGGER=%i gui_window_on=%s"
              " (Display OpenCV Status Windows on GUI Desktop)"
              % (WINDOW_BIGGER, gui_window_on))
        print("                  CAMERA_FRAMERATE=%i fps video stream
speed"
              % CAMERA_FRAMERATE)
        print("Sub-Directories . imageSubDirMaxHours=%i (0=off)"
              "  imageSubDirMaxFiles=%i (0=off)"
              % (imageSubDirMaxHours, imageSubDirMaxFiles))
        print("                  imageRecentDir=%s imageRecentMax=%i
(0=off)"
              % (imageRecentDir, imageRecentMax))
        if spaceTimerHrs > 0:   # Check if disk mgmnt is enabled
            print("Disk Space  ..... Enabled - Manage Target Free Disk
Space."
                  " Delete Oldest %s Files if Needed" % (spaceFileExt))
            print("                  Check Every spaceTimerHrs=%i hr(s)
(0=off)"
                  "  Target spaceFreeMB=%i MB  min is 100 MB)"
                  % (spaceTimerHrs, spaceFreeMB))
            print("                  If Needed Delete Oldest
spaceFileExt=%s  spaceMediaDir=%s"
                  % (spaceFileExt, spaceMediaDir))
        else:
            print("Disk Space  ..... Disabled - spaceTimerHrs=%i"
                  "  Manage Target Free Disk Space. Delete Oldest %s Files"
                  % (spaceTimerHrs, spaceFileExt))
            print("            ..... spaceTimerHrs=%i (0=Off)"
                  "  Target spaceFreeMB=%i (min=100 MB)"
                  % (spaceTimerHrs, spaceFreeMB))
        print("")
        print("--------------------------------------------------------------
---------------------")
    return

#---------------------------------------------------------------------------
---------------------
def take_calibration_image(filename, cal_image):
    """
    Create a calibration image for determining value of IMG_VIEW_FT
variable
```

```
    Create calibration hash marks
    """
    for i in range(10, CAMERA_WIDTH - 9, 10):
        cv2.line(cal_image, (i, y_upper - 5), (i, y_upper + 30), cvRed, 1)
    # This is motion window
    cv2.line(cal_image, (x_left, y_upper), (x_right, y_upper), cvBlue, 1)
    cv2.line(cal_image, (x_left, y_lower), (x_right, y_lower), cvBlue, 1)
    cv2.line(cal_image, (x_left, y_upper), (x_left, y_lower), cvBlue, 1)
    cv2.line(cal_image, (x_right, y_upper), (x_right, y_lower), cvBlue, 1)
    print("")
    print("---------------------------- Create Calibration Image ---------
--------------------")
    print("")
    print("    Instructions for using %s image for camera calibration" %
filename)
    print("")
    print("1 - Use a known size reference object in the image like a
vehicle")
    print("    at the required distance.")
    print("2 - Record cal_obj_px  Value using Red y_upper hash marks at
every 10 px")
    print("3 - Record cal_obj_mm of object. This is Actual length in mm of
object above")
    print("4 - Edit config.py and enter the values for the above
variables.")
    print("")
    print("    Calibration Image Saved To %s%s" % (baseDir, filename))
    print("")
    print("---------------------- Press cntl-c to Quit Calibration Mode ---
--------------------")
    print("")
    return cal_image

#----------------------------------------------------------------------------
---------------------
def subDirLatest(directory): # Scan for directories and return most recent
    dirList = [name for name in os.listdir(directory) if
os.path.isdir(os.path.join(directory, name))]
    if len(dirList) > 0:
        lastSubDir = sorted(dirList)[-1]
        lastSubDir = os.path.join(directory, lastSubDir)
    else:
        lastSubDir = directory
    return lastSubDir

#----------------------------------------------------------------------------
---------------------
def subDirCreate(directory, prefix):
    now = datetime.datetime.now()
    # Specify folder naming
    subDirName = ('%s%d%02d%02d-%02d%02d' %
                  (prefix, now.year, now.month, now.day, now.hour,
now.minute))
    subDirPath = os.path.join(directory, subDirName)
    if not os.path.exists(subDirPath):
        try:
            os.makedirs(subDirPath)
        except OSError as err:
            logging.error('Cannot Create Directory %s - %s, using default
location.',
                          subDirPath, err)
            subDirPath = directory
        else:
            logging.info('Created %s', subDirPath)
    else:
        subDirPath = directory
    return subDirPath

#---------------------------------------------------------------------------
---------------------
def deleteOldFiles(maxFiles, dirPath, prefix):
    # Delete Oldest files gt or eq to maxfiles that match filename prefix
    try:
        fileList = sorted(glob.glob(os.path.join(dirPath, prefix + '*')),
                          key=os.path.getmtime)
    except OSError as err:
        logging.error('Problem Reading Directory %s - %s', dirPath, err)
    else:
        while len(fileList) >= maxFiles:
            oldest = fileList[0]
            oldestFile = oldest
            try:   # Remove oldest file in recent folder
                fileList.remove(oldest)
                os.remove(oldestFile)
            except OSError as err:
                logging.error('Cannot Remove %s - %s', oldestFile, err)

#---------------------------------------------------------------------------
---------------------
def subDirCheckMaxFiles(directory, filesMax):  # Count number of files in a
folder path
    fileList = glob.glob(directory + '/*jpg')
    count = len(fileList)
    if count > filesMax:
        makeNewDir = True
        logging.info('Total Files in %s Exceeds %i ', directory, filesMax)
    else:
        makeNewDir = False
    return makeNewDir

#---------------------------------------------------------------------------
---------------------
def subDirCheckMaxHrs(directory, hrsMax, prefix):   # Note to self need to
add error checking
    # extract the date-time from the directory name
    dirName = os.path.split(directory)[1]   # split dir path and keep
dirName
    dirStr = dirName.replace(prefix, '')   # remove prefix from dirName so
just date-time left
    dirDate = datetime.datetime.strptime(dirStr, "%Y-%m-%d-%H:%M")  #
convert string to datetime
    rightNow = datetime.datetime.now()   # get datetime now
    diff = rightNow - dirDate           # get time difference between dates
    days, seconds = diff.days, diff.seconds
    dirAgeHours = days * 24 + seconds // 3600  # convert to hours
    if dirAgeHours > hrsMax:   # See if hours are exceeded
        makeNewDir = True
        logging.info('MaxHrs %i Exceeds %i for %s', dirAgeHours, hrsMax,
directory)
    else:
        makeNewDir = False
    return makeNewDir
```

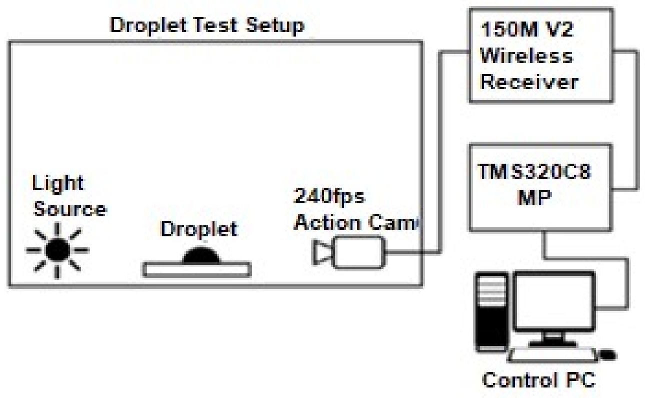
```
#-----------------------------------------------------------------------------------------
def subDirChecks(maxHours, maxFiles, directory, prefix):
    # Check if motion SubDir needs to be created
    if maxHours < 1 and maxFiles < 1:  # No Checks required
        # logging.info('No sub-folders Required in %s', directory)
        subDirPath = directory
    else:
        subDirPath = subDirLatest(directory)
        if subDirPath == directory:   # No subDir Found
            logging.info('No sub folders Found in %s', directory)
            subDirPath = subDirCreate(directory, prefix)
        elif (maxHours > 0 and maxFiles < 1):   # Check MaxHours Folder Age
Only
            if subDirCheckMaxHrs(subDirPath, maxHours, prefix):
                subDirPath = subDirCreate(directory, prefix)
        elif (maxHours < 1 and maxFiles > 0):   # Check Max Files Only
            if subDirCheckMaxFiles(subDirPath, maxFiles):
                subDirPath = subDirCreate(directory, prefix)
        elif maxHours > 0 and maxFiles > 0:   # Check both Max Files and
Age
            if subDirCheckMaxHrs(subDirPath, maxHours, prefix):
                if subDirCheckMaxFiles(subDirPath, maxFiles):
                    subDirPath = subDirCreate(directory, prefix)
                else:
                    logging.info('MaxFiles Not Exceeded in %s', subDirPath)
    os.path.abspath(subDirPath)
    return subDirPath

#-----------------------------------------------------------------------------------------
def filesToDelete(mediaDirPath, extension=image_format):
    return sorted(
        (os.path.join(dirname, filename)
         for dirname, dirnames, filenames in os.walk(mediaDirPath)
         for filename in filenames
         if filename.endswith(extension)),
        key=lambda fn: os.stat(fn).st_mtime, reverse=True)

#-----------------------------------------------------------------------------------------
def saveRecent(recentMax, recentDir, filename, prefix):
    """ save specified most recent files (timelapse and/or motion) in
recent subfolder"""
    deleteOldFiles(recentMax, recentDir, prefix)
    try:    # Copy image file to recent folder
        shutil.copy(filename, recentDir)
    except OSError as err:
        logging.error('Copy from %s to %s - %s', filename, recentDir, err)

#-----------------------------------------------------------------------------------------
def freeSpaceUpTo(freeMB, mediaDir, extension=image_format):
    """ Walks mediaDir and deletes oldest files until spaceFreeMB is
achieved
    Use with Caution """
    mediaDirPath = os.path.abspath(mediaDir)
    if os.path.isdir(mediaDirPath):
        MB2Bytes = 1048576  # Conversion from MB to Bytes
        targetFreeBytes = freeMB * MB2Bytes
        fileList = filesToDelete(mediaDir, extension)
        totFiles = len(fileList)
        delcnt = 0
        logging.info('Session Started')
        while fileList:
            statv = os.statvfs(mediaDirPath)
            availFreeBytes = statv.f_bfree*statv.f_bsize
            if availFreeBytes >= targetFreeBytes:
                break
            filePath = fileList.pop()
            try:
                os.remove(filePath)
            except OSError as err:
                logging.error('Del Failed %s', filePath)
                logging.error('Error: %s', err)
            else:
                delcnt += 1
                logging.info('Del %s', filePath)
                logging.info('Target=%i MB  Avail=%i MB  Deleted %i of %i
Files ',
                             targetFreeBytes / MB2Bytes,
                             availFreeBytes / MB2Bytes,
                             delcnt, totFiles)
                if delcnt > totFiles / 4:  # Avoid deleting more than 1/4
of files at one time
                    logging.warning('Max Deletions Reached %i of %i',
delcnt, totFiles)
                    logging.warning('Deletions Restricted to 1/4 of total
files per session.')
                    break
        logging.info('Session Ended')
    else:
        logging.error('Directory Not Found - %s', mediaDirPath)

#---------------------------------------------------------------------------
---------------------
def freeDiskSpaceCheck(lastSpaceCheck):
    if spaceTimerHrs > 0:   # Check if disk free space timer hours is
enabled
        # See if it is time to do disk clean-up check
        if (datetime.datetime.now() - lastSpaceCheck).total_seconds() >
spaceTimerHrs * 3600:
            lastSpaceCheck = datetime.datetime.now()
            if spaceFreeMB < 100:   # set freeSpaceMB to reasonable value
if too low
                diskFreeMB = 100
            else:
                diskFreeMB = spaceFreeMB
            logging.info('spaceTimerHrs=%i  diskFreeMB=%i  spaceMediaDir=%s
spaceFileExt=%s',
                         spaceTimerHrs, diskFreeMB, spaceMediaDir,
spaceFileExt)
            freeSpaceUpTo(diskFreeMB, spaceMediaDir, spaceFileExt)
    return lastSpaceCheck

#---------------------------------------------------------------------------
---------------------
def get_image_name(path, prefix):
    # build image file names by number sequence or date/time
    rightNow = datetime.datetime.now()
    filename = ("%s/%s%04d%02d%02d-%02d%02d%02d.jpg" %
                (path, prefix, rightNow.year, rightNow.month, rightNow.day,
                 rightNow.hour, rightNow.minute, rightNow.second))
    return filename

#----------------------------------------------------------------------------
--------------------
def log_to_csv_file(data_to_append):
    log_file_path = baseDir + baseFileName + ".csv"
    if not os.path.exists(log_file_path):
        open(log_file_path, 'w').close()
        f = open(log_file_path, 'ab')
        # header_text = '"YYYYMMDD","HH","MM","Speed","Unit","    Speed
Photo Path            ","X","Y","W","H","Area","Direction"' + "\n"
        # f.write( header_text )
        f.close()
        logging.info("Create New Data Log File %s", log_file_path)
    filecontents = data_to_append + "\n"
    f = open(log_file_path, 'a+')
    f.write(filecontents)
    f.close()
    return

#----------------------------------------------------------------------------
--------------------
def speed_camera():
    ave_speed = 0.0
    # initialize variables
    frame_count = 0
    fps_time = time.time()
    first_event = True   # Start a New Motion Track
    event_timer = time.time()
    start_pos_x = 0
    end_pos_x = 0
    # setup buffer area to ensure contour is fully contained in crop area
    x_buf = int((x_right - x_left) / 10)
    y_buf = int((y_lower - y_upper) / 8)
    travel_direction = ""
    # initialize a cropped grayimage1 image
    # Only needs to be done once
    image2 = vs.read()    # Get image from PiVideoSteam thread instance
    try:
        # crop image to motion tracking area only
        image_crop = image2[y_upper:y_lower, x_left:x_right]
    except:
        vs.stop()
        logging.warn("Problem Connecting To Camera Stream.")
        logging.warn("Restarting Camera.  One Moment Please ...")
        time.sleep(4)
        return
    if verbose:
        if gui_window_on:
            logging.info("Press lower case q on OpenCV GUI Window to Quit
program")
            logging.info("        or ctrl-c in this terminal session to
Quit")
        else:
            logging.info("Press ctrl-c in this terminal session to Quit")

        if loggingToFile:
            logging.info("Sending Logging Data to %s (Console Messages
Disabled)", logFilePath)
```

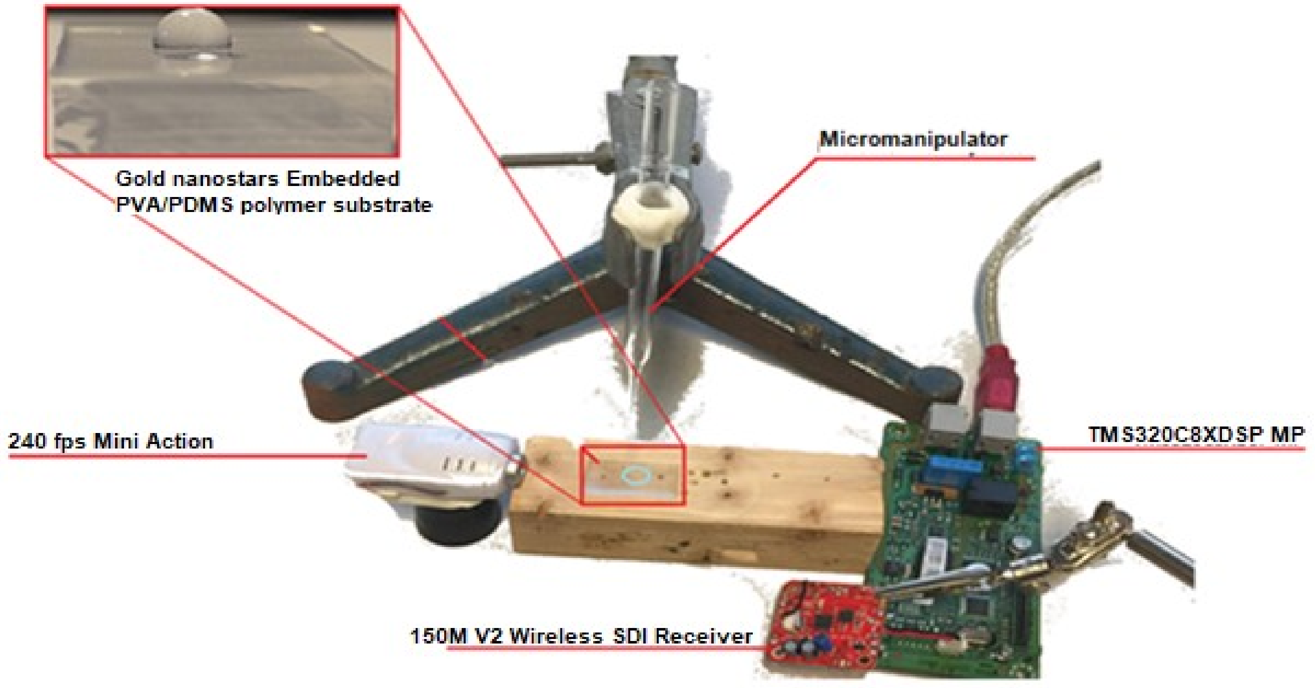
```
        else:
            logging.info("Start Logging Speed Camera Activity to Console")
    else:
        print("INFO  : NOTE: Logging Messages Disabled per verbose=%s" %
verbose)
    if pluginEnable:
        logging.info("Plugin %s is Enabled." % pluginName)
    if calibrate:
        logging.info("IMPORTANT: Camera Is In Calibration Mode ....")
    logging.info("Begin Motion Tracking .....")
    # Calculate position of text on the images
    font = cv2.FONT_HERSHEY_SIMPLEX
    if image_text_bottom:
        text_y = (image_height - 50)  # show text at bottom of image
    else:
        text_y = 10  # show text at top of image
    grayimage1 = cv2.cvtColor(image_crop, cv2.COLOR_BGR2GRAY)
    event_timer = time.time()
    # Initialize prev_image used for taking speed image photo
    prev_image = image2
    still_scanning = True
    lastSpaceCheck = datetime.datetime.now()
    speed_path = image_path
    while still_scanning:    # process camera thread images and calculate
speed
        image2 = vs.read()    # Get image from PiVideoSteam thread instance
        if WEBCAM:
            if (WEBCAM_HFLIP and WEBCAM_VFLIP):
                image2 = cv2.flip(image2, -1)
            elif WEBCAM_HFLIP:
                image2 = cv2.flip(image2, 1)
            elif WEBCAM_VFLIP:
                image2 = cv2.flip(image2, 0)
        # crop image to motion tracking area only
        image_crop = image2[y_upper:y_lower, x_left:x_right]
        if time.time() - event_timer > event_timeout:  # Check if event
timed out
            # event_timer exceeded so reset for new track
            event_timer = time.time()
            first_event = True
            start_pos_x = 0
            end_pos_x = 0
        if display_fps:   # Optionally show motion image processing loop
fps
            fps_time, frame_count = get_fps(fps_time, frame_count)
        # initialize variables
        motion_found = False
        biggest_area = MIN_AREA
        cx, cy = 0, 0   # Center of contour used for tracking
        mw, mh = 0, 0   # w,h width, height of contour
        # Convert to gray scale, which is easier
        grayimage2 = cv2.cvtColor(image_crop, cv2.COLOR_BGR2GRAY)
        # Get differences between the two greyed images
        differenceimage = cv2.absdiff(grayimage1, grayimage2)
        # Blur difference image to enhance motion vectors
        differenceimage = cv2.blur(differenceimage, (BLUR_SIZE, BLUR_SIZE))
        # Get threshold of blurred difference image based on
THRESHOLD_SENSITIVITY variable
        retval, thresholdimage = cv2.threshold(differenceimage,
THRESHOLD_SENSITIVITY,
                                               255, cv2.THRESH_BINARY)
        try:
            # opencv 2 syntax default
            contours, hierarchy = cv2.findContours(thresholdimage,
                                                   cv2.RETR_EXTERNAL,
                                                   cv2.CHAIN_APPROX_SIMPLE)
        except ValueError:
            # opencv 3 syntax
            thresholdimage, contours, hierarchy =
cv2.findContours(thresholdimage,

cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)
        total_contours = len(contours)
        # Update grayimage1 to grayimage2 ready for next image2
        grayimage1 = grayimage2
        # find contour with biggest area
        if contours:
            for c in contours:
                # get area of next contour
                found_area = cv2.contourArea(c)
                if found_area > biggest_area:
                    (x, y, w, h) = cv2.boundingRect(c)
                    # check if complete contour is completely within crop
area
                    if (x > x_buf and
                            x + w < x_right - x_left - x_buf and
                            y > y_buf and
                            y + h < y_lower - y_upper - y_buf):
                        motion_found = True
                        biggest_area = found_area
                        cx = int(x + w/2)   # put circle in middle of width
                        cy = int(y + h/2)   # put circle in middle of
height
                        mw = w
                        mh = h
            if motion_found:
                # Process motion event and track data
                if first_event:   # This is a first valide motion event
                    first_event = False
                    start_pos_x = cx
                    end_pos_x = cx
                    track_start_time = time.time()
                    logging.info("New  - cxy(%i,%i) Start New Track", cx,
cy)
                else:
                    if end_pos_x - start_pos_x > 0:
                        travel_direction = "L2R"
                    else:
                        travel_direction = "R2L"
                    if (abs(cx - end_pos_x) > x_diff_min and abs(cx -
end_pos_x) < x_diff_max):
                        # movement is within acceptable distance range of
last event
                        end_pos_x = cx
                        tot_track_dist = abs(end_pos_x - start_pos_x)
                        tot_track_time = abs(time.time() -
track_start_time)
                        ave_speed = float((abs(tot_track_dist /
tot_track_time)) * speed_conv)
                        if abs(end_pos_x - start_pos_x) >= track_len_trig:
                            # Track length exceeded so take process speed
photo
                            if ave_speed > max_speed_over or calibrate:
                                logging.info(" Add - cxy(%i,%i) %3.2f %s
px=%i/%i"
                                             " C=%i %ix%i=%i sqpx %s",
                                             cx, cy, ave_speed,
speed_units,
                                             abs(start_pos_x - end_pos_x),
track_len_trig,
                                             total_contours, mw, mh,
biggest_area, travel_direction)
                                # Resized and process prev image before
saving to disk
                                prev_image = image2
                                if calibrate:       # Create a calibration
image
                                    filename = get_image_name(speed_path,
"calib-")
                                    prev_image =
take_calibration_image(filename, prev_image)
                                else:
                                    # Check if subdirectories configured
and create as required
                                    speed_path =
subDirChecks(imageSubDirMaxHours,

imageSubDirMaxFiles,
                                                                 image_path,
image_prefix)
                                    if image_filename_speed:
                                        speed_prefix =
str(int(round(ave_speed))) + "-" + image_prefix
                                    else:
                                        speed_prefix = image_prefix
                                    filename = get_image_name(speed_path,
speed_prefix)
                                if spaceTimerHrs > 0:
                                # if required check free disk space and
delete older files (jpg)
                                    lastSpaceCheck =
freeDiskSpaceCheck(lastSpaceCheck)
                                if image_max_files > 0:
                                # Manage a maximum number of files and
delete oldest if required.
                                    deleteOldFiles(image_max_files,
speed_path, image_prefix)
                                # Add motion rectangle to image
                                if image_show_motion_area:
                                    if SHOW_CIRCLE:
                                        cv2.circle(prev_image, (cx +
x_left, cy + y_upper),
                                                   CIRCLE_SIZE, cvRed,
LINE_THICKNESS)
                                    cv2.line(prev_image, (x_left, y_upper),
                                             (x_right, y_upper), cvRed, 1)
                                    cv2.line(prev_image, (x_left, y_lower),
                                             (x_right, y_lower), cvRed, 1)
                                    cv2.line(prev_image, (x_left, y_upper),
                                             (x_left, y_lower), cvRed, 1)
                                    cv2.line(prev_image, (x_right,
y_upper),
                                             (x_right, y_lower), cvRed, 1)
                                big_image = cv2.resize(prev_image,
(image_width, image_height))
                                if image_text_on:
                                    # Write text on image before saving
                                    image_text = ("SPEED %.1f %s - %s"
                                                  % (ave_speed,
speed_units, filename))
                                    text_x = int((image_width / 2) -
                                                 (len(image_text) *
image_font_size / 3))
                                    if text_x < 2:
                                        text_x = 2
                                    logging.info(" Ave Speed is %.1f %s %s
",
                                                 ave_speed, speed_units,
travel_direction)
                                    cv2.putText(big_image, image_text,
(text_x, text_y),
                                                font, FONT_SCALE,
(cvWhite), 2)
                                logging.info(" Saved %s", filename)
                                cv2.imwrite(filename, big_image)
                                if imageRecentMax > 0 and not calibrate:
                                    # Optional save most recent files to a
recent folder
                                    saveRecent(imageRecentMax,
imageRecentDir, filename,
                                               image_prefix)
                                if log_data_to_CSV:    # Format and Save
Data to CSV Log File
                                    log_time = datetime.datetime.now()
                                    log_csv_time =
("%s%04d%02d%02d%s,%s%02d%s,%s%02d%s"
                                                    % (quote,
                                                       log_time.year,
log_time.month,
                                                       log_time.day,
                                                       quote,
                                                       quote,
log_time.hour, quote,
                                                       quote,
log_time.minute, quote))
                                    log_csv_text =
("%s,%.2f,%s%s%s,%s%s%s,%i,%i,%i,%i,%i,%s%s%s"
                                                    % (log_csv_time,
ave_speed,
                                                       quote, speed_units,
                                                       quote, quote,
filename,
                                                       quote, cx, cy, mw,
mh, mw * mh,
                                                       quote,
travel_direction, quote))
                                    log_to_csv_file(log_csv_text)
                                logging.info("End  - Tracked %i px in %.3f
sec",
                                             tot_track_dist,
tot_track_time)
```

```
                                time.sleep(track_timeout)   # Optional Wait
to avoid dual tracking.
                            else:
                                logging.info("End  - Skip Photo SPEED %.1f
%s"
                                             " max_speed_over=%i  %i px in
%.3f sec"
                                             " C=%i A=%i sqpx",
                                             ave_speed, speed_units,
                                             max_speed_over,
tot_track_dist,
                                             tot_track_time,
total_contours, biggest_area)
                                time.sleep(track_timeout)   # Optional Wait
to avoid dual tracking
                            # Track Ended so Reset Variables for next cycle
through loop
                            start_pos_x = 0
                            end_pos_x = 0
                            first_event = True
                        else:
                            logging.info(" Add - cxy(%i,%i) %3.1f %s"
                                         " px=%i/%i C=%i %ix%i=%i sqpx %s",
                                         cx, cy, ave_speed, speed_units,
                                         abs(start_pos_x - end_pos_x),
                                         track_len_trig, total_contours,
                                         mw, mh, biggest_area,
travel_direction)
                            end_pos_x = cx
                        event_timer = time.time()  # Reset event_timer
since valid motion was found
                    else:
                        if show_out_range:
                            if abs(cx - end_pos_x) >= x_diff_max:
                                # Ignore movements that exceed Max px
movement allowed
                                logging.info(" Out - cxy(%i,%i) Dist=%i is
>=%i px"
                                             " C=%i %ix%i=%i sqpx %s",
                                             cx, cy, abs(cx - end_pos_x),
x_diff_max,
                                             total_contours, mw, mh,
biggest_area, travel_direction)
                            else:
                                logging.info(" Out - cxy(%i,%i) Dist=%i is
<=%i px"
                                             " C=%i %ix%i=%i sqpx %s",
                                             cx, cy, abs(cx - end_pos_x),
x_diff_min,
                                             total_contours, mw, mh,
biggest_area, travel_direction)
                                event_timer = time.time()  # Reset
event_timer since valid motion was found
                if gui_window_on:
                    # show small circle at motion location
                    if SHOW_CIRCLE:
                        cv2.circle(image2, (cx + x_left * WINDOW_BIGGER,
                                            cy + y_upper * WINDOW_BIGGER),
                                   CIRCLE_SIZE, cvGreen, LINE_THICKNESS)
                    else:
                        cv2.rectangle(image2, (int(cx + x_left - mw/2),
                                         int(cy + y_upper - mh/2)),
                                    (int(cx + x_left + mw/2),
                                     int(cy + y_upper + mh/2)),
                                    cvGreen, LINE_THICKNESS)
        if gui_window_on:
            # cv2.imshow('Difference Image',difference image)
            cv2.line(image2, (x_left, y_upper), (x_right, y_upper), cvRed,
1)
            cv2.line(image2, (x_left, y_lower), (x_right, y_lower), cvRed,
1)
            cv2.line(image2, (x_left, y_upper), (x_left, y_lower), cvRed,
1)
            cv2.line(image2, (x_right, y_upper), (x_right, y_lower), cvRed,
1)
            image_view = cv2.resize(image2, (image_width, image_height))
            cv2.imshow('Movement (q Quits)', image_view)
            if show_thresh_on:
                cv2.imshow('Threshold', thresholdimage)
            if show_crop_on:
                cv2.imshow('Crop Area', image_crop)
            # Close Window if q pressed
            if cv2.waitKey(1) & 0xFF == ord('q'):
                cv2.destroyAllWindows()
                logging.info("End Motion Tracking ......")
                vs.stop()
                still_scanning = False

#---------------------------------------------------------------------------
---------------------
if __name__ == '__main__':

    show_settings()
    try:
        WEBCAM_TRIES = 0
        while True:
            # Save images to an in-program stream
            # Setup video stream on a processor Thread for faster speed
            if WEBCAM:   #  Start Web Cam stream (Note USB webcam must be
plugged in)
                WEBCAM_TRIES += 1
                logging.info("Initializing USB Web Camera Try .. %i",
WEBCAM_TRIES)
                vs = WebcamVideoStream().start()
                vs.CAM_SRC = WEBCAM_SRC
                vs.CAM_WIDTH = WEBCAM_WIDTH
                vs.CAM_HEIGHT = WEBCAM_HEIGHT
                if WEBCAM_TRIES > 3:
                    logging.error("USB Web Cam Not Connecting to WEBCAM_SRC
%i", WEBCAM_SRC)
                    logging.error("Check Camera is Plugged In and Working
on Specified SRC")
                    logging.error("        and Not Used(busy) by Another
Process.")
                    logging.error("Exiting %s", progName)
                    sys.exit(1)
                time.sleep(4.0)  # Allow WebCam to initialize
            else:
                logging.info("Initializing Pi Camera ....")
                vs = PiVideoStream().start()
                vs.camera.rotation = CAMERA_ROTATION
                vs.camera.hflip = CAMERA_HFLIP
                vs.camera.vflip = CAMERA_VFLIP
                time.sleep(2.0)  # Allow PiCamera to initialize
            speed_camera()
    except KeyboardInterrupt:
        vs.stop()
        print("")
        logging.info("User Pressed Keyboard ctrl-c")
        logging.info("Exiting %s %s", progName, version)
        sys.exit()
```

APPENDIX C

Low cost Droplet Analysis Experimental Setup


Droplet Test Setup
Light Source
Droplet
240fps Action Cam
150M V2 Wireless Receiver
TMS320C8 MP
Control PC


Gold nanostars Embedded PVA/PDMS polymer substrate
Micromanipulator
240 fps Mini Action
TMS320C8XDSP MP
150M V2 Wireless SDI Receiver

Code:

```
import wx
import os
import string
import popen2
import glob
import Image
import wx.lib.plot as plot
from mydrop import *

VERSION = 0.4

class MyFrame(wx.Frame):
    """
    This is MyFrame.  It just shows a few controls on a wxPanel,
    and has a simple menu.
    """
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, -1, title,
                          pos=(150, 150), size=(525, 350))

        # Create the menubar
        menuBar = wx.MenuBar()
        # and a menu
        menu = wx.Menu()
        # add an item to the menu, using \tKeyName automatically
        # creates an accelerator, the third param is some help text
        # that will show up in the statusbar
        menu.Append(103, "Change &Directory\tAlt-D", "Change Directory")
        menu.Append(102, "Run &TraceDrop\tAlt-T", "Run TraceDrop")
        menu.Append(101, "E&xit\tAlt-X", "Exit")
        # bind the menu event to an event handler
        self.Bind(wx.EVT_MENU, self.OnTimeToClose, id=101)
        self.Bind(wx.EVT_MENU, self.OnTraceDrop, id=102)
        self.Bind(wx.EVT_MENU, self.OnCdir, id=103)
        # and put the menu on the menubar
        menuBar.Append(menu, "&File")

        menu2 = wx.Menu()
        menu2.Append(201, "&Help\tAlt-H", "Help")
        menu2.Append(202, "&About\tAlt-A", "About xtracedrop")

        self.Bind(wx.EVT_MENU, self.OnHelp, id=201)
        self.Bind(wx.EVT_MENU, self.OnMsg, id=202)

        menuBar.Append(menu2, "Help")
        self.SetMenuBar(menuBar)
        self.CreateStatusBar()

        # Now create the Panel to put the other controls on.
        panel = wx.Panel(self)

        # Add a few control fields
        textin = wx.StaticText(panel, -1, "infile              conc     a/b
resid ")
        self.input = ''
        fieldin = wx.TextCtrl(panel, 500, self.input, size=(300, 150),
style=wx.TE_MULTILINE)
        fieldin.Bind(wx.EVT_TEXT, self.GetData)
        fieldin.SetToolTipString("Edit filenames of drop images\n"
                                "Enter detergent concentrations next to
file names\n"
                                "Results will be placed next to filename\n"
                                "Delete or add lines as needed")
        self.fieldin = fieldin

        self.filetemplate = '*.jpg'
        templatetext = wx.StaticText(panel, -1, 'filename template')
        templatefield = wx.TextCtrl(panel, 510, self.filetemplate,
size=(125, -1))
        templatefield.Bind(wx.EVT_TEXT, self.GetData)
        templatefield.Bind(wx.EVT_KILL_FOCUS, self.FindFiles)
        self.templatefield = templatefield

        self.dropctr = '2150 1300'
        textctr = wx.StaticText(panel, -1, "center:")    # drop center
        fieldctr = wx.TextCtrl(panel, 501, self.dropctr, size=(125, -1))
        fieldctr.Bind(wx.EVT_TEXT, self.GetData)
        fieldctr.SetToolTipString("Approx Center of the drop in all the
images")
        self.fieldctr = fieldctr

        self.outfile = 'results.dat'
        textout = wx.StaticText(panel, -1, "results file: ")  # results
file
        fieldout = wx.TextCtrl(panel, 502, self.outfile, size=(125,-1))
        fieldout.Bind(wx.EVT_TEXT, self.GetData)
        fieldout.SetToolTipString("Enter filename for output data")
        self.fieldout = fieldout



        tracebtn = wx.Button(panel, -1, "Run", style=wx.BU_EXACTFIT)
#control button2
        plotbtn = wx.Button(panel, -1, "Plot Results",
style=wx.BU_EXACTFIT)
        # bind the button events to handlers
        tracebtn.Bind(wx.EVT_BUTTON, self.OnTraceDrop)
        plotbtn.Bind(wx.EVT_BUTTON, self.OnPlotResults)
        plotbtn.Bind(wx.EVT_RIGHT_DOWN, self.OnPlotResultsRight)
        plotbtn.SetToolTipString("Plot ratio vs. conc\n"
                                "Right click to save plot")

        choicelist = []
        for i in range(50): choicelist.append("%s" % i)
        choicefield = wx.Choice(panel, 520, size=(50,30),
choices=choicelist)
        choicefield.Bind(wx.EVT_CHOICE, self.GetChoice)
        choicefield.SetToolTipString("select image for plotting")
        self.choicefield = choicefield
        self.choicefield.SetSelection(0)
        self.plotchoice = 0

        plotfitbtn = wx.Button(panel, -1, "Show Fit", style=wx.BU_EXACTFIT)
        plotfitbtn.Bind(wx.EVT_BUTTON, self.OnPlotFit)
        plotfitbtn.Bind(wx.EVT_RIGHT_DOWN, self.OnPlotFitRight)  #right
click->print
        plotfitbtn.SetToolTipString("Show fit for selected image\n"
                                    "Right click will save plot to file")
        showimagebtn = wx.Button(panel, -1, "Show Img",
style=wx.BU_EXACTFIT)
```

```
        showimagebtn.Bind(wx.EVT_BUTTON, self.OnShowImg)
        showimagebtn.SetToolTipString("View original for selected image")

        closebtn = wx.Button(panel, -1, "Close", style=wx.BU_EXACTFIT)   #
control button1
         closebtn.Bind(wx.EVT_BUTTON, self.OnTimeToClose)
         closebtn.SetToolTipString("Close Program")



#--------------
# Use a sizer to layout the controls, stacked vertically and with
        # a 10 pixel border around each
        sizer = wx.BoxSizer(wx.VERTICAL)

        sizera = wx.BoxSizer(wx.VERTICAL)
        sizera.Add(textin, 0, wx.ALL, 5)
        sizera.Add(fieldin, 0, wx.ALL, 5)

        sizerb = wx.BoxSizer(wx.VERTICAL)
        sizerb.Add(templatetext, 0, wx.LEFT,15)
        sizerb.Add(templatefield, 0, wx.ALL, 5)
        sizerb.Add(textctr,0,wx.LEFT,30)
        sizerb.Add(fieldctr,0,wx.ALL, 5)
        sizerb.Add((20,20))
        sizerb.Add(textout, 0, wx.LEFT, 20)
        sizerb.Add(fieldout, 0, wx.ALL, 5)

        sizerc = wx.BoxSizer(wx.HORIZONTAL)
        sizerc.Add(sizera, 0, wx.ALL, 10)
        sizerc.Add(sizerb, 0, wx.ALL, 10)

        sizer2 = wx.BoxSizer(wx.HORIZONTAL)

# place buttons horizontally
        sizer2.Add(tracebtn, 0, wx.LEFT | wx.RIGHT, 10)
        sizer2.Add(plotbtn, 0, wx.LEFT | wx.RIGHT, 5)
        sizer2.Add(choicefield, 0, wx.LEFT, 20)
        sizer2.Add(plotfitbtn, 0, wx.LEFT , 5)
        sizer2.Add(showimagebtn, 0, wx.LEFT, 5)
        sizer2.Add(closebtn, 0, wx.LEFT, 25)

        sizer.Add((20,10))
        sizer.Add(sizerc, 0, wx.ALL, 10)
        sizer.Add(sizer2, 0, wx.ALL, 10)

        panel.SetSizer(sizer)
        panel.Layout()
        self.plotdata = []


#initialize results # look for files in this directory for processing

    def FindFiles(self,event):
        self.fieldin.Clear()
        imgfiles = glob.glob('%s' % self.filetemplate)
        imgfiles.sort()
        if len(imgfiles) == 0:
            cwd = os.getcwd()
            print '%s: No Files Found' % cwd
        else:
            for file in imgfiles:
                self.fieldin.AppendText('%s\n' % file)
        return True


    def GetData(self, event):
        data = event.GetString()
        if event.GetId() == 500:
            self.input = data
        if event.GetId() == 501:
            self.dropctr = data
        if event.GetId() == 502:
            self.outfile = data
        if event.GetId() == 510:
            self.filetemplate = data

    def GetChoice(self, event):
        if event.GetId() == 520: self.plotchoice = event.GetSelection()

    def OnShowImg(self, event):
        try:
            if self.plotchoice >= len(self.drop):
                print 'Selected Img out of range'
                return
            else:
                img = Image.open(self.drop[self.plotchoice].imgfile)
                img.show()
        except:
            dropdata = string.rstrip(self.input)
            droplist = string.split(dropdata,'\n')
            numfiles = len(droplist)
            if self.plotchoice >= numfiles:
                print 'Selected Img out of range'
                return
            else:
                filename = string.split(droplist[self.plotchoice])[0]
                img = Image.open(filename)
                img.show()


    def OnTimeToClose(self, evt):
        """Event handler for the button click."""
        self.Close()

    def OnTraceDrop(self, evt):
        dropdata = string.rstrip(self.input)
        droplist = string.split(dropdata,'\n')
        numfiles = len(droplist)
        self.concmin = 10000.
        self.concmax = 0.
        self.ratiomin = 100000.
        self.ratiomax = 0.

        # run tracedrop
        self.fieldin.Clear()
        self.results = []     # list of tuples containing results
        self.drop = []       # list of class instances
        index = 0
        for drop in droplist:  # analyze each drop
            if ':' in drop[0]: drop = drop[1:]  #strip first element
            filename = string.split(drop)[0]
            try:
                conc = float(string.split(drop)[1])
            except:
                conc = 0.0     #default value if not entered into gui
            nextdrop = MyDrop(filename, self.dropctr)
            nextdrop.setconc(conc)
            nextdrop.tracedrop()   #run algorithm to trace drop and fit to
ellipse
            self.drop.append(nextdrop)
            self.results.append((nextdrop.a1,nextdrop.b1,
                                nextdrop.x1,nextdrop.y1,nextdrop.error))
            self.fieldin.AppendText('%s: %s  %.2f  %.2f  %.2f\n' % (index,
nextdrop.imgfile,
                                        nextdrop.conc, nextdrop.ratio,
nextdrop.error))
            index += 1

        # write file with results from tracedrop
        output_txt = open(self.outfile,'w')   #write out results
        self.plotdata = []
        index = 0
        for drop in self.drop:
            output_txt.write( '%s\t%.2f\t%.2f\t%.2f\n'%  (drop.imgfile,
drop.conc,
                                        drop.ratio,drop.error))
            self.plotdata.append((drop.conc,drop.ratio))
            if drop.conc > self.concmax: self.concmax = drop.conc
            if drop.conc < self.concmin: self.concmin = drop.conc
            if drop.ratio > self.ratiomax: self.ratiomax = drop.ratio
            if drop.ratio < self.ratiomin: self.ratiomin = drop.ratio
            index += 1
        output_txt.close()
        self.concmax *= 1.1
        self.ratiomax *= 1.1
        self.concmin *= 0.9
        self.ratiomin *= 0.9
        print "**** All Done *****"

    def OnPlotResults(self, event):
        if len(self.plotdata) == 0:
            print 'No results to plot'
            return False
        title = '%s' % self.outfile
        self.sumplot = MyPlot(self.plotdata, title)

    def OnPlotResultsRight(self, event):
        self.OnPlotResults(event)
        #self.sumplot.myplot.SaveFile('test.png')  #this doesn't work!???
        self.sumplot.myplot.SaveFile()

    def OnPlotFit(self, event):
        index = self.plotchoice
        if (index+1) >= len(self.drop):
            print 'Selected file out of range'
            return
        filebase = string.split(self.drop[index].imgfile,'.')[0]
        #prepare data used for fit
        self.fitdata = []
        i = 0
        for x in self.drop[index].x_init:
            self.fitdata.append((self.drop[index].x_init[i],
                                self.drop[index].y_init[i]))
            i += 1
        self.xmax = max(self.drop[index].x_init)
        self.xmin = min(self.drop[index].x_init)
        self.ymax = max(self.drop[index].y_init)
        self.ymin = min(self.drop[index].y_init)
        #prepare data from edge of drop (excluded from fit)
        self.edgedata = []
        i = 0
        for x in self.drop[index].x_edge:
            self.edgedata.append((self.drop[index].x_edge[i],
                                self.drop[index].y_edge[i]))
            i += 1
        if max(self.drop[index].x_edge) > self.xmax:
            self.xmax = max(self.drop[index].x_edge)
        if min(self.drop[index].x_edge) < self.xmin:
            self.xmin = min(self.drop[index].x_edge)
        if max(self.drop[index].y_edge) > self.ymax:
            self.ymax = max(self.drop[index].y_edge)
        if min(self.drop[index].y_edge) < self.ymin:
            self.ymin = min(self.drop[index].y_edge)

        self.xmax *= 1.1
        self.ymax *= 1.1
        self.xmin *= 0.9
        self.ymin *= 0.9

        # create data for fitted ellipse
        p1 = [self.drop[index].a1,self.drop[index].b1,
                         self.drop[index].x1,self.drop[index].y1]
        self.calcdata = []
        xst = int(self.drop[index].x1-self.drop[index].a1 + 1)
        xend = int(self.drop[index].x1+self.drop[index].a1 - 1)
        for xcalc in range(xst,xend):
            ycalc=self.drop[index].y1-sqrt(ellipse(xcalc,p1))
            self.calcdata.append((xcalc,ycalc))

        title = '   a/b= %.2f  error=%.2f' % (self.drop[index].ratio,
                                            self.drop[index].error)
        #pflag = (event.GetEventType() == 10029)   # event for right button
        self.nextplot = PlotFit(self.drop[index].imgfile, title)  # creates
plot

        self.plotchoice += 1     #increment counter in gui
        self.choicefield.SetSelection(self.plotchoice)

    def OnPlotFitRight(self, event):
        self.OnPlotFit(event)
        self.nextplot.plotfit.SaveFile()
        #self.nextplot.plotfit.SaveFile('plotfit.png')  #this is supposed
to work

    def OnMsg(self, event):
        d = wx.MessageDialog (self, "xtracedrop ver. %s\n"
                    "This program calculates shape of a series of drops."
                    "find help at
http://www.nysbc.net/twiki/bin/view/Main/tracedrop\n"
                    "send bug reports to stokes@nyu.edu, rice@nysbc.org"
                    % VERSION,
                    "About xtracedrop", wx.OK)
        d.ShowModal()
        d.Destroy()

    def OnHelp(self, event):
        d = wx.MessageDialog (self, "find help at\n"
        "http://www.nysbc.net/twiki/bin/view/Main/EmIP",
                                "Help for EMIP", wx.OK)
        d.ShowModal()
        d.Destroy()

    def OnCdir(self, event):
        frame=InfoWindow(None, 'New Directory')
        frame.Show(True)
        return True



def SpawnProcess(program, input='', output=False, redirect = ''):
    print '************************\n', program
    exe = string.split(program)[0]  #isolate program name
    t = popen2.Popen3('which %s' % exe, True)
    if t.wait():  #wait for process to finish and get exit status
        print '***ERROR: program %s not in path' % exe
        return False
    p = popen2.Popen4(program)                #spawn process
    if input: p.tochild.write(input)                   #write stdinput to
program
    p.tochild.close()
    if output or redirect != '':
        stdout = "".join(p.fromchild.readlines())   #get stdoutput from
program
        p.fromchild.close()
        if redirect:
            f = open(redirect, 'w')
            f.write(stdout)
            f.close()
        else:
            print stdout
    status = p.poll()  #0=normal end, -1=still running, >0 indicates error
    print
'\n************************\nDone!\n************************\n'
    return True

class MyPlot(wx.Frame):
    def __init__(self, data, title):
        self.frame1 = wx.Frame(None, title="tracedrop results", id=-1,
size=(410, 340))
        self.panel1 = wx.Panel(self.frame1)
        self.panel1.SetBackgroundColour("yellow")

        # mild difference between wxPython26 and wxPython28
        if wx.VERSION[1] < 7:
            self.myplot = plot.PlotCanvas(self.panel1, size=(400, 300))
        else:
            self.myplot = plot.PlotCanvas(self.panel1)
            self.myplot.SetInitialSize(size=(400, 300))
        # enable the zoom feature (drag a box around area of interest)
        self.myplot.SetEnableZoom(True)

        # list of (x,y) data point tuples
        #data = [(1,2), (2,3), (3,5), (4,6), (5,8), (6,8), (12,10), (13,4)]
        # draw points as a line
        line = plot.PolyLine(data, colour='red', width=1)
        # also draw markers, default colour is black and size is 2
        # other shapes 'circle', 'cross', 'square', 'dot', 'plus'
        marker = plot.PolyMarker(data, marker='triangle')
        # set up text, axis and draw
        gc = plot.PlotGraphics([line, marker], title, 'conc', 'a/b')
        self.myplot.Draw(gc, xAxis=(app.frame.concmin,app.frame.concmax),
                     yAxis=(app.frame.ratiomin,app.frame.ratiomax))

        self.frame1.Show(True)

class PlotFit(wx.Frame):
    def __init__(self, filename, title):
        self.frame1 = wx.Frame(None, title="tracedrop fit", id=-1,
size=(600, 630))
        self.panel1 = wx.Panel(self.frame1)
        self.panel1.SetBackgroundColour("yellow")

        # mild difference between wxPython26 and wxPython28
        if wx.VERSION[1] < 7:
            plotter = plot.PlotCanvas(self.panel1, size=(600, 630))
        else:
            self.plotfit = plot.PlotCanvas(self.panel1)
            self.plotfit.SetInitialSize(size=(600, 600))
        # enable the zoom feature (drag a box around area of interest)
        self.plotfit.SetEnableZoom(True)

        # list of (x,y) data point tuples
        #data = [(1,2), (2,3), (3,5), (4,6), (5,8), (6,8), (12,10), (13,4)]
        # draw points as a line
        line = plot.PolyLine(app.frame.calcdata, colour='blue', width=2)
        # also draw markers, default colour is black and size is 2
        # other shapes 'circle', 'cross', 'square', 'dot', 'plus'
        marker1 = plot.PolyMarker(app.frame.fitdata, colour='green',
marker='cross')
        marker2 = plot.PolyMarker(app.frame.edgedata, colour='red',
marker='plus')
        # set up text, axis and draw
        gc = plot.PlotGraphics([marker1, marker2, line], filename+title,
'x', 'y')
        (xmax, xmin) = (app.frame.xmax, app.frame.xmin)
        (ymax, ymin) = (app.frame.ymax, app.frame.ymin)
        xrange = xmax - xmin
        yrange = ymax - ymin
        range = max(xrange, yrange)  # ensure that plot has is isotropic
        range2 = range/2.
        xmean = (xmax + xmin)/2.
        ymean = (ymax + ymin)/2.
        self.plotfit.Draw(gc, xAxis=(xmean-range2, xmean+range2),
                     yAxis=(ymean-range2, ymean+range2))

        self.frame1.Show(True)

class InfoWindow(wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, -1, title, pos=(50,150),
size=(350,200))

        panel = wx.Panel(self)

        currentdir = wx.StaticText(panel, -1,
            "enter working directory:\n(? will create list of possible
subdirectories)")
        directory = wx.TextCtrl(panel, -1, os.getcwd(), size=(300,-1))

        dirbtn = wx.Button(panel, 1000, "Change Dir")
        exitbtn = wx.Button(panel, 2000, "Close")

        directory.Bind(wx.EVT_TEXT, self.GetDir)
        directory.SetFocus()

        dirbtn.Bind(wx.EVT_BUTTON, self.OnChDir)
        exitbtn.Bind(wx.EVT_BUTTON, self.SubExit)

        self.newdir = ""

        sizerbtn = wx.BoxSizer(wx.HORIZONTAL)
        sizerbtn.Add(dirbtn, 0, wx.LEFT|wx.RIGHT, 10)
        sizerbtn.Add(exitbtn, 0, wx.LEFT|wx.RIGHT, 10)

        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(currentdir, 0, wx.ALL, 10)
        sizer.Add(directory, 0, wx.ALL, 10)
        sizer.Add(sizerbtn, 0, wx.ALL, 10)

        panel.SetSizer(sizer)
        panel.Layout()


    def GetDir(self,event):
        self.newdir = '%s' % event.GetString()

    def OnChDir(self,event):
        if os.path.exists(self.newdir):
            os.chdir(self.newdir)
            app.frame.FindFiles(wx.EVT_TEXT)
        else:
            print ("no such directory: %s" % self.newdir)
            root = os.path.split(self.newdir)[0]
            while not os.path.exists(root):
                root = os.path.split(root)[0]
            print("valid subdirectories of\n%s:" % root)
            #os.system('ls -F %s | grep /' % root)
            filelist = os.listdir(root)
            for file in filelist:
                if os.path.isdir(file): print file
            return

    def SubExit(self,event):
        self.Close(True)


class MyApp(wx.App):
    def OnInit(self):
        self.frame = MyFrame(None, "xtracedrop")
        self.SetTopWindow(self.frame)

        #print "Print statements go to this stdout window by default."

        self.frame.Show(True)
        return True
```

```
if __name__ == '__main__':
      app = MyApp(redirect=True)
      app.MainLoop()
```