UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE FÍSICA

Robotic-Assisted Approaches for Image-Controlled Ultrasound Procedures

Guilherme Alexandre da Costa Correia

Mestrado Integrado em Engenharia Biomédica e Biofísica
Perfil em Engenharia Clínica e Instrumentação Médica

Dissertação orientada por:

Professor Rui Pedro Duarte Cortesão
Professora Guiomar Gaspar de Andrade Evans

2019

ACKNOWLEDGEMENTS

I would like to start by thanking Prof. Rui Cortesão for giving me the opportunity to join his research group at the Institute of Systems and Robotics of the University of Coimbra, and by all his support and guidance throughout the year. I would also like to thank Prof. Guiomar Evans for the patience, the orientation and for all the scientific advice.

I would like to thank my parents and my brother, for the unconditional support and motivation. A special thanks goes also to all my aunts and uncles, cousins, grandmother and grandfathers for all the fun moments, all the hugs, and all the encouragement. Thank you all for being always there for me.

My friends are a second family for me. To all of them: Cláudia, Beatriz, Nuno, Maria, Rita, João and Raquel, my heartfelt thanks for all the companionship, patience and support over these last five years.

ABSTRACT

Ultrasound (US) systems are very popular in the medical field for several reasons. Compared to other imaging techniques such as CT or MRI, the combination of low-priced and portable hardware with real-time image acquisition enables great flexibility regarding medical applications, from simple diagnostics tasks to high precision ones, including those with robotic assistance. Unlike other techniques, the image quality and procedure accuracy are highly dependent on user skills for spatial ultrasound probe positioning and orientation around a region of interest (ROI) for inspection. To make diagnostics less prone to error and guided procedures more precise, and consequently safer, the US approach can be coupled to a robotic system. The probe acts as a camera to the patient body and relevant imaging information can be used to control a robotic arm, enabling the creation of semi-autonomous, cooperative and possibly fully autonomous diagnostics and therapeutics.

In this project our aim is to develop a semi-autonomous tool for tracking defined structures of interest within US images, that outputs meaningful spatial information of a target structure (location of the centre of mass [CM], main orientation and elongation). Such tool must accomplish real-time requirements for future use in autonomous image-guided robotic systems. To this end, the concepts of moment-based visual servoing and active contours are fundamental. Active contours possess an underlying physical model allowing deformation according to image information, such as edges, image regions and specific image features. Additionally, the mathematical framework of vision-based control enables us to establish the types of necessary information for controlling a future autonomous system and how such information can be transformed to specify a desired task.

Once implemented in MATLAB the tracking and temporal performance of this approach is tested in built agar-agar phantoms embedded with water-filled balloons, for stability demonstration, probe motion robustness in translational and rotational movements, as well as promising capability in responding to target structure deformations. The developed framework is also inside the expected levels, being compatible with a 25 frames per second image acquisition setup. The framework also has a standalone tool capable of dealing with 50 fps. Thus, this work lays the foundation for US guided procedures compatible with real-time approaches in moving and deforming targets.

Keywords:

Ultrasound imaging, image-guided procedures, visual servoing, active contours, image moments

RESUMO

A aquisição de imagens de ultrassons (US) é atualmente uma das modalidades de aquisição de imagem mais implementadas no meio médico por diversas razões. Quando comparada a outras modalidades como a tomografia computorizada (CT) e ressonância magnética (MRI), a combinação da sua portabilidade e baixo custo com a possibilidade de adquirir imagens em tempo real resulta numa enorme flexibilidade no que diz respeito às suas aplicações em medicina. Estas aplicações estendem-se desde o simples diagnóstico em ginecologia e obstetrícia, até tarefas que requerem alta precisão como cirurgia guiada por imagem ou mesmo em oncologia na área da braquiterapia. No entanto ao contrário das suas contrapartes devido à natureza do princípio físico da qual decorrem as imagens, a sua qualidade de imagem é altamente dependente da destreza do utilizador para colocar e orientar a sonda de US na região de interesse (ROI) correta, bem como, na sua capacidade de interpretar as imagens obtidas e localizar espacialmente as estruturas no corpo do paciente.

De modo para tornar os procedimentos de diagnóstico menos propensos a erros, bem como os procedimentos guiados por imagem mais precisos, o acoplamento desta modalidade de imagem com uma abordagem robótica com controlo baseado na imagem adquirida é cada vez mais comum. Isto permite criar sistemas de diagnóstico e terapia semiautónomos, completamente autónomos ou cooperativos com o seu utilizador. Esta é uma tarefa que requer conhecimento e recursos de múltiplas áreas de conhecimento, incluindo de visão por computador, processamento de imagem e teoria de controlo.

Em abordagens deste tipo a sonda de US vai agir como câmara para o interior do corpo do paciente e o processo de controlo vai basear-se em parâmetros tais como, as informações espaciais de uma certa estrutura-alvo presente na imagem adquirida. Estas informações que são extraídos através de vários estágios de processamento de imagem são utilizadas como realimentação no ciclo de controlo do sistema robótico em questão. A extração de informação espacial e controlo devem ser o mais autónomos e céleres possível, de modo a conseguir produzir-se um sistema com a capacidade de atuar em situações que requerem resposta em tempo real.

Assim, o objetivo deste projeto foi desenvolver, implementar e validar, em MATLAB, as bases de uma abordagem para o controlo semiautónomo baseado em imagens de um sistema robótico de US e que possibilite o rastreio de estruturas-alvo e a automação de procedimentos de diagnóstico gerais com esta modalidade de imagem. De modo a atingir este objetivo foi assim implementada nesta plataforma, um programa semiautónomo com a capacidade de rastrear contornos em imagens US e capaz de produzir informação relativamente à sua posição e orientação na imagem. Este programa foi desenhado para ser compatível com uma abordagem em tempo real utilizando um sistema de aquisição *SONOSITE TITAN*, cuja velocidade de aquisição de imagem é de 25 fps. Este programa depende de fortemente de conceitos integrados na área de visão por computador, como computação de momentos e contornos ativos, sendo este último o motor principal da ferramenta de rastreamento.

De um modo geral este programa pode ser descrito como uma implementação para rastreamento de contornos baseada em contornos ativos. Este tipo de contornos beneficia de um modelo físico subjacente que o permite ser atraído e convergir para determinadas características da imagem, como linhas, fronteiras, cantos ou regiões específicas, decorrente da minimização de um funcional de energia definido para a sua fronteira. De modo a simplificar e tornar mais célere a sua implementação este modelo dinâmico recorreu à parametrização dos contornos com funções harmónicas, pelo que as suas variáveis de sistema são descritoras de Fourier. Ao basear-se no princípio de menor energia o sistema pode ser encaixado na formulação da mecânica de Euler-Lagrange para sistemas físicos e a partir desta podem

extrair-se sistemas de equações diferenciais que descrevem a evolução de um contorno ao longo do tempo. Esta evolução dependente não só da energia interna do contorno em sim, devido às forças de tensão e coesão entre pontos, mas também de forças externas que o vão guiar na imagem. Estas forças externas são determinadas de acordo com a finalidade do contorno e são geralmente derivadas de informação presente na imagem, como intensidades, gradientes e derivadas de ordem superior. Por fim, este sistema é implementado utilizando um método explicito de Euler que nos permite obter uma discretização do sistema em questão e nos proporciona uma expressão iterativa para a evolução do sistema de um estado prévio para um estado futuro que tem em conta os efeitos externos da imagem.

Depois de ser implementado o desempenho do programa semiautomático de rastreamento foi validado. Esta validação concentrou-se em duas vertentes: na vertente da robustez do rastreio de contornos quando acoplado a uma sonda de US e na vertente da eficiência temporal do programa e da sua compatibilidade com sistemas de aquisição de imagem em tempo real. Antes de se proceder com a validação este sistema de aquisição foi primeiro calibrado espacialmente de forma simples, utilizando um fantoma de cabos em N contruído em acrílico capaz de produzir padrões reconhecíveis na imagem de ultrassons. Foram utilizados padrões verticais, horizontais e diagonais para calibrar a imagem, para os quais se consegue concluir que os dois primeiros produzem melhores valores para os espaçamentos reais entre pixéis da imagem de US.

Finalmente a robustez do programa foi testada utilizando fantomas de 5%(m/m) de agar-agar incrustados com estruturas hipoecogénicas, simuladas por balões de água, construídos especialmente para este propósito. Para este tipo de montagem o programa consegue demonstrar uma estabilidade e robustez satisfatórias para diversos movimentos de translação e rotação da sonda US dentro do plano da imagem e mostrando também resultados promissores de resposta ao alongamento de estruturas, decorrentes de movimentos da sonda de US fora do plano da imagem.

A validação da performance temporal do programa foi feita com este a funcionar a solo utilizando vídeos adquiridos na fase anterior para modelos de contornos ativos com diferentes níveis de detalhe. O tempo de computação do algoritmo em cada imagem do vídeo foi medido e a sua média foi calculada. Este valor encontra-se dentro dos níveis previstos, sendo facilmente compatível com a montagem da atual da sonda, cuja taxa de aquisição é 25 fps, atingindo a solo valores na gama entre 40 e 50 fps.

Apesar demonstrar uma performance temporal e robustez promissoras esta abordagem possui ainda alguns limites para os quais a ainda não possui solução. Estes limites incluem: o suporte para um sistema rastreamento de contornos múltiplos e em simultâneo para estruturas-alvo mais complexas; a deteção e resolução de eventos topológicos dos contornos, como a fusão, separação e auto-interseção de contornos; a adaptabilidade automática dos parâmetros do sistema de equações para diferentes níveis de ruido da imagem e finalmente a especificidade dos potenciais da imagem para a convergência da abordagem em regiões da imagem que codifiquem tipo de tecidos específicos.

Mesmo podendo beneficiar de algumas melhorias este projeto conseguiu atingir o objetivo a que se propôs, proporcionando uma implementação eficiente e robusta para um programa de rastreamento de contornos, permitindo lançar as bases nas quais vai ser futuramente possível trabalhar para finalmente atingir um sistema autónomo de diagnóstico em US. Além disso também demonstrou a utilidade de uma abordagem de contornos ativos para a construção de algoritmos de rastreamento robustos aos movimentos de estruturas-alvo no a imagem e com compatibilidade para abordagens em tempo-real.

Palavras-Chave:

Ultrassonografia, procedimentos guiados por imagem, controlo visual, contornos ativos, momentos da imagem

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CM – Centre of Mass

CT – Computed Tomography

DOF – Degrees of Freedom

fps – frames per second

MRI – Magnetic Resonance Imaging

ROI – Region of Interest

US – Ultrasound

CHAPTER 1 - INTRODUCTION TO ROBOTIC-ASSISTED MEDICAL ULTRASOUND PROCEDURES

Due to the high-quality and often critical requirements of US tasks, the introduction of robotic systems in medicine is a growing practice in the development of medical applications, either in diagnostics or therapeutics. Since in many cases medical procedures require high precision and accuracy, it seems logical to introduce robotic systems for US tasks in order to be able to deliver expert-grade medical care to all citizens. Contrary to common belief, the main objective of robotic systems in medicine is not to replace the physician, but to provide better tools and information to enhance medical procedures, improving accurate diagnosis and promoting medical safety. Most of these systems work with a close relationship with the physician, where shared control strategies compensate for any unexpected and external disturbances. Additionally, in Globe regions where the availability of medical experts is reduced, such as in under-developed countries, the difference between life and death can many times be the introduction of robotic systems, that are autonomous, tele-operated or even cooperative, allowing remote experts to accurately treat or diagnose a patient at a distance and in real-time. One of the approaches that allows the creation such systems is the coupling of robotic technologies with image acquisition devices, allowing physicians to look and inspect inside the patient, for treatment or diagnosis. Due to mobility constraints and flexibility, ultrasound (US) imaging ends up being the most used technique where the acquired image integrates control of a medical procedures mediated by a robotic platform. With the objective of creating a future robotic system capable of delivering semi-automated medical diagnosis through ultrasound imaging, this M.Sc. thesis aims to develop a semi-autonomous tool for tracking defined structures of interest in US images, outputting meaningful spatial information of target structures in real-time. For this purpose, this dissertation is focused on the definition, implementation and validation of the base framework for image-controlled procedures.

1.1 Thesis Organization

The thesis is organized as follows. This first chapter covers different approaches for the inclusion robots that are tele-manipulated and image-controlled in the medical field, presenting several solutions and exposing their desirable features in terms of diagnostics and therapeutics. The second chapter lays the theoretical foundations for the creation of image-controlled systems, beginning with how a robot can respond to image features and how it can conciliate visual control with physician cooperation, and furthering into the underlying physical model that makes this possible. The third chapter addresses the schematic description on how a tracking algorithm is designed and implemented for the MATLAB platform and the fourth chapter covers the experimental setup to validate algorithm tracking properties. Finally, the last chapter discusses usability, pointing out current shortcomings and how they can be compensated.

1.2 State-Of-The-Art on Robotic-Assisted Ultrasound Procedures

In the rest of this chapter a description of the robotic systems that are nowadays being implemented in the medical medium, spanning from tele-operation to image guidance.

1.2.1 Tele-Operated Robotic Systems

Tele-operated robotic systems have the purpose of enabling expert-grade medical care to be available over large distances through interfacing medical experts with a purposely built robotic system. Such systems usually rely on two mechanical units: an end-effector and an actuator, meaning a tool through which the expert issues commands to the system and a robot architecture that will carry out the instruction, where, typically the communication between components is based on master-slave relationships. The communication between components, as described in Figure 1-1, is normally done through wireless network mediums, as Wi-Fi, 3G, 4G, through which important information needs to be sent bilaterally. This information includes actuator commands from master to slave as well as video stream and force feedback information, from slave to master, in order to guarantee accuracy and precision in the execution of diagnostics tasks.

Figure 1-1: Representation of the master-slave relationship between medical and robot workstations.

In the actuator department, early solutions, involve reduced DOF robots such as TER[1] with 4 DOF, that dates back to 2002, as described in [1], which employs a soft approach of a robotic system. This robot employs a set of cables coupled to two parallel systems of antagonistic pressure-controlled artificial muscles, as depicted in Figure 1-2(a) . The first system allows the translation of a coupled probe along the patient's surface and the second controls the orientation of the probe, thus enabling a total 3D control of the probe in the task space. Each of these systems are connected by artificial muscles, which are cylindrical rubber tube braided with a textile structure, with an air entry at an end and a plugged fixture at the other. Being a pneumatic system, when the pressure is raised each muscle contracts through conversion of the axial pressure forces into contraction forces by the textile shell, shortening the muscle length, as shown in Figure 1-2(b)-(c). Also, the muscle can act as a spring, whose stiffness is controlled through the pressure inside its cylinder, given the muscles are in an equilibrium position. This double compliance property allied with closed-loop position control, allow the system to adapt its contact force with the patients' body for a force feedback approach.

[1] TER - Robotic tele-ultrasound system

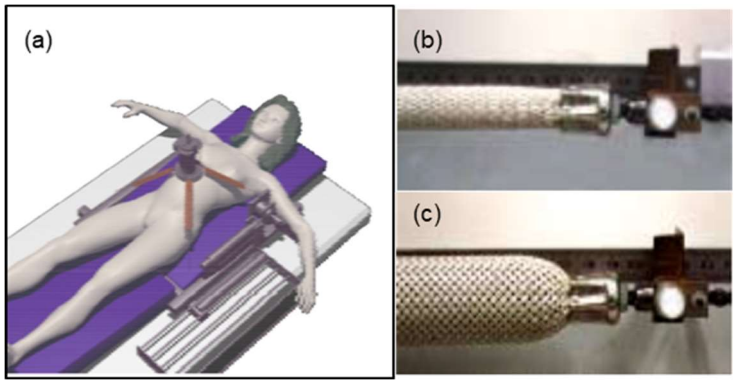

Figure 1-2: Depiction of the robot architecture of the TER system: (a) - CAD model of the 4DOF actuator frame; (b) - relaxed state of aMcKibben pneumatic muscle; (c) - Contracted state of a McKibben pneumatic muscle. [1]

Another method is the OTELO[2] tele-echography system, described in [2], which has a different approach, it uses a lightweight 6 DOF robot frame, enabling quick deployment of the system in isolated areas and for ER procedures in ambulatory voyages. The slave side consists of a mechanical structure coupled to a holding frame, shown in Figure 1-3(a), that is meant to be put on an area in the patient's body, establishing a region of interest, that needs to be manually positioned by the acting physician on site. The system is underactuated, which means that not every degree of freedom is used for robot position control, in fact only five are used, three for orientation and two for translation in the directions perpendicular to the probes' axis, as shown in Figure 1-3(b). The remaining degree of freedom, along the probe axis is designed to always maintain contact with the patient's skin, allowing the expert to control the contact force between the probe and the skin, allowing force control feedback.

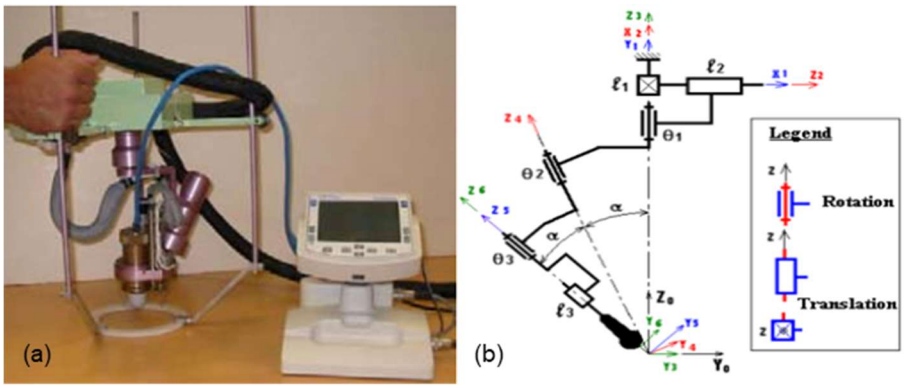

Figure 1-3: Depiction of the robot architecture of the OTELO system: (a) - Prototype of the light-weight 6DOF robot frame; (b) - Kinematic description of the robot frame motion in each DOF. [3]

The construction and design of custom robotic frames can be costly and time consuming, both for design and licensing, as such other approaches rely on using readily available robot architectures well established in the industrial sector, such as in the ROSE project [4]. In the early settings of this system, an anthropomorphic 7-DOF WAM robotic arm has been used, as shown in Figure 1-4, with an integrated force sensor-driven adaptive control strategy adaptive compliant motion control based on online stiffness estimation. The system can respond either for compliant environments as in abdominal US or in rigid environments as in thoracic US. The system is also able to behave while in free space motion and switch to admittance control when it detects the probe is in contact with a surface. The ROSE project is currently using two lightweight robots from Haption that were customized for tele-ultrasound diagnosis.

[2] OTELO -: mObile Tele-Echography using an ultra-Light rObot

As various diagnostic tasks require different applied forces, involving body areas with different rigidities (e.g., thoracic and abdominal ultrasound), the robotic system must adapt to conditions in order to provide the operator with the sense of tele-presence. As such, different control architectures have been deployed, as the ones described in [5] and [6], where the robot arm is ready to work in different rigidity regions, acting more or less compliant accordingly. Other approaches include motion compensation for robotic-assisted open tele-surgery of the heart [7] to provide a better working environment to the surgeon and enabling interventions without having to stop the organ itself, being less invasive to patient's condition.

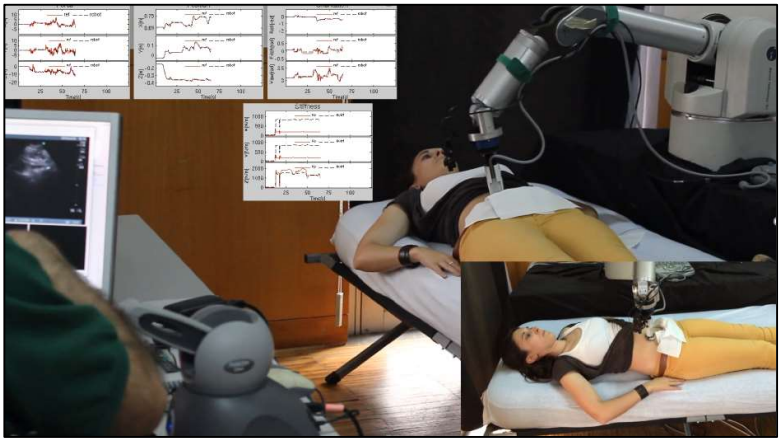

Figure 1-4: Early developments of the ROSE robotic system implemented for a routine ovarian ultrasound diagnostic procedure.

The common denominator in these approaches is the concern with accuracy of the procedures, which is an issue to be tackled both in the effector and human side of the equation. In the human side, the accuracy of the procedure is largely affected with the ability for the system to give haptic feedback to the operator. For this purpose, most master stations are equipped with haptic devices such as PHANTOM Omni. These devices have a defined workspace and can issue actuation commands with a resolution of 0.023 mm of resolution in the x, y and z directions, however, they are themselves joint actuated. Sets of motors in the joints enables a 3D sense of force resolution which allows the physician to experience the force feedback from the patient's skin, enable to transduce stiffness information with a resolution of 1.90 N/mm, so that the operator can sense the features of the area in analysis.

1.2.2 Image-Controlled Robotic Systems

Even when the conditions for tele-operating ultrasound systems are favourable there can be unexpected situations or system-bound limitations that can diminish the accuracy of the diagnostic. Communications failure and lag are one of the main reasons for accuracy loss, but robot movement limitation, due to effector workspace limits, and image quality loss over distance can also impact the performance of the procedure. These and other numerable limitations push to the conclusion that a simple master-slave relationship architecture although necessary, can benefit from shared control between commands issued by the medical expert and automatic behaviour based on the *in-situ* acquired image. The control strategies which revolve around image acquisition, processing and therefore behaviour planning on the robot are grouped in as Visual Servoing strategies. One of the most prominent groups dedicated to investigation and implementation of these kinds of strategies is Alexandre Krupa's Lagadic Group from IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires) of the University of Rennes, having developed approaches that range from ultrasound guided needle insertion [8]–[10] to 3D ultrasound guidance using image speckle noise correlation to estimate out-of-plane motions of the probe [11], [12].

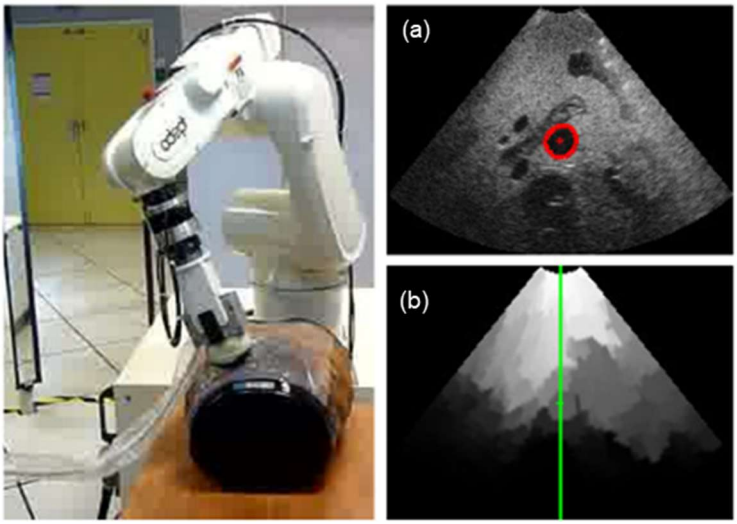

Figure 1-5: Implementation of visual servoing techniques based on both image moments (a) and image quality (b) for US probe guidance and image optimization in a humanoid phantom. [13]

Regarding visual servoing strategies it can be found that many approaches to the information present in an image can be taken to accomplish a panoply of behaviours: from tracking, to organ section search and probe adjustment. For instance, in [14] image moments of interest structures are used to prompt the robot to centre the probe in a specified organ section, as shown in Figure 1-5(a), allowing the robot to make both an automatic search in an area of interest for the pose of the probe that delivers the desired section and maintaining that pose over time. Other features, simpler or more complex, can be used to achieve the same behaviour, as described in [15], [16], where only through matching the pixel intensities between a desired and current image is enough to give reliable tracking results, while at the same time sparing time in image processing for systems with higher frame rates. Other applications use ultrasound image speckle and visual control for the automatic calibration of the robotized ultrasound probe system calibration [17], enabling the spatial reconstruction of a set of 2D US images along a line, in order to produce accurate probe localization in 3D space.

Lastly, another interesting application of visual servoing is described in [18]. In this case we have a tele-operated system that shares the control with an architecture that optimizes image quality. Using a model of propagation of ultrasounds along a scanning direction in tissue, an image quality map can be computed, defining which areas of the image are free from rigid tissue obstruction. The system can then compute which probe position maximizes the image quality in the central propagation line, shown in Figure 1-5(c), providing the medical expert with the best examination window possible with the least effort and time consumed.

Despite all the applications described earlier it can be pointed out that in a medical context the combination of the expertise of the acting physician with the accuracy of an automated system is yet to be completely achieved. This dissertation takes the path of visual servoing largely described by Krupa's group and further develops this path for the purpose of semi-automatic diagnosis supported by ultrasound features and robotic technologies.

CHAPTER 2 - THEORETICAL FRAMEWORK

This chapter covers the theoretical aspects that comprise the implementation of a visual servoing based approach. The definition of a visual servoing framework is presented and the image processing tools necessary for tracking structures on US image are also defined.

2.1 Visual Servoing Framework

Visual servoing control is defined in [19] as a group of control strategies that use computer vision data to control the motion of a robotic system. This data can be acquired either from an imaging device (camera, US probe) coupled directly onto a robotic manipulator, in which the image motion is induced by the movement of the robot itself, eye-in-hand approach (EiH), or with a set of fixed imaging devices observing the workspace motion of the robot from a stationary position, eye-to-hand approach. These approaches rely on techniques from image processing, computer vision and control theory, and as such are based on the aim of minimizing an error, e(t), which is typically defined by:

$$e(t) = s(m(t), p) - s^* \tag{3.1}$$

In general, the objective is to minimize the error vector, which can be defined as the difference of a measured feature vector s and a desired feature vector s^*. In turn, the measured features vector can be defined as a function of a set, $m(t)$, image feature measurements (i.e., coordinates of interest, object centroids, Etc.) and p, a set of parameters with additional information on the geometrical model of the acquisition system.

As a very general formulation, this expression encompasses an immense variety of approaches, as different sets of feature vectors can be defined for different applications or tasks. The size of the feature vector is also not fixed and is also dependent on the application. For instance, we have the many approaches taken by Krupa's group. In [15], the proposed task is structure tracking and the feature vector is taken as the pixel intensity of an acquired image, which is compared to a reference image section to which the robot must navigate. In [20] the task is to position the robot on a desired section of a target organ, in such way that the only necessary information is given by the contour of the target itself, as such the feature vector is defined as a reduced set of information. As demonstrated the control schemes mainly differ in the definition of the feature vector, but once it is selected the design of the control law is simpler, being a velocity controller the most straightforward approach.

To establish a velocity controller, it is required to define a relationship between the temporal variations of the feature vector and the velocity of the moving imaging device. Defining the instantaneous velocity vector for the probe as $V_p(v_p, \omega_p)$, separating v_d as the translational velocity and ω_p as the rational velocity, this relationship can be expressed as follows:

$$\dot{s} = L_s V_p \tag{3.2}$$

In which $L_s \in \mathbb{R}^{k\times 6}$ is called the interaction matrix related to s, also called a feature Jacobian, that can possibly consider both feature vector information and the acquisition system's geometrical information. If we take (1) and (2) into account the variation of the feature error $e(t)$, can be expressed by a similar relationship:

$$\dot{e} = L_e V_p \tag{3.3}$$

Usually the next step is to force an exponentially decreasing behaviour for the feature error, meaning that $\dot{e} = -\lambda e$. If now we consider V_d as a commanded velocity, we reach the basic control law for visual servoing:

$$V_p = -\lambda L_e^{+} e \tag{3.4}$$

In which $L_e^{+} \in \mathbb{R}^{6\times k}$ is the Moore-Penrose pseudo-inverse matrix associated to the system´s definition of interaction matrix. As in real-time approaches is nearly impossible to determine an exact value for the interaction matrix, an estimation operator must be defined. Usually the estimation taken is one as identifying the error interaction matrix with the feature interaction matrix, as such as that $\widehat{L_e} = L_s$, arriving at a final basic law:

$$V_p = -\lambda L_s^{+}(s - s^*) \tag{3.5}$$

Which dictates the probe velocities needed to apply to reach convergence to a desired feature vector. As the previous approach is a rough approximation it inserts uncertainty into the control law, revealing some initialization problems and non-exponential behaviour at start, which is why some authors prefer to define a more robust operator, such as:

$$\widehat{L_e^{+}} = \frac{1}{2}(L_s + L_{s^*})^{+} \tag{3.6}$$

Involving the interaction matrices computed both at a current feature vector position and on the desired position, which can guarantee a better overall behaviour and smooth trajectories both in the image space and in the robot's task space.

As we aim to apply visual servoing to diagnostics scenarios, other control issues beyond image must be considered. Additionally, as the shape of target-structures in ultrasound imaging are highly dependent on the contact forces between the probe and the patient's body, force control also needs to be addressed implicitly or explicitly, together with visual servoing. For instance, using switched control, in which one kind of control is turned on while the other one is turned off is always an option. More advanced approaches are however described in [21] enabling joint force and visual control and hierarchic control.

The principle of the first approach is to divide the various components of the commanded velocities in appropriate way. This means the definition of a partitioned control law and a partitioned interaction matrix, as in:

$$\dot{s_t} = L_{s_t} V_c = [L_v \quad L_\omega] \begin{bmatrix} v_c \\ \omega_c \end{bmatrix} = L_v v_c + L_\omega \omega_c \tag{3.7}$$

In this case we have separation of translational and rotational components. But in many cases even these components can be divided in order to better decouple the different degrees of freedom for 3D motion. If we have an underactuated system, it is possible to make an optimal approach in fusing two types of control using what is called the redundancy framework. The approach is based on the hierarchization of tasks, defining a primary and a secondary, where the constraints of the second are projected into the null space of the vision-based task. This means that the secondary task will not affect the regulation of the primary task error to zero. Defining the global error as:

$$e_g = \widehat{L_e^+} e + P_e e_s \tag{3.8}$$

In which, e_g is the newly defined global error and $P_e = \left(I_6 - \widehat{L_e^+}\widehat{L_e}\right)$, the null space projection matrix. This approach enables using unconstrained degrees of freedom to complete the secondary task, be it vision-based or force feedback related, for which in this case the set of features may be a constant set of contact forces.

2.2 Fourier Active Contour Model

Keeping close to the approach detailed in [22] after defining how the robot should be controlled, the focus must shift to image processing tools necessary to drive the implementation forward. We plan to follow a similar approach for tracking target structures in the framework of an active contours.

Defined first by Terzopoulos in [23], an active contour, also known as a snake, is an energy-minimizing curve that can be guided by external constraint forces and can be influenced by image bound forces to converge into specific image features, such as lines and edges. These contour models lock on to nearby features enabling their accurate localization and providing a unified solution to several computer vision problems, such as edge, line and subjective contour detection, motion tracking and even matching in stereo images. Unlike other detection techniques these models are active, which means they are always minimizing the functional that defines their global energy, which is the motor for their dynamic behaviour. Formally these models encompass a global energy defining functional to which two types of constraint forces contribute, internal and external forces. If we defined a curve parametrically by $C(u) = \left(x(u), y(u)\right)$, then we can define the total snake energy as:

$$E_S^T = \int_0^1 E_S\big(C(u)\big)du = \int_0^1 E_{int}\big(C(u)\big) + E_{ext}\big(C(u)\big)du \tag{3.9}$$

The internal energy term derives from the internal forces of the contour, associated with bending and the external derives from image constraint forces specifically defined to make the contour converge on the desired features.

Early implementations of active contours operated on non-parametrized contour, using greedy algorithms that compute forces applied on each point of the defined contour, which implies considering every point a variable, meaning an implementation time that grows with $\mathcal{O}\big(N_p\big)$, the number of ordered points that defines it, which for a detailed approach can become incompatible with real-time implementation.

2.2.1 General Parametric Formulation for Active Contours

One way to formulate this concept in a way that reduces the number of variables that are needed is to re-parametrize the contour. The general form for a curve parametrization is defined as follows:

$$C_q(u) = x_c + \sum_k^n q_k \Phi_k(u) \tag{3.10}$$

In this expression C_q is linearly defined and depends on parameter vector q of dimension n, x_c is a point belonging to the contour and Φ_i is a two-dimensional matrix belonging to a set of orthogonal or descriptor base functions. These descriptor functions can appear in many forms, such as in [24] where a polar function framework is used to drive an active contour approach, that although solving the algorithm runtime problem cannot support non-convex curves. In [25] the contour curve is defined in the B-Spline framework assuring smoothness constraints and, through reduced description with a parameter vector of curve nodes and weights, also reduces the run time making it compatible with real-time and robust to image noise.

Following the definition in [24], [26], [27] for the dynamic behaviour, we consider the system will converge to minimize its energy functional according to the Euler-Lagrange equation. Considering q, the vector of contour descriptors, as a vector of system variables, the kinetic energy, T, and the potential energy, U, of the contour must follow and are defined by the expressions:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial q_i}\right) - \frac{\partial U}{\partial q_i} = Q_i \tag{3.11}$$

$$T = \int_{C_q} \frac{1}{2} \mu \, \|C_t\|^2 du \tag{3.12}$$

$$U = \int_{C_q} \frac{1}{2} (k_1 \|C_u\|^2 + k_2 \|C_{uu}\|^2) du \tag{3.13}$$

In this formulation Equation (3.11) is the embodiment of the energy minimization principle in the form of Euler-Lagrange equation, where:

$$Q_i = \int_{C_q} f^{\mathrm{T}}(u) \frac{\partial C_q(u)}{\partial q_i} du \tag{3.14}$$

and Q_i is the generalized external force applied to the contour relative to the variable component q_i. Equation (3.12) is the definition of kinetic energy, in which:

$$C_t = \frac{\partial C_q}{\partial t} = \sum_i \dot{q}_\iota \, \Phi_i \tag{3.15}$$

and μ represents the linear mass density of the contour. Finally, Equation (3.13) is the definition of a contour's potential energy according to Tikhonov's regularization problem, where:

$$C_u = \frac{\partial C_q}{\partial u} = \sum_i q_i \, \Phi_i' \tag{3.16}$$

$$C_{uu} = \frac{\partial^2 C_q}{\partial u^2} = \sum_i q_i \, \Phi_i'' \tag{3.17}$$

and k_1 represents the extension component and k_2 the curvature component of the potential energy term. Combining Equations (3.11) -(3.13) with (3.14) -(3.16), keeping in mind that $\|\Phi\|^2 = \Phi^T\Phi$, we arrive at:

$$\begin{cases} \frac{d}{dt}\left(\frac{\partial T}{\partial q_i}\right) = \int_{C_q} \mu\left(\sum_j \ddot{q}_j \Phi_j^T \Phi_i\right) du \\ \frac{\partial U}{\partial q_i} = \int_{C_q} k_1\left(\sum_j q_j \Phi_j'^T \Phi_i'\right) + k_2\left(\sum_j q_j \Phi_j'^T \Phi_i'\right) du \end{cases} \tag{3.18}$$

From the definition of generalized external force, we can divide f, the applied external force to each contour point, in two components: a viscous and the image bound ($f = f_{visc} + f_{image}$). Then we can define a viscous force distribution due to the medium in the form: $f_{visc} = -\gamma C_t$, as a regular viscous force would and the image bound forces as a conservative force, deriving from a potential, in the form: $f_{image} = -\nabla E$. As such we would reach the results:

$$\begin{cases} \int_{C_q} f_{visc}^{\mathrm{T}}(u) \frac{\partial C_q(u)}{\partial q_i} du = -\int_{C_q} \gamma\left(\sum_j \dot{q}_j \Phi_j\right)^T \Phi_i du \\ \mathrm{Q_{im}}(q_i) = -\int_{C_q} E_{im}^T(u) \frac{\partial C_q(u)}{\partial q_i} du \end{cases} \tag{3.19}$$

Going back to Equation (3.11) and inputting all the previous results the system of differential equations that regulates the dynamic behaviour of a general parametric active contour:

$$M\ddot{q} + C\dot{q} + Kq = Q_{im}(q) \tag{3.20}$$

The model vaguely represents a flat two-dimensional medium in which the contour floats, evolving though a force balance, where M represents the inertia of the curve, C represents the effect of the medium's viscosity and K represents the joined effect of the elasticity and bending forces of the contour, being normally decompose as $K = K_1 + K_2$. The definitions for these matrices derive from the previous model, and are defined as follows:

$$\begin{cases} M = [M_{mn}] = \mu \int_{Cq} \Phi_m^T \Phi_n du \\ C = [C_{mn}] = \gamma \int_{Cq} \Phi_m^T \Phi_n du \\ K_1 = \left[K_{1_{mn}}\right] = k_1 \int_{Cq} \Phi_m'^T \Phi_n' du \\ K_2 = \left[K_{2_{mn}}\right] = k_2 \int_{Cq} \Phi_m''^T \Phi_n'' du \end{cases} \tag{3.21}$$

2.2.2 Fourier Formulation for Active Contour

Following closely the approach introduced by Li, Krupa and Collewet in [26], [27], a contour can be re-parametrized by taking the Fourier series of its components. Starting from the following definition:

$$C_q(u) = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix} = \begin{bmatrix} a_o \\ c_0 \end{bmatrix} + \sum_{l=1}^{h} \begin{bmatrix} a_l & b_l \\ c_l & d_l \end{bmatrix} \begin{bmatrix} \cos(lu) \\ \sin(lu) \end{bmatrix} \tag{3.22}$$

With $u \in [0,2\pi]$ and a_l, b_l, c_l, d_l being the coefficients of the Fourier series decomposition. The information on the shape of a contour will be condensed in the form of coefficients representing the weights of different ellipses that are drawn in harmonic proportion. Developing (3.22) into the standard formulation, by expanding the Fourier series:

$$C_q(u) = \begin{bmatrix} a_o \\ c_0 \end{bmatrix} + \sum_{l=1}^{h} (a_l \begin{bmatrix} \cos(lu) \\ 0 \end{bmatrix} + b_l \begin{bmatrix} \sin(lu) \\ 0 \end{bmatrix} + c_l \begin{bmatrix} 0 \\ \cos(lu) \end{bmatrix} + d_l \begin{bmatrix} 0 \\ \sin(lu) \end{bmatrix}) \tag{3.23}$$

A variable vector q with size $(4h + 2 \times 1)$, can be defined as $q = (a_o, a_1 \dots a_h, b_1 \dots b_h, c_0, c_1 \dots c_h, d_1 \dots d_h)$, where h represents the number of preserved harmonics, that encodes the contour shape information and from which a curve can be reconstructed. And the basis set of functions, Φ_i, becomes apparent in (3.23). As is recurrent from Fourier analysis the more harmonic coefficients preserved the more accurate the representation of a contour will be and for which Nyquist's sampling theorem stands true. Meaning that for a contour of N_p sample points the maximum number harmonics for accurate reconstruction of a contour from descriptors is $N_p/2$, after which aliasing effects corrupt the description.

This parametrization with elliptic Fourier descriptors holds the advantage over polar description in the sense that it is compatible with non-convex curves and that its description base is a fully orthogonal set of 2π-periodic function, meaning that while determining the dynamics matrices only same index base functions will deal non-zero integral values, meaning these matrices can be defined only through their diagonal, shown as follows:

$$\begin{cases} M = \mu \operatorname{diag}(2\pi, \pi \dots \pi, \pi \dots \pi, \\ \qquad 2\pi, \pi \dots \pi, \pi \dots \pi) \\ C = \gamma \operatorname{diag}(2\pi, \pi \dots \pi, \pi \dots \pi, \\ \qquad 2\pi, \pi \dots \pi, \pi \dots \pi) \\ K_1 = k_1 \operatorname{diag}(0, \pi l^2 \dots \pi h^2, \pi l^2 \dots \pi h^2, \\ \qquad 0, \pi l^2 \dots \pi h^2, \pi l^2 \dots \pi h^2) \\ K_2 = k_2 \operatorname{diag}(0, \pi l^4 \dots \pi h^4, \pi l^4 \dots \pi h^4, \\ \qquad 0, \pi l^4 \dots \pi h^4, \pi l^4 \dots \pi h^4) \end{cases} \tag{3.24}$$

In order to simplify the system of equations, and according to [28], the mass of each contour point is defined as zero, meaning that $\mu = 0$ and $M = [0]$, assuming a system where all inertial effects are neglected and/or undesired, the system of ODE's is reduced to:

$$\dot{q(t)} = -\mathrm{C}^{-1}Kq(t) + C^{-1}Q_{im}(q(t)) \tag{3.25}$$

For the system to be able to be implemented on a computer it needs to be discretized, which can be done identifying as $\dot{q} = (q_{n+1} - q_n)\Delta t$ and $q = q_n$, employing an explicit Euler's method, we reach the iterative formula for the evolution of the contour in descriptor space:

$$q_{n+1} = (I_{4h+2} - \Delta t C^{-1}K)q_n + \Delta t C^{-1}Q(q_n) \tag{3.26}$$

This is a useful formulation because it resembles the definition of a discrete state-space defined system, like as follows:

$$\begin{gathered} q[n+1] = A * q[n] + B * Q_{im}[n] \\ y[n] = C * q[n] \end{gathered} \tag{3.27}$$

Where $A = [I_{4h+2} - \Delta t C^{-1}K]_{4h+2}$, $B = [\Delta t C^{-1}]_{4h+2}$, $C = I_{4h+2}$ are the state matrices, q is the state vector and Q_{im} acts as the input vector for the system.

2.2.3 Definition of External Image Forces

The premise of this model is its ability to converge into desired features that can appear in an US image for which it relies in the action of external forces specifically modelled to repel or attract the contour. As initially approached by Terzopoulos in [23], these forces may derive from definition of a global image energy functional, that can be defined by the linear combination of specific image functionals, as such:

$$E_{global} = \sum_k w_k E_k \tag{3.28}$$

In practice the specific energy functionals used here are the line, edge and termination functionals, as they are defined in [23].The simplest one, the line functional, can be expressed as the image intensity itself, then if we define it as $I(x, y)$, then the functional yields:

$$E_{line} = -I(x, y) \tag{3.29}$$

Depending on the sign of its corresponding weight, w_{line}, it will make the contour be attracted either to bright lines or dark ones.

The edge functional is defined based on the image gradient, meaning that changes in gradient is what defines an edge of the image. This term works well, in the sense that it can make the contour converge into an edge, however it has a limited range of operation, meaning that if the contour is defined very far away from it, it won't converge in useful time. Other undesired result particular to the specific contour model implemented is that for very steep potentials, instead of converging, the contour simply oscillates about the desired edge, producing an erroneous result as well as consuming more time due to the algorithm never reaching a proper settling time. To compensate this behaviour, to smoothen and add capture range to the edge functional, before computing the gradients, the image is filtered with a gaussian smoothing kernel, $G(\sigma)$, meaning the edge functional can be defined as:

$$E_{edge} = -\|\nabla[\mathrm{G}(\sigma) * I(x,y)]\|^2 \tag{3.30}$$

The last functional used in this approach is the termination functional, it provides the contour with the ability to be attracted to line segment terminations, corners and sensitive to subjective contours. Using the previously defined smoothed image identifying $C(x,y) = \mathrm{G}(\sigma) * I(x,y)$, the functional is defined as the of the gradient angle θ in the perpendicular direction, introducing a measure of curvature:

$$E_{term} = \frac{\partial \theta}{\partial n_T} = \frac{C_{yy}C_x^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{\left(C_x^2 + C_y^2\right)^{3/2}} \tag{3.31}$$

Where C_n are the first and second order partial derivatives of the smoothed image. The combination of different functional weights can produce different but useful behaviours in the contour evolution. As an example, combining E_{edge} and E_{term}, shown in **Figure 2-1**(b)-(c), with similar weights enables the contour to effectively converge into subjective edges. And combining E_{line}, shown in **Figure 2-1**(a), and E_{edge}, enables an improved edge convergence depending on whether the region inside the contour is lighter or darker.

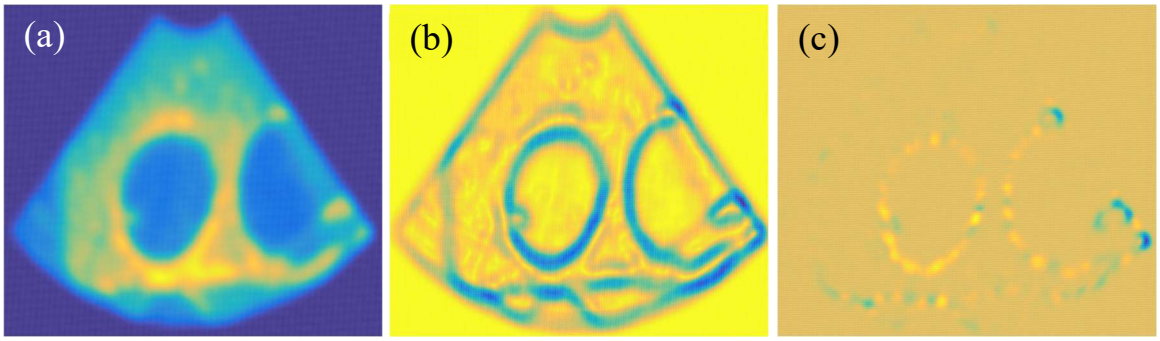

Figure 2-1: Representation of different image potentials: (a) - Line potential; (b) - Edge potential; (c) - Termination potential.

Due to the imaging principle of US, usually the image is corrupted with speckle noise, which is prejudicial to the convergence of active contours in the sense that it blurs edges and masks interest structures and the usual step is to preform speckle filtering, with bilateral or median filters. However, the image noise characteristics (mean and variance) can also be used in a statistical approach to an active contour algorithm. As described in [29], the external forces can be modelled recurring to the definition of a statistical image potential, derived from the measurement of the image mean, variance or even the introduction of a multivariate vector for region identification through convolution with texture kernels. This delivers an approach that can deal with image noise in a more robust way. Due to simplicity and time constraints our first iteration of implementing active contours will be based on image noise filtering.

2.3 Model Stability and Dimensioning

Along with the definition of the model for Fourier active contours, come a set of control parameters that ultimately define the dynamic behaviour of the model itself, which effects interact with one another to create the bigger picture. The set of parameters that directly affect the contour behaviour are $\gamma, k_1$and k_2, due to altering the control matrices defined in (3.24), however, the parameter N_p, the number of points in a contour and h, the number of harmonics, also have effect, due to them changing the intrinsic

properties of the parametrization. Lastly, it is important to define the time step, Δt, for the algorithm, that defines how fast the contour algorithm will evolve.

2.3.1 Definition and Effects of Internal Force Parameters

Since the model is defined by a linear differential equation, thus translatable into a space-state system, these interactions between the effects of parameters are established by the eigenvectors of the state space matrix, which from (3.25) is identifiable as $A = -C^1(K_1 + K_2)$. Since these are all diagonal matrices, their eigenvalues are their diagonal values, meaning, that each eigenvalue associated with each harmonic can be defined, as:

$$\lambda_l = -\frac{k_1 l^2 + k_2 l^4}{\gamma}, 0 \leq l \leq h \tag{3.32}$$

Since k_1 and k_2 are always positive the equation system is always stable since for every harmonic value the corresponding pole is on the left side of the complex plane. It is also useful to recognize that (3.32) is in fact a balance of forces between the internal forces of the contour and the viscous forces of the medium, since it can be used as a base definition tailor the behaviour of the system. For the parameters to be non-arbitrary we define another balance of forces, between the components of the internal force, elasticity and curvature. For a weak conditioning, these force balances can then be translated into the expressions:

$$\begin{gathered} 0 \leq \frac{k_1 h^2 + k_2 h^4}{\gamma} \leq \alpha, \qquad \alpha \in \mathbb{R}^+ \\ 0 \leq \frac{k_1 h^2}{k_2 h^4} \leq \beta, \qquad \beta \in \mathbb{R}^+ \end{gathered} \tag{3.33}$$

These balances are defined according to the harmonic of highest order, since they represent the eigenvalue with the highest value for fixed values of γ, k_1 and k_2. As such from the defined balances, with given choices of balance ratios, the conditions for the choice of parameters k_1 and k_2 become:

$$\begin{cases} 0 < k_1 \leq \dfrac{\alpha\beta\gamma}{(\beta+1)h^2} \\ \dfrac{k_1}{\beta h^2} < k_2 \leq \dfrac{\alpha\gamma - h^2 k_1}{h^4} \end{cases} \tag{3.34}$$

The definition of these ratios is also important to understand the physical behaviour of the system. As such, after analysing the ratio expressions and by experimental observation, for α we can state that:

- If $0 < \alpha \leq 1$: We obtain contours mainly ruled by the viscous force from the medium, meaning that, it will sensitive to external forces, however producing a non-smooth evolution, since regularization terms are undervalued;
- If $\alpha > 1$: We obtain contours mainly ruled by internal forces, dealing low sensitivity to external forces but a smooth contour evolution.

For the case of ratio β, the balance for the components of internal force, we can state that:

- For $0 < \beta \leq 1$: We obtain contours whose evolution is mainly conditioned by elastic forces. Meaning that the contour will contract rapidly, however since the curvature regularization term is undervalued, it might evolve forming sharp edges;
- For $\beta > 1$: We obtain contours whose evolution is mainly conditioned by contour regularization forces. Meaning a slow contracting contour, whose curvature more rapidly evolves into unit at every point.

For a strong conditioning of the system, meaning a set of parameters with fixed ratios we have the expressions:

$$\begin{cases} k_1 = \dfrac{\alpha\beta\gamma}{(\beta+1)h^2} \\ k_2 = \dfrac{\alpha\gamma}{(\beta+1)h^4} \end{cases} \tag{3.35}$$

However useful a strong conditioning might look, it becomes limiting to the characteristics of the system, in the sense that one needs to know the exact value of the ratios for a given system configuration.

2.3.2 Viscous Force and Sensibility to External Forces

In order to define a measure of the contours' sensibility to the action of external forces, the approach taken is like before, through force balances. As such, considering a system where the internal forces are null, we have $q(\dot{t}) = C^{-1}Q_{im}(\mathrm{t})$, meaning the evolution is conditioned by the balance between viscous force and the external force. In this case we can think of $S = \|dq/dt\|_2 = \|C^{-1}Q_{im}\|_2$ as a global measurement for the action of external forces on the contour and as such if we can establish a global balance of forces as:

$$S(t) = \frac{\|Q_{im}(t)\|_2}{sqrt(\lambda_{max}(C))} = \frac{\|Q_{im}(t)\|_2}{\pi\gamma} \tag{3.36}$$

To achieve a measure that is not time-dependent we have to define a mean sensitivity. This means defining an approach to define the mean 2-norm of the image acting force. For this we define a random force vector in which the applied force components (x, y) to each point in the contour is a random variable with a standard Gaussian distribution [$F_x^n \sim N(0,1)$ and $F_y^n \sim N(0,1)$]. Following transformation (11), the applied force is translated into the descriptor space, and since it defines a linear combination of the original force components with the components of $\partial C/\partial q$, its own components are normally distributed $[{Q^i}_q \sim N(\mu_i, \sigma_i)]$ and since the mean for each cartesian component is zero we assume a priori that $\mu_i = 0$. The next step is to take the 2-norm of the Q_{im}, this means the creation of yet another random variable, $\|Q_{im}\|$ where:

$$\|Q_{im}\|^2 = \sum_{n=1}^{4h+2} \left(\frac{Q_i^{im}}{\sigma_i}\right)^2 \sim \chi^2(4h+2) \tag{3.37}$$

To overcome the issue of having to estimate a mean vector and then its norm, the variance for each component is estimated and a general estimated variance is used to normalize the image force vector: $\hat{\sigma}_{gen} = mean(\widehat{\sigma_1}, \dots \hat{\sigma}_{4h+2})$. As such we normalize and assume homoscedasticity such that:

$$\frac{\|Q_{im}\|^2}{\hat{\sigma}_{gen}^2} = \frac{1}{\hat{\sigma}_{gen}^2} \sum_{n=1}^{4h+} \left(Q_i^{im}\right)^2 \sim \chi^2(v) \tag{3.38}$$

In this case v, should be $4h+2$, however since we take assumptions, we can only aim to be significantly close. Through an iterative Monte-Carlo based process is possible to generate samples of random image forces, compute the estimators for the mean squared 2-norm and fit the chi-square distribution with v degrees of freedom, allowing the definition of confidence intervals for that parameter. Since $E\left(\chi^2(v)\right) = \|Q_{im}\|_m^2 = v$, then we can also estimate de confidence intervals for the values of the 2-norm of the force vector as:

$$\|Q_{im}\|_m = \sqrt{\hat{\sigma}_{gen}^2 v} \tag{3.39}$$

However, there must be acknowledgement that, since we make the assumption that all the means from the image force components are null, which is not always true, this estimation method for a high number of samples can overestimate the intervals for $\|Q_{im}\|$. In order to compensate this effect, we must turn to the true distribution of $\|Q_{im}\|^2$, when $\mu_i \neq 0, i = 0, \dots, 4h+2$, which is a non-central chi-square distribution:

$$\frac{\|Q_{im}\|^2}{\hat{\sigma}_{gen}^2} = \frac{1}{\hat{\sigma}_{gen}^2} \sum_{n=1}^{4h+2} \left(Q_i^{im}\right)^2 \sim \chi^2(v_{nc}, \lambda_{nc}) \tag{3.40}$$

Where the non-centrality parameter is $\lambda = \sum_{i=1}^{4h+2} \mu_i^2$. This means that estimating the non-central distribution that fits the image force sample, yields:

$$\|Q_{im}\|_m = \sqrt{\hat{\sigma}_{gen}^2 (v_{nc} + \lambda_{nc})} \tag{3.41}$$

With this definition, the overestimation is somewhat compensated in the sense that the mean of the non-normalized 2-norm vector, falls within the confidence intervals for $\|Q_{im}\|_m$, that are derived from the intervals of the non-central distribution parameters. This estimation needs to be made each time we decide to use a different number of harmonics, since the higher the number of harmonics, the more sensible the contour. Finally, this yields our self-defined mean sensibility:

$$S_m^n = \frac{\left(\sqrt{\hat{\sigma}_{gen}^2 (v_{nc} + \lambda_{nc})}\right)_n}{\pi\gamma} \tag{3.42}$$

2.3.3 Model Compensation with Balloon Force for Far Initialization

One of the main issues with the use of parametric active contours is their necessity to be initialized with a contour that lays in the neighbourhood of the desired features to be tracked. To deal with far contour initializations we could introduce pressure forces, as done in [25] that depend on the normal direction of the contour at each point, however that is time consuming so, as done in [24], yet another contour energy potential term is added to the internal forces, defined as $E_s = k_3 S_q$. When $k_3 > 0$, we will have a fast-contracting contour, meaning better faraway initializations for features already inside the contour. When $k_3 < 0$, we can have expanding contours, enabling initialization of contours inside the features to be tracked. The force deriving from this potential proportional to the area enclosed by the contour is defined by means of the Green theorem as such as that:

$$\begin{aligned} S_q &= \int_0^{2\pi} x(u) \frac{dy(u)}{dt}\, du - y(t) \frac{dx(u)}{du}\, du \\ &= \pi\left(\sum_{l=1}^{h} l(a_l d_l - b_l c_l)\right) \end{aligned} \tag{3.43}$$

If we define the force as derivative of the potential for each harmonic, then $K_3 = \partial E_s / \partial q = k_3 P S q$, where:

$$\begin{aligned} S = \text{diag}\,(&0, \pi l \ldots \pi h, \pi l \ldots \pi h, \\ &0, \pi l \ldots \pi h, \pi l \ldots \pi h) \end{aligned}$$

And P is a permutation matrix defined as:

$$\begin{gathered} P_{(1;1)} = P_{(2h+2;2h+2)} = 0 \\ P_{(1+l;3h+2+l)} = P_{(3h+2+l;1+l)} = 1 \\ P_{(h+1+l;2h+2+\,)} = P_{(2h+2+\,;\,h+1+l)} = -1 \\ l = 1, \ldots, h \end{gathered} \tag{3.44}$$

This alternative representation of the K_3 matrix as linear operation mediated by a permutation matrix allows faster computation times because it avoids computing a permutation of vector q at each iteration of the contour evolution and allows an analysis of the effect of this matrix in the general eigenvalues of the differential equation, meaning it can be used to better design the value of the scalar k_3, that mediates this surface force. As such, with the balloon force effect the eigenvalues become:

$$\lambda_l = -\frac{k_1 l^2 + k_2 l^4 \mp k_3 l}{\gamma}, 0 \le l \le h \tag{3.45}$$

For previously fixed values of k_1 and k_2, the positive eigenvalues introduced by this force do not affect the overall stability of the system, for $k_3 > 0$ and the same for negative eigenvalues, for $k_3 < 0$. However , this duality means that for $k_3 > 0$, $k_1 l^2 + k_2 l^4 - k_3 l > 0$ and that for $k_3 < 0$, $k_1 l^2 + k_2 l^4 + k_3 l > 0$, which allows to establish the global stability condition for each harmonic as:

$$\left| \frac{k_3}{k_1 l + k_2 l^3} \right| < 1, 0 \le l \le h \tag{3.46}$$

From this condition it can be established that if we desire a totally stable system, then the balloon force will always be significantly less important than the internal forces, which defeats the purpose of introducing this force as means of compensating the dynamics of the original system in order to make it converge faster, which however can be achieved by splitting condition (3.43) into two conditions. The rationale behind this separation is that the harmonic with the role of defining the global surface enclosed by the contour is the first one, $h = 1$, while the others deal with local areas related to contour deformation, for $h > 2$. As such to control inflation or deflation of the contour, mediated by balloon force, we have:

$$\begin{aligned} \left|\frac{k_3}{k_1 + k_2}\right| &= \rho \\ \left|\frac{k_3}{2k_1 + 8k_2}\right| &< 1 \end{aligned} \tag{3.47}$$

While the first condition establishes the force balance between internal forces and balloon force in the first harmonic of the contour the second guarantees that for higher harmonics the dynamic behaviour of the system remains stable. These conditions yield the definition of k_3 and bounds for ρ as:

$$\begin{aligned} |k_3| &= \rho(k_1 + k_2) \\ 0 < \rho &< \frac{2k_1 + 8k_2}{k_1 + k_2} \end{aligned} \tag{3.48}$$

In terms of contour the effects of the value of ρ are as follows:

- For $k_3 > 0$: The contour always behaves in a contracting fashion regardless of the value of ρ. For values of ρ between zero and one, the additional contraction force is relatively less intense than the internal regularization forces, meaning that the contour evolves rapidly into an ellipse. For values bigger than one the contraction forces are predominant and the contour contracts in a less regulated fashion, allowing the formation of edges in the contour;
- For $k_3 < 0$: The contour presents two distinct behaviours, dependant on the value of ρ. For values between zero and one, the regularization forces are predominant over the expansion forces, meaning the contour will still contract, however at a lower rate and in with very smooth contour features. For $\rho = 1$, the rate of contraction equals the rate of expansion and as such the global area remains the same, however since higher order harmonics are still at play the contour will regularize regardless, tending to an ellipse. For values between one and its maximal bound, the contour will acquire an expanding behaviour, which will enable initialization from the inside of structures of interest. An important note is that, since higher order harmonics are stable the contour will expand in the way best suited to the local shape of an interest structure, showing good sensibility to external and viscous force action.

2.3.4 Discrete System and Stability in Evolution

Since the model of the Fourier active contour defined is continuous, in order to be useful and implemented it needs to be in its discretized from, which was already defined in expressions (3.26) and (3.27), arriving at the discrete space-state model for the evolution of the contour. The discretization implies adding a final parameter to the model, Δt, the time step between iterations. Among other effects, this parameter will definitively command the speed of evolution of a contour and their overall stability in all harmonics over time of evolutions, meaning that the proper time step will either maintain the contour characteristics designed previously, slightly deviate from them, or turn the contour completely unstable. In order to makes this selection, we take from (3.27) the state-space matrix of the system, which in this case has eigenvalues defined as:

$$\lambda_l^d = 1 - \frac{\Delta t(k_1 l^2 + k_2 l^4 \mp k_3 l)}{\gamma}, 0 \leq l \leq h \tag{3.49}$$

Since were in the discrete domain, in order for the contour model to be stable, then the modulus of all its eigenvalues must be contained in the unitary complex circle. Since all eigenvalues are dependent on the harmonics in a crescent manner, we need only to constraint the highest one to be sure all others are stable as well, which leads to the stability condition:

$$\left| 1 - \frac{\Delta t(k_1 h^2 + k_2 h^4 + k_3 h)}{\gamma} \right| < 1 \tag{3.50}$$

Considering that all other parameters are already fixed then, the time step can be bounded by:

$$0 < \Delta t < \frac{2\gamma}{k_1 h^2 + k_2 h^4 + k_3 h} \tag{3.51}$$

With this condition we can choose the proper time step, however it should be used cautiously, for it does not distinguish oscillating stability with regular exponential stability. Again, since we are on the discrete domain, any eigenvalue with modulus less than one is considered stable however if its value is negative, we will have oscillating behaviour, which is ill advised due to reduced convergence of the active contour in these states. Splitting condition (3.52) in half we will have harmonics with all correspondent eigenvalues positive, which means overall smooth and contour evolution and convergence.

$$0 < \Delta t < \frac{\gamma}{k_1 h^2 + k_2 h^4 + k_3 h} \tag{3.52}$$

If it is in our interest to be able to better control the smoothness, stability and evolution speed of the contour we can use (3.52) to place the poles in a determined interval. Using a parameter p with value between zero and one, representing the minimum location of the last pole of the system, then fitting (3.52) between p and one, then:

$$\frac{(p-1)\gamma}{k_1 h^2 + k_2 h^4 + k_3 h} < \Delta t < \frac{\gamma}{k_1 h^2 + k_2 h^4 + k_3 h} \tag{3.53}$$

The relationship between contour evolution speed and its smoothness and stability is inverse. Meaning that to achieve higher speeds of evolution we must compromise smoothness and stability.

2.4 Re-dimensioning of the Model

From the previous sections to determine the behaviour of an active contours model we have to define the set of control parameters. If there is the need to have a model with a similar behaviour that is based on a different number of harmonics or number of contour points then we need to define the re-dimensioning relations that allow the control parameters to be change while keeping the force balances that define behaviour constant.

2.4.1 Re-dimensioning for Different Number of Contour Points

The system's behaviour will change with the number of points in a contour derived from changes in the internal forces potential energy, which is undesirable, since it forces us to define a different set of parameters for each value. This can be tackled by taking a reparameterization of the contour.

If as in [29] we admit the contour to be computationally defined by $C(u)$, where u the index of its position in an array of N_p points with $u = 1,2, \dots N_p + 1$, then applying and affine transformation as such as we get a contour defined by $C^*(\mu) = C^*(mu + b)$, defined by N_p^* points we can conclude that:

$$\frac{\partial}{\partial \mu} = \frac{1}{m}\frac{\partial}{\partial u} \quad d\mu = mdu \tag{3.54}$$

Where $m = N_p^*/N_p$. Inserting (52) into the internal potential energy definition (14) and forcing the balance to be the same in each case, we get that the relationship between reparametrized parameters (k_1^*,k_2^*) and a set of base parameters (k_1,k_2) has to be:

$$\begin{aligned} k_1^* &= \left(N_p^*/N_p\right) k_1 \\ k_2^* &= \left(N_p^*/N_p\right)^3 k_2 \end{aligned} \tag{3.55}$$

This possibility to reparametrize the models for different numbers of points opens the possibility to choose when to add or remove points from the contour for tracking of bigger or smaller structures, in which having a reduced and increased points is an advantage for capturing detail and for model stability.

2.4.2 Re-dimensioning for Different Number of Harmonics

As different levels of detail in the contour of an interest structure prompt higher number of harmonics in order to be described accurately, the active contour model needs to be able to preserve relationships between parameters in order to be fully useful. This means we need to establish transformations between parameters of from a model using h harmonics and another using h^*, while preserving the ratios derived from the balance between forces.

Starting with the internal force parameters if we take their definitions in (3.35), and force the ratios α and β to be invariant, we can conclude that the relationship between parameters must be defined as:

$$\begin{aligned} k_1^* &= R^{3/2} * k_1 \\ k_2^* &= R^{7/2} * k_2 \end{aligned} \tag{3.56}$$

Where $R = {h}/{h^*}$. Using the same approach for re-dimensioning the balloon force parameter, by inserting definitions (35) in (46) and keeping all ratios invariant, then:

$$k_3^* = R^{7/2}\left(\frac{1+\beta h^{*2}}{1+\beta h^2}\right) * k_3 \tag{3.57}$$

In many cases since β is preferably small, the second term of the product can be considered to equal one and even when approximation is not possible, it shows how parameter β traduces the balance in internal forces in the contour model. Finally, the last parameter to re-dimension is γ,which controls the viscous force, but more importantly, controls the sensibility of the model to external forces. With the same approach as before, keeping the chosen sensibility invariant, we get the condition:

$$\gamma^* = \gamma \frac{\|Q_{im}\|^*}{\|Q_{im}\|} = \gamma \sqrt{\frac{\sigma^{*2}_{gen}(v^* + \lambda^*)}{\sigma^2_{gen}(v+\lambda)}} \tag{3.58}$$

To prevent having to estimate the medium force over and over for different values of h^*we can however make some approximations to the previous condition. Assuming that both distributions are homoscedastic, then $\sigma^2_{gen} = \sigma^{*2}_{gen}$, and that the squared 2-norm follows the theoretical chi-square distribution then, $h = v + \lambda$, and thus we reach an approximated version of (56):

$$\gamma^* \approx R^{-1/2} * \gamma \tag{3.59}$$

Applying these re-dimensioning conditions does not guarantee that the model will behave the same for every harmonic introduced. What it guarantees is that the interval between the minimum and the maximum eigenvalue remains the same, redistributing the remaining in this interval. Since the minimum eigenvalue ($l = 1$) controls the global shape of the contour and the maximum eigenvalue ($l = h$) controls the local deformations of the contour, this redistribution allows a contour to keep its overall behaviour and previously defined force balances, independently of the number of harmonics.

2.5 Extraction of Geometrical Properties

After being able to define the dynamic model for active contours, the next step towards a visual servoing approach is to be able to extract geometrical properties from the defined contour model. One straightforward approach normally is to take the contour as described in a polygonal fashion (through its ordered vertexes) and compute their image moments. Image moments are mathematical condensed tools that allow the expression of geometrical properties of a region through the weighted sum of its

constituents. Then if we define a two-dimensional region, $D \in \mathbb{R}^2$, and a weighting function, $f: \mathbb{R}^2 \rightarrow \mathbb{R}$, the general definition of image moment is:

$$M_{p,q} = \iint f(x,y) x^p y^q dxdy \tag{3.60}$$

This weighting function, in most simple cases can be reduced to the binary representation of an interest region, or to the image intensity for general applications with grayscale images, however for specific applications this function assumes more complex formulations.

To reduce the computational cost for image moments in region formulation the original definition can be reorganized to work only with the boundary of the interest region. Assuming, as in [30], that the boundary is a parametrizable curve, define as $\partial D = C(u) = \big(x(u), y(u)\big),\ u \in [0\ L]$, then applying Green's theorem to (3.60) yields the definition of boundary contours:

$$M_{p,q}^B = \frac{1}{2}\int_0^L x(u)^p y(u)^p \left[\frac{x(u)y'(u)}{p+1} - \frac{y(u)x'(u)}{q+1}\right] du \tag{3.61}$$

Choosing the parametrization of ∂D to match with the parametrization for a contour $C(u)$ using Elliptic Fourier descriptors, defined in (3.22) with $L = 2\pi$, as done in [30], then we get a general formula to compute moments of order $p+q$, only using a set of descriptors, which is:

$$\begin{aligned} M_p^B = \frac{1}{4}\sum_{i=0}^{h}\sum_{j=0}^{h}\{&\alpha_{i,j,p,q}\big(m^c_{|i-j|,p,q} + m^c_{i+j,p,q}\big) \\ &+ \beta_{i,j,p,q}\big[m^s_{i+j,p,q} - \sigma(i-j)m^s_{|i-j|,p,q}\big] \\ &+ \gamma_{i,j,p,q}\big(m^c_{|i-j|,p,q} - m^c_{i+j,p,q}\big)\} \end{aligned} \tag{3.62}$$

Where $\alpha, \beta, \gamma_{i,j,p,q}$ are parameters derived from de combination of descriptors, fully defined in [30], and $m^c, m^s_{k,p,q}$ are the projection of the contours powers' on each of the models' harmonic basis functions, which can be defined as:

$$\begin{aligned} m^c_{k.p,q} &= \int_0^{2\pi} x(u)^p y(u)^q \cos(ku) du \\ m^s_{k.p,q} &= \int_0^{2\pi} x(u)^p y(u)^q \sin(ku) du \end{aligned} \tag{3.63}$$

To compute each of these integrals the integration by parts rule is applied and as show in [30], this computation yields expressions containing terms equal to (3.62) that are dependent on the moments of order $p+q-1$. This means that for computing the moments of higher order we must compute all lower order moments, meaning a recursive computation scheme can be applied, as introduced in [30], to further reduce computation time.

These computed moments, however useful they may be, do not immediately traduce into geometrical information with specific meaning, needing to be further refined for this information to be extracted.

Meaningful geometrical information is the information we can extract to characterize the location, orientation and features of an interest structure present the image space or that allow a parallel comparison between this structure and a well-known geometrical object. As was done in [22], [31] we can define sets of parameters that derive from the moments of an interest structure.

The first and most simple parameter is the area of the structure itself, which is simply identified as the zeroth order moment:

$$a = M^B_{0,0} \tag{3.64}$$

However useful this parameter is, it offers no information about the location of the structure.

Classical location and orientation parameters such as the coordinates of the centre of mass (CM) and principal orientation (θ) can be extracted from the centred form of image moments ($\mu_{m,n}$), which can be obtained from raw moments ($M^B_{p,q}$) with:

$$\mu_{p,q} = \sum_{i}^{p}\sum_{j}^{q}\binom{p}{i}\binom{q}{j}(-\bar{x})^{p-i}(-\bar{y})^{q-j}M^B_{i,j} \tag{3.65}$$

Where $\bar{x} = M^B_{1,0}/M^B_{0,0}$ and $\bar{y} = M^B_{0,1}/M^B_{0,0}$, which on their own define the CM of the interest structure. From higher order moments, specifically 2nd order centred moments we can also compute the principal orientation angle, with:

$$\theta = \frac{1}{2}\arctan\left(\frac{2\mu_{1,1}}{\mu^2_{2,0} - \mu^2_{0,2}}\right) \tag{3.66}$$

Where θ defines the angular deviation of the structures' principal axis from the horizontal axis of the image.

As shown in [31], by matching a known geometric object model, raw and centred moments can even be used to compute simple geometrical primitives for ellipses through computation of the major and minor axes and ellipse eccentricity, from the 2nd order moments, with:

$$\begin{aligned} l_1 &= \sqrt{2\left(\mu_{2,0} + \mu_{0,2} + \sqrt{\left(\mu_{2,0} - \mu_{0,2}\right)^2 + 4\mu^2_{1,1}}\right)} \\ l_2 &= \sqrt{2\left(\mu_{2,0} + \mu_{0,2} - \sqrt{\left(\mu_{2,0} - \mu_{0,2}\right)^2 + 4\mu^2_{1,1}}\right)} \end{aligned} \tag{3.67}$$

Going even further as in [11], sets of more complex moments such as Hu's moment invariants can be computed from base moments, which can used as out-of-plane motion quantifiers.

Ultimately, the choice of which parameters to compute in a visual servoing approach are dependent on the task the robotic system is charged to perform. The set of control parameters should also be chosen

as such each parameters' variation offers good decoupling properties regarding the 6DOF for motion on 3D space, translations in (x, y, z) and the respective rotations about these axes.

CHAPTER 3 - IMPLEMENTATION IN MATLAB

With all theoretical concepts defined and ready, the goal of this chapter is to describe the implementation phase of the algorithm in the MATLAB 2015a platform. It will start with the initial context of program setup and move on to a more detailed description of the different steps the program must cover in order to be an effective and accurate tracking tool. As well as a formal description, this chapter also aims for a schematic description approach so as the program flow is more easily understandable.

Beginning with the overarching program, it works in conjunction with the Image Acquisition Toolbox existent in MATLAB allowing the acquisition of the ultrasound image from a Pinnacle Dazzle frame grabber. It captures an 8-bit 640 by 460 image at a rate of 25 fps, which will serve as the input for its next stage, in the implementation of the real-time tracking algorithm. The flow of the overarching program can be summarized by the scheme in Figure 3-1.

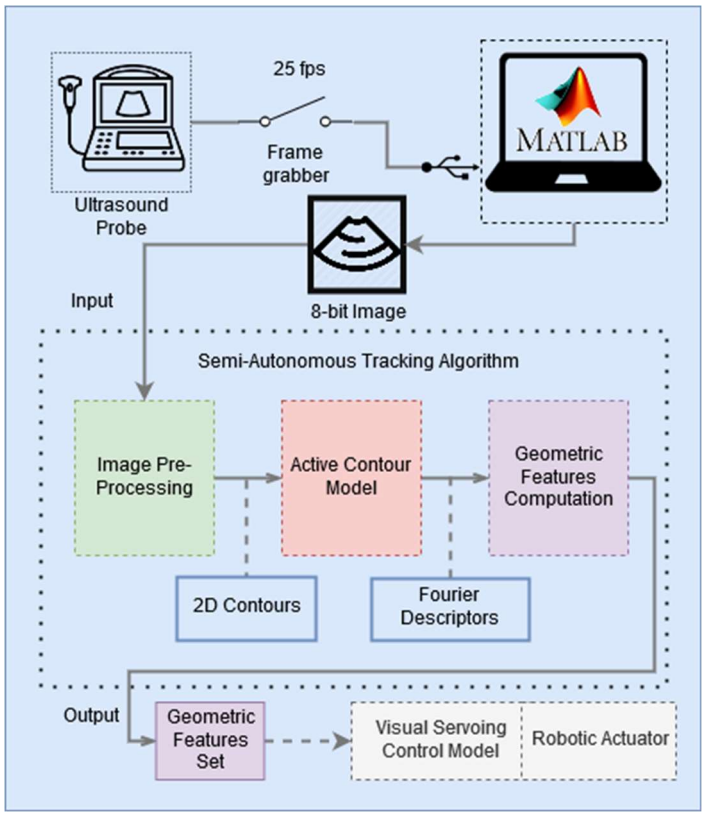

Figure 3-1: Scheme representing the experimental setup coupled with the tracking algorithm

In terms of organization the flow of the implemented tracking algorithm can be divided into three main steps: an image pre-processing, an active contour model evolution step, and finally a geometric feature computation step, which will output geometric features that naturally might be used in a future visual servoing assisted or controlled task.

The first step deals with the denoising and smoothening of the acquired image, the computation of image gradients and their derivatives and also, in the first iteration of the algorithm, the contour initialization required in the next step. The second step takes the contour defined in the previous step and the image potentials and iterate the model of active contours until it reaches a stable state, meaning that it converges into a selected structure. Finally, the last step, takes the contour in its descriptor form and compute the geometrical features of interest to be used in visual servoing.

Going more into detail on the tracking program, it has an initialization phase and a running phase. In the first one all supporting structures, either for the image processing step or the active contour model step are computed. For the image processing step, the filtering kernels for gaussian smoothing (with $\sigma = 6$ and size $3\sigma - 1$) and for image speckle filtering (circular kernel of radius 8) are defined using the MATLAB functions FSPECIAL and STREL.

The gaussian kernel is defined by the parameter σ , the standard deviation, and the parameter size = $3\sigma - 1$, the kernel size enabling the central representation of the gaussian distribution for up to 99% of its content, shown in Figure 3-2(a).The speckle kernel is defined as pixel neighbourhood with a circular pattern and a radius of 8 pixel, as shown in Figure 3-2(b). These parameters were chosen in order to provide ultrasound images that have both smoothened gradients, to prevent an active contour with oscillating behaviour around image edges, and a lower level of speckle noise, to make image regions more uniform, as shown in Figure 3-2(c).

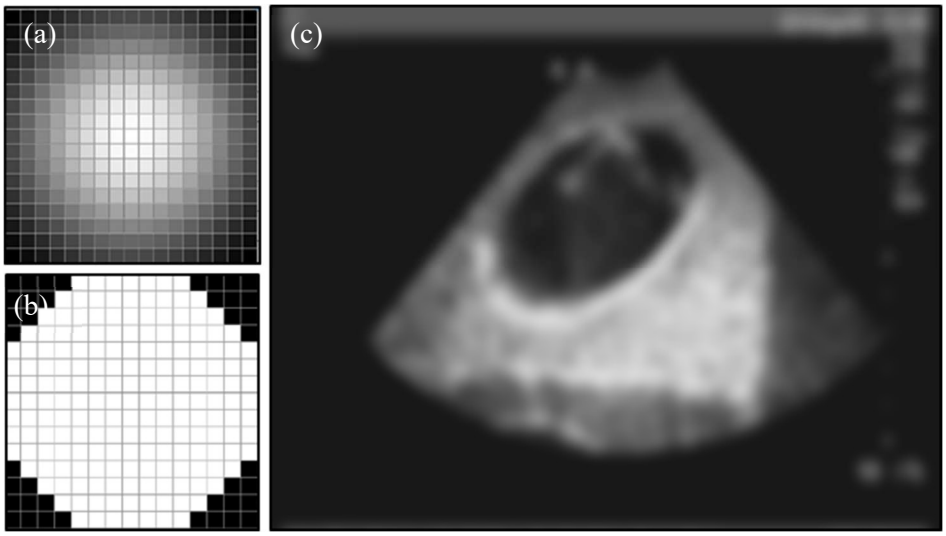

Figure 3-2: Image processing kernels: Gaussian kernel (a) and Circular kernel (b), along with their result on a sample image (c).

For the active contour model step the support structures that are computed are the behaviour control matrices defined in (3.24) and the contour derivatives relative to the descriptor vector. This contour derivative matrix, as defined in [27], acts as a mapping matrix between the image space and the descriptor space allowing the computation of external forces acting in the descriptor space.

The following phase of the algorithm is the running phase. In this phase the program analyses one frame at a time, moving through the previously established steps, as can be show in the scheme from Figure 3-3.

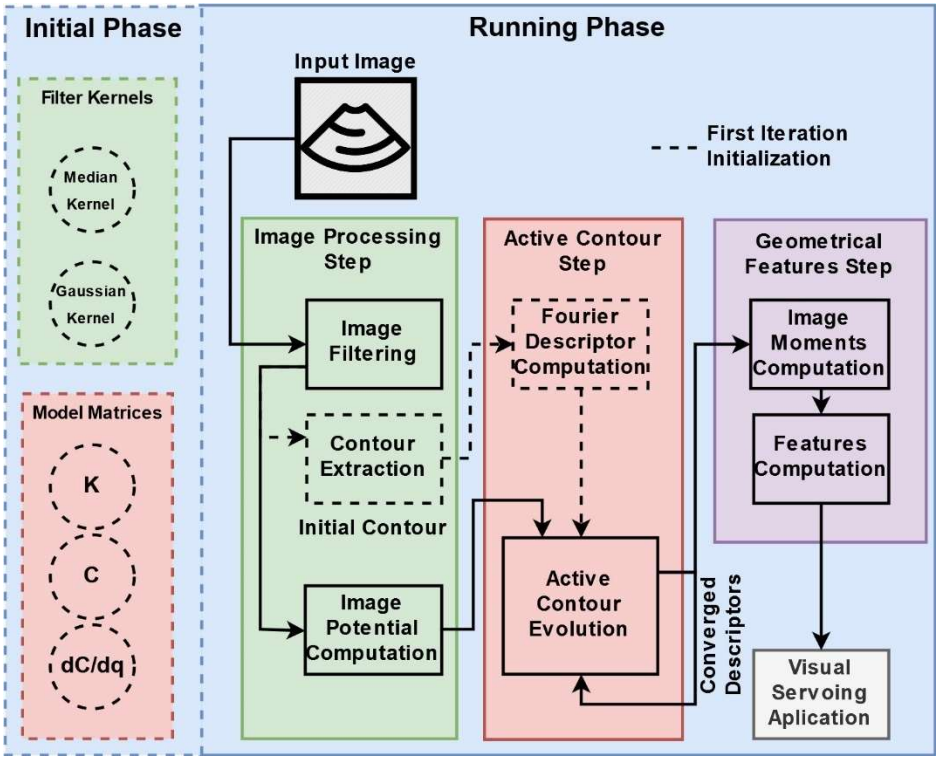

Figure 3-3: Scheme representing the overall organization of the tracking program.

Additionally, on its first run of the running phase, the program acquires the first frame and pre-processes it according to the scheme, however, since the active contour step requires a contour initialization, the image processing step is extended with a contour extraction phase, providing this step with a set of points that will be decomposed into Fourier descriptors, thus building the initial state vector for the discrete state system defined in (3.27).

Furthering the description of the sequence of the algorithm we will go into details on to how the various steps of the algorithm are built, starting with the image processing step. This step has a singular flow chain, to work for the first iteration and a main flow chain, to work for every iteration (including the first), as is described in the scheme from Figure 3-4.

The singular flow for frame number one concerns the extraction of a contour from the image to be used as initialization for the active contour model, which can be done in two ways, through manual selection, or a combination of image segmentation, morphology processing and a contour extraction algorithm. The first, the manual approach, takes use of MATLAB's figure handle call-back functions, allowing the recording of the mouse cursor position on the image, providing a user specified contour, which is delivered into an array of 2D points. The second approach, the semi-automatic, relies on image segmentation, through Otsu's method, described in [32], to separate background from foreground in an image, followed by a morphology processing step, through opening and dilation operations using a circular neighbourhood to obtain pixel regions with no holes, and finally a contour extraction step, using Moore-Neighbour's tracing algorithm, to extract the contour coordinates for the pixel regions' outer perimeter. In case this step extracts more than one contour, the correct one can be chosen by the user, through call-back functions in the figure handle class.

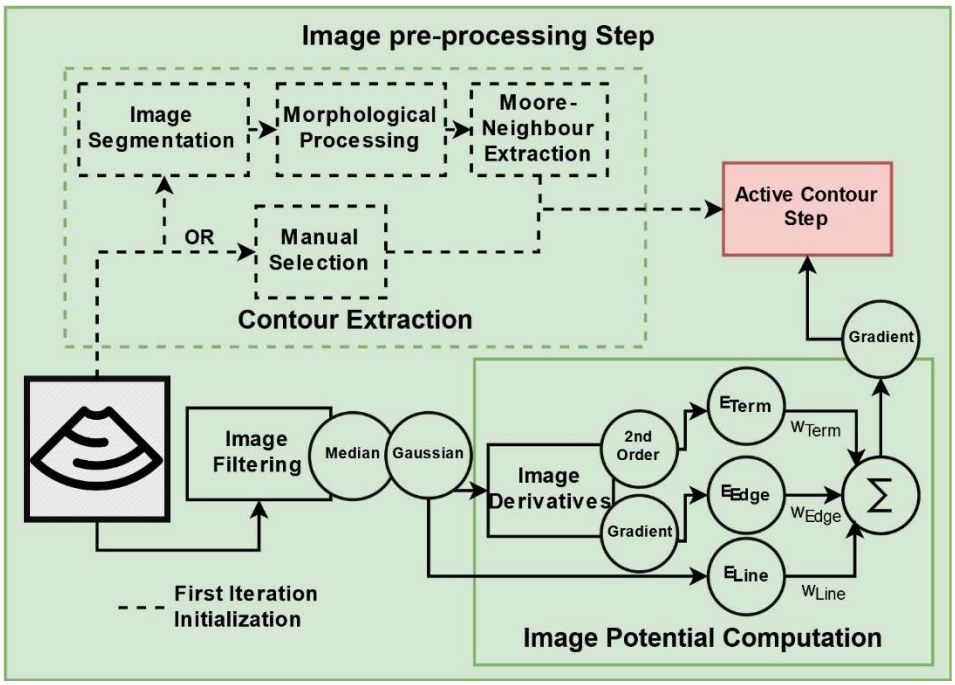

Figure 3-4: Scheme representing the organization of the image pre-processing step.

The main flow, for every frame, concerns the denoising and smoothening of image edges along with the computation of the image potential that will drive the external forces applied on the contour that will make it converge on a structure. Since this task can take a decent amount of time to compute, it was decided to take advantage of MATLAB's capabilities of GPU computation.

As such after its acquisition, the image is moved into a GPU Array and then is convoluted with the image filtering kernels previously mentioned (with appropriate padding so to preserve its size) and in sequence, twice convoluted with vertical and horizontal Sobel kernels, to compute the image's first order derivatives (I_x and I_y) and second order derivatives (I_{xx} , I_{yy} and I_{xy}). The different feature potentials are computed from these derivatives, according to their definitions in (3.29,3.30,3.31) and the global potential computed as the weighted sum of all feature potentials. Finally, the global potential is again once differentiated in in the same manner as before, to compute the horizontal and vertical components of the external force applied at each pixel of the image.

Moving into the active contour model step, it can be shown by the scheme in Figure 3-5 that it will take a non-linear flow. It will function in an iterative cycle within the same image frame until a satisfactory convergence is reached. On its first iteration the program will take the initialization contour and decompose it into Fourier descriptors, which is done by taking its DFT, through MATLAB's FFT function, of each for each of the contours' components (x and y) and truncating it for the number of desired harmonics (h), yielding the initial state vector for the model.

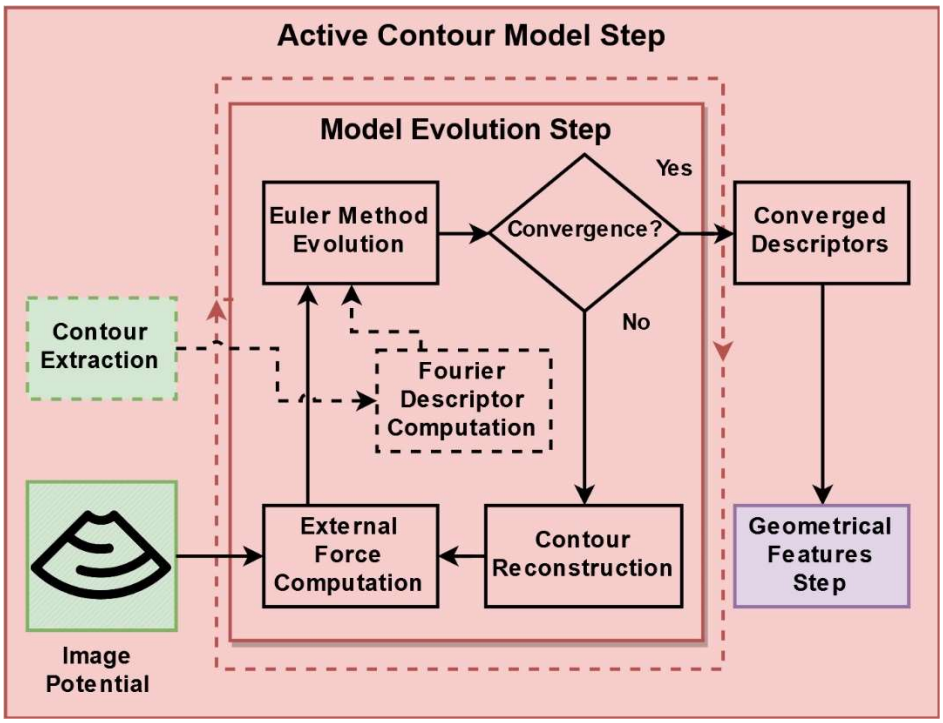

Figure 3-5: Scheme representing the organization of the active contour model step.

Then with the initial contour and the image potential the external force applied to each contour point in descriptor space are computed, using the discretized a version of expression (11):

$$Q_{im}(q) = \sum_{n=1}^{N_p} [-\nabla E_m(n)] * \frac{\partial C}{\partial q}(n) \tag{4.1}$$

Where $\partial C/\partial q\,(n)$ is the derivate of the contour description in relation to the descriptor vector at each point of the contour and $\nabla E_m(n)$ the computed image potential at each point of the contour. This external force and the initialized descriptor vector are then used in the evolution of the model, through the iterative expression that defined the discrete model of the Fourier active contour obtained assuming a Euler explicit method, defined in (3.26) and assuming also no inertial effects on the contour, meaning that since $M = [0]$ the second order derivative effects are nullified.

The outputted evolved set of descriptors is then checked for convergence. The convergence criterion used is a measure of displacement of the contour points between the evolved and the previous contours. According to [26], it can be defined as the norm of the difference between the two sets of descriptors:

$$\|q_{n+1} - q_n\| < \epsilon \tag{4.2}$$

However, with this definition the convergence criterion ϵ has does not have a physical meaning in the context of the image. As such, we compute the IDFT of the difference and divide by N_p, returning the descriptor vector back into point by point description. This yields a measure of displacement point-by-point and taking its mean we get a global convergence criterion in which the criterion has meaning.

$$\text{mean}\left(IDFT\left(q_{n+1}-q_n, N_p\right)\right) < \epsilon \tag{4.3}$$

If the measure is lower that the criteria, then the contour is converged, the cycle brakes and the program will continue to run to the next step, otherwise the evolved descriptors are reconstructed into an evolved contour, and the cycle begins a new with external force computation, within the same frame.

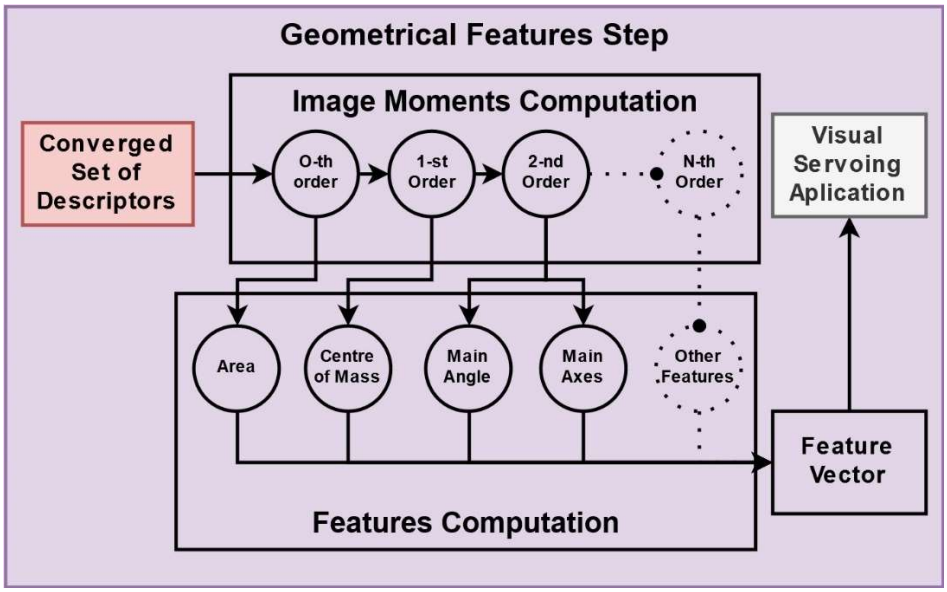

Figure 3-6: Scheme representing the organization of the geometrical features step.

The last step is the computation of geometrical features from the set of converged Fourier descriptors. This step takes use of the recursive computation for the different orders of moments, mentioned before, and their definition in (3.61). The algorithm always starts with the zeroth order moment and usually ends at the second order moments (although it can compute until any desired N^{th} order moment). From the computed moments the contours' geometrical features are computed according to expressions (3.62) -(3.64). The main set of features computed in this approach are the contour enclosed area, the centre of mass and main orientation, however as additional features, the eccentricity and main axes can be computed from the 2^{nd} order moments with (3.65). As shown in Figure 3-6 the geometrical features are then organized into a feature vector that may be used in the future in a visual servoing approach.

CHAPTER 4 - EXPERIMENTAL SETUP AND RESULTS

After achieving an efficient implementation for the tracking program, the next step is to test its tracking capabilities, its robustness to poor image quality and structure movement and its precision in the computation of a target's geometrical features. For this purpose, we cover the process of the image calibration, through the implementation of a pixel spacing estimation method using a N-wire calibration phantom. In a second it will cover the actual testing of the tracking algorithm, where a tissue emulating phantom is used to produce artificial structures.

In both cases, the experimental setup includes an ultrasound image acquisition SonoSite TITAN and a probe coupled with a laptop with an Intel Core i7-7700HQ @ 2.80Gz CPU and a NVIDIA GeForce GTX 1060 GPU running the MATLAB 2015a platform, on which the image processing, active contour tracking and feature computation is set to run. The image acquisition system outputs an image stream at 25 fps which is captured from the S-Video output source and delivered through an USB frame grabber. The interface between the USB grabber and MATLAB is established through the video capture objects from the Image Acquisition Toolbox.

4.1 Image Calibration

Since the ultrasound imaging device is designed to work with a closed acquisition system, the calibration information present in the demanded DICOM tags are not accessible unless an acquisition server is setup, which is a non-available option. As such in this first phase, the available option is to use a specifically designed calibration phantom to insert a specific set of image features that allow the measurement of calibration information in an indirect manner.

4.1.1 Calibration Phantom Architecture

According to [33], there are several phantom designs that can be used to accurately extract calibration information from an ultrasound image. Usually the calibration aims to establish the transformation matrix that enables the location of an image in 3D space, when coupled to a robotic arm or a localization device. However, in this case, the only parameters needed are the pixel spacings in the horizontal and vertical directions. Thinking of future application in full calibration the phantom design followed is described as a N-Wire phantom, as shown in Figure 4-1(a). It consists in two 10 mm thick, 150 mm by 200 mm parallel sheets of acrylic anchored in a base piece designed to keep them 60 mm apart. The sheets are drilled with a specific two-fold 5 mm hole pattern starting at the centre of each sheet and spanning and 75x120 mm area. The holes in the pattern are spaced 10 mm in the vertical direction and 15 mm in the horizontal direction, which allow the creation of a multitude of image recognizable feature patterns. The phantom is submerged in a bath of distilled water, which will act as a background medium for ultrasound propagation.

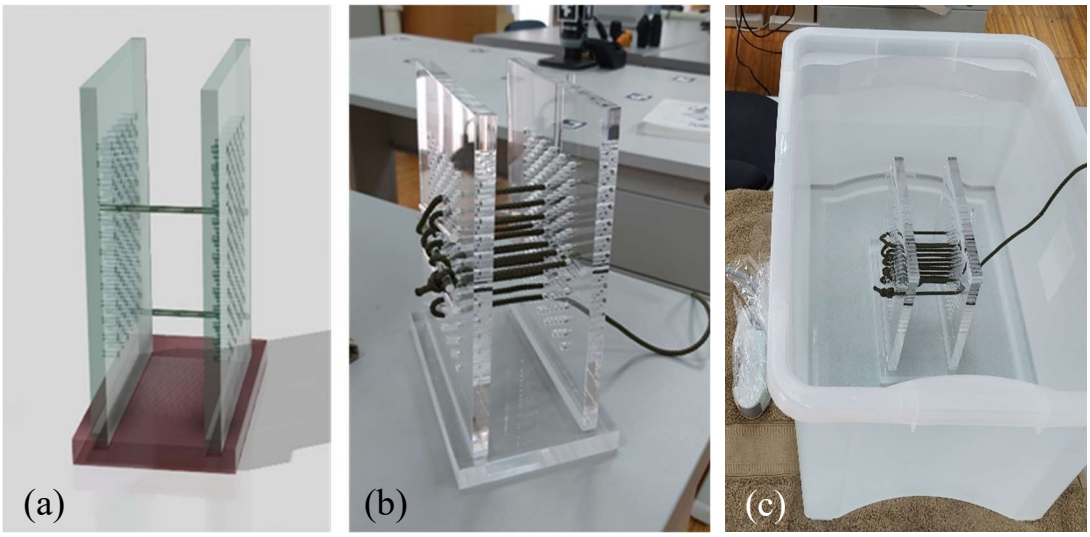

Figure 4-1: Representation of the N-Wire phantom used in calibration: (a) - CAD model; (b) - the phantom weaved in a cross shaped pattern; (c) - the experimental setup for the calibration procedure.

In order to establish patterns that can appear in ultrasound imaging the weaving material placed in between holes needs to be significantly denser than water, to create regular hyperechogenic spots on the image plane. This is achieved by weaving 5 mm thick nylon thread through each hole in the most appropriate pattern, as shown in Figure 4-1(b). These nylon threads are held in place and at appropriate tension using a set of 5 mms thick and 10 mm long cylindrical acrylic rods, that help to assure the nylon threads are kept as straight as possible.

4.1.2 Feature Acquisition Procedure

The acquisition of calibration images done with the setup from Figure 4-1(c) was made in three stages, consistent with the three feature patterns implemented on the phantom: a horizontal line pattern, a vertical line pattern and a diagonal line pattern. For each pattern three penetration depth configurations of the probe: 7.5, 10 and 13 cm were sequentially set. The patterns used in this approach are formed by interweaving the nylon thread with the holes in the phantom walls have the following specifications:

- Horizontal Pattern – The nylon thread is weaved one hole at a time an in the central line of the horizontal direction of the calibration phantom. The horizontal real world spacing is desired to be 15 mm and the vertical close to zero, as shown in Figure 4-2(a);
- Vertical Pattern - The nylon thread is weaved one hole at a time an in the central line of the vertical direction of the calibration phantom. The vertical real world spacing is desired to be 10 mm and the horizontal close to zero, as shown in Figure 4-2(b);
- Diagonal Pattern – The nylon thread is weaved one hole at a time, one in the vertical direction and one in the horizontal as such that the pattern divides the calibration phantom diagonally in two equal parts. The vertical real world spacing is desired to be 5 mm and the horizontal 7.5 mm as such that the pattern originates a line segment with a slope of 33.6 degrees, as shown in Figure 4-2(c).

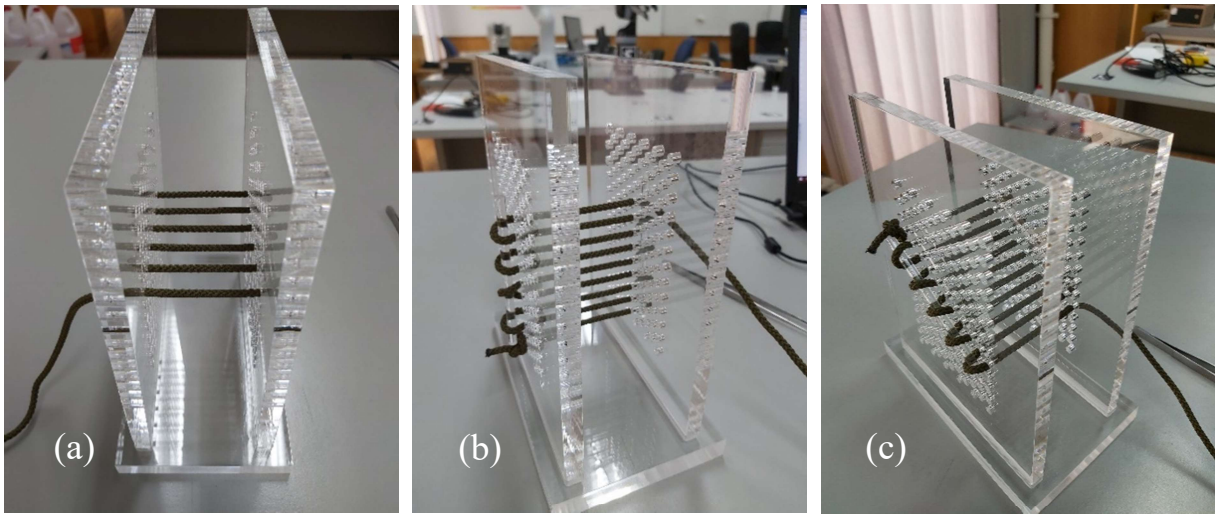

Figure 4-2: Patterns implemented on the N-Wire phantom for calibration: (a) - Horizontal; (b)-Vertical; (c)- Diagonal.

For each combination of pattern and depth the acquisition procedure was kept the same: the ultrasound probe was placed on the midpoint between the two sheets of acrylic, with the image plane perpendicular to the nylon thread pattern and the probe at a vertical configuration. Controlling the mode and gains of the acquisition device, it was possible to acquire images as Figure 4-3 where the only apparent features in the image plane are the cross-sections of the nylon thread, appearing at an even spacings.

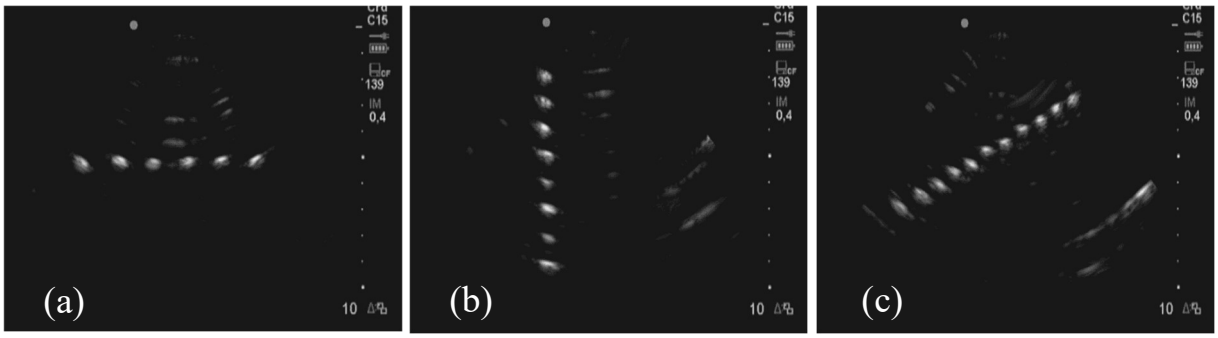

Figure 4-3: Acquired images for a vertical (a), horizontal (b) and a diagonal pattern (c) on the calibration phantom.

After the images were acquired, they underwent processing to extract the centres of mass of each of the hyperechogenic marker features, through a processing resembling the contour extraction step from the initialization of the tracking algorithm.

4.1.3 Pixel Spacing Computation

Once the features' centres of mass are computed in pixels these are arranged into a vector form and before computing the distance between centres this vector is sorted in such way that the distance between centres of mass is minimalized. This important step helps to guarantee that in an unsupervised calibration, the extracted centres that conform into a specified pattern are all grouped as closely as possible within the centre vector. Then the consecutive pixel distances are computed taking de difference between centres. Furthermore, the resulting centre distance vector is sorted to eliminate outlier distances that might be related to erroneous extraction of features, leaving only the vertical and horizontal distances that are compatible with the pattern implemented on the calibration phantom. The processed feature vector is then averaged, and its standard deviation is also computed. Having the average

horizontal and vertical distances for each of the depth configurations, then the vertical and horizontal spacings in the image can be computed with:

$$s_{X|Y} = \frac{d_{Rx|Ry}}{P_{dx|dy}} \tag{5.1}$$

Where $d_{Rx|y}$ is the real world vertical and horizontal spacings of the calibration phantom's hole pattern in millimetres and $P_{dx|y}$ is the measured average vertical and horizontal spacings in pixels. The error associated with the estimation of $s_{x|y}$ is also computed, through error propagation using:

$$\sigma_{S_{X|Y}} = \sqrt{\frac{1}{P_{dx|dy}^2} * \sigma_{d_{Rx|Ry}}^2 + \frac{d_{Rx|Ry}^2}{P_{dx|dy}^4} * \sigma_{P_{dx|dy}}^2} \tag{5.2}$$

Where $\sigma_{d_{Rx|Ry}}$ is the typical tolerance in millimetres for a CNC milling process in acrylic or plexiglass according to the ISO 2768 directive, that regulates the general tolerances for CNC milling processes, and whose value is assumed to be 0.1 mm for medium precision procedures. And $\sigma_{P_{dx|dy}}$ is the standard deviation for the measured spacing in the image in pixels. As such the measurements of the pixel spacings for each depth in millimetres per pixel are presented on Table 4-1.

Table 4-1: Results for the image calibration routine in mm/px.

Depth of Focus	Horizontal Pattern		Vertical Pattern		Diagonal Pattern	
	s_X (mm/px)	s_Y (mm/px)	s_X (mm/px)	s_Y (mm/px)	s_X (mm/px)	s_Y (mm/px)
7.5 cm	0.24	-	-	0.23	0.25	0.24
	± 0.01			± 0.01	± 0.02	± 0.05
10 cm	0.31	-	-	0.29	0.30	0.36
	± 0.02			± 0.04	± 0.02	± 0.08
13 cm	0.39	-	-	0.38	0.39	0.37
	± 0.04			± 0.02	± 0.05	± 0.08

Once it is desirable that error in the estimation of the real world spacing is minimum the adopted values for these parameters are the ones extracted from the simple vertical and horizontal patterns.

4.2 Tracking Algorithm Testing

In this phase, to validate the proof of concept of the active contour-driven tracking algorithm the program had to undergo testing. Due to the premature state of development of the program and as it would be time consuming, the testing routines were not performed in human subjects. However, thorough testing was done using a handmade ultrasound phantom, specifically designed to provide the best contrast possible between a simulated background tissue medium and a given target structure.

4.2.1 Ultrasound Phantom

Ultrasound phantoms are a simpler, cost efficient and less time-consuming approach when it comes to testing imaging related algorithms as well as enabling to setup specific situations to which the algorithm itself must conform and perform at peak efficiency. For these cases, in ultrasound imaging, one of the most important features in building accurate ultrasound phantoms, other than matching the speed of sound in soft or muscle tissue (1540 m/s), is to be able to reproduce the degree of ultrasound speckle noise. To accomplish this objective the approach taken was the one described [34], where a phantom was produced with a mixture of 5% (m/m) of Agar-Agar powder.

The process begins with measuring 1.25 L of water into a recipient and adding a measured 62.5 g of Agar-Agar powder and mixing the mixture until all agglomerations are dissolved. In this case, since the powder used was a coarser kind, the mixture was also liquefied using a blender, until it was homogeneous. Then the mixture was gradually heated up to a light boil and continuously mixed to form a uniform paste. At that state, approximately 4 g of flour were added to the mixture and it was stirred until it was homogeneous again. The flour is added to act as a suspended infra-resolution colloidal diffractor of ultrasound waves and emulates the introduction of speckle noise often found on soft tissue. The structures of interest, in this case, will be represented by water filled balloons, which due to their highly hypoechogenic nature will form darkened shapes in the image, similar to how cysts are often represented.

To complete the process a first layer of mixture is poured into a recipient to a height of approximately 5 cm and is allowed to set at room temperature for 20 minutes until the top layer is beginning to gelify, while keeping the rest of the mixture mildly heated and properly stirred. Then two filled water balloons of different sizes are set on top of the layer positioned in a diagonally opposed manner, which then are covered until half their height with mixture and covered with a thin layer of mixture. The setup is left to gelify for 10 minutes, while always keeping attention on the remaining mixture. Finally an elongated water balloon filled to a length of 14 cm is positioned on the diagonal and applying force submerged in the gelifying mixture until it lies diagonally tilted in the vertical direction, the remainder of the mixture is poured until it covers the balloon and is set to a final gelification period until it reaches room temperature.

4.2.2 Algorithm Performance Testing

The performance of the tracking algorithm is tested in both a qualitative and quantitative manner using the experimental setup shown in Figure 4-4. The qualitative approach is focused on the various kinds of motion and deformation that the probe movement can induce in the structures present in the image. The quantitative approach is focused on the computational time that is consumed by the algorithm while tracking a structure in the image, with increasing details, which is inherently related with the number of harmonics and contour points that describe the active contour model.

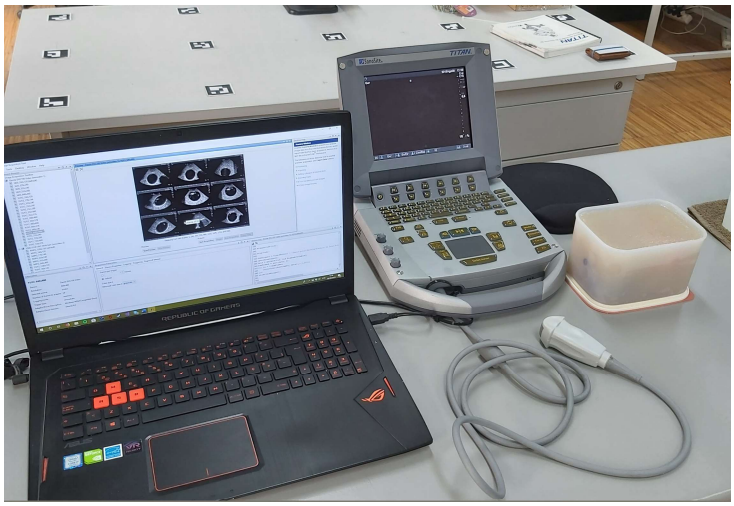

Figure 4-4: The experimental setup used in the program performance testing routine.

4.2.2.1 Contour Parameter Choice

Before going further into testing there is the need to define the set of parameters that will act as the base configuration for the active contour model tracking that is implemented. From experimentation and observation of the algorithm's behaviour, the ratio configuration that best serves the purpose of the tracking algorithm and that serves as base is expressed in Table 4-2. The choice of these parameters was made iteratively by experimenting on still images of known features until correct convergence was obtained. This choice is important to avoid unwanted contour behaviours shown in **Figure 4-5** such as convergence to false minimums [Figure 4-5(b)], over sensitivity [Figure 4-5(a)] and finally contour instability [Figure 4-5(c)].

Table 4-2: Set of parameters that define the desired behaviour for the contour model.

Parameter	*Value*
h	6
α	0.0027
β	0.4020
S	0.0309
ρ	2.3693
p	0.5000
$\|\|Q_{im}\|\|_m$	56.0886

With these ratios defined, the active contour model parameters are computed from (3.35), (3.45) and (3.48), which for the base configuration are displayed on Table 4-3.

Table 4-3: Set of parameters for the base active contour model.

Parameter	*Value*
h	6
k_1	0.0107
k_2	5.4068E-4
k_3	-0.0265
γ	624.0791
Δt	46.6747

The subsequent configurations for higher or lower harmonics are defined with respect to this base configuration through the re-dimensioning relations established earlier in Chapter 3.

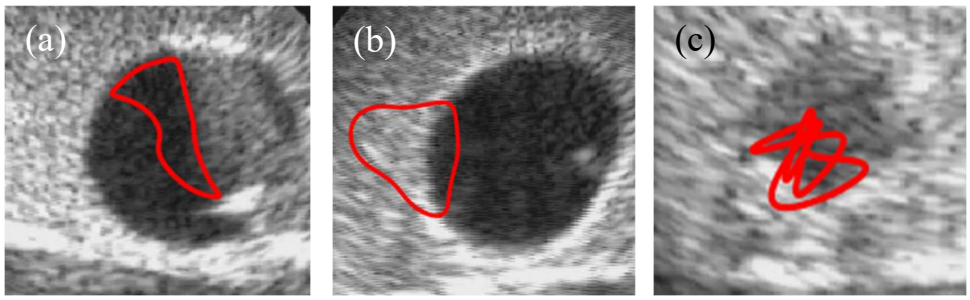

Figure 4-5: Unwanted behaviour for contour evolution: (a) – Over sensitivity to image potentials; (b) – Convergence to a false image potential minimum; (c) – Unstable contour evolution.

4.2.2.2 Performance Testing Routine

To achieve comparable qualitative results for active contour models with different levels of detail, a standard testing routine is defined. The idea with the tracking test routine is to generate the all the types of motion that can affect the contour tracking algorithm: translations in the image planes, in-plane rotation (perpendicular to the image plane) and extensions an contractions in one axis and two axes, which are consequence of out-of-plane probe motions.

As such, the performance testing routine starts by placing the ultrasound probe in the centre of the ultrasound phantom, with the image plane aligned and centred with the elongated balloon's cross section, for which the desired initial position is the similar to Figure 4-6(b).

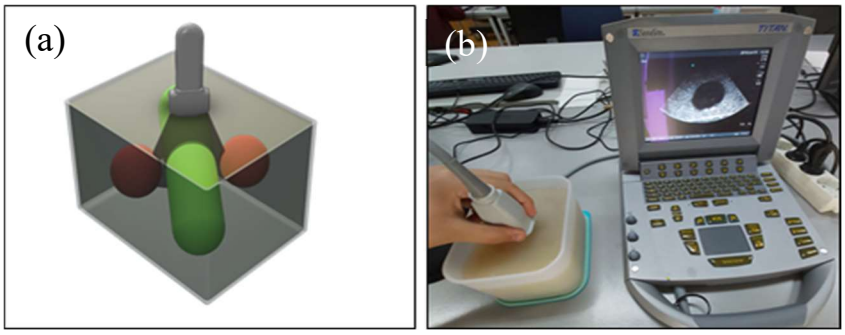

Figure 4-6: Representation of the initial position in the performance testing routine , as shown in a CAD model (a) and in the real experimental setup (b).

After the first placement the image acquisition is initialized as in Figure 4-6(a), as well as the initial contour for the tracking program. Then the probe is moved back and forth along the axis of the elongated balloon without rotating the probe as in Figure 4-7(a), which causes the tracked section to move vertically in the image plane (a) completing the first motion. The second motion is acquired after centring the probe and then moving the probe perpendicular to the balloon's axis left and right as in Figure 4-7(b), which causes the tracked section to move horizontally in the image plane. The last in plane motion is acquired rotating the probe laterally on the direction normal to the image plane as in Figure 4-7(c).

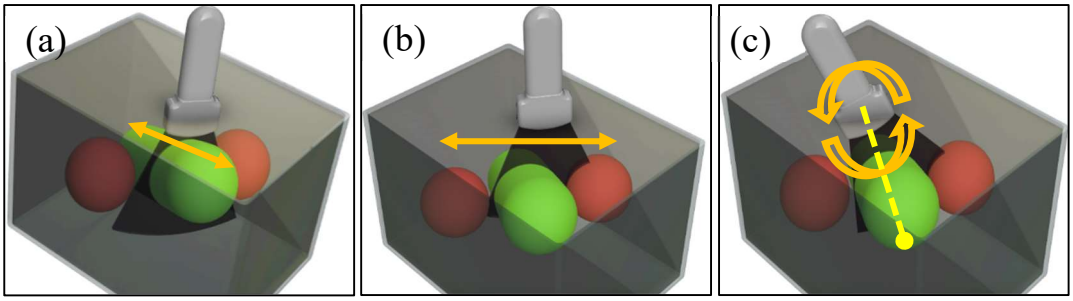

Figure 4-7: Representation of the in-plane motions induced in the performance routine, the vertical translation (a), the horizontal translation (b) and the in-plane tilting (c).

The elongation on one axis is achieved by placing the probe on the centre of the balloon and rotating the probe along its own axis as in Figure 4-8(a), which makes the image plane section the balloon at a different angle and produce elliptical sections, along a diagonal direction on the image, enabling the contour to stretch and contract along one direction. The last motion is the two-direction contraction and it can be acquired by centring the probe on the spherical balloons as in Figure 4-8(b). The probe is moved in the direction perpendicular to the image plane, which results in circular sections of varying radius, where the contraction of its contour is uniform in the vertical and horizontal directions.

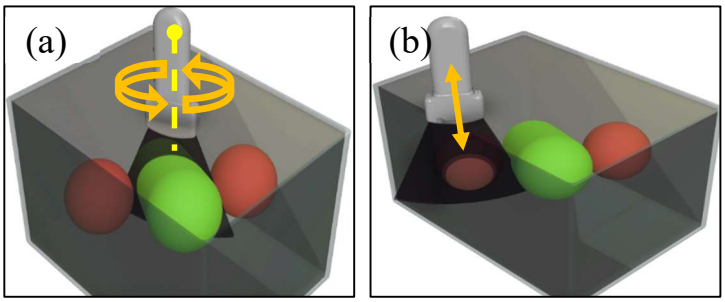

Figure 4-8:Representation of the out-of-plane motions induced in the performance routine, rotation on the probe axis resulting in contour deformation (a), out-of-plane translation resulting in contour contraction (b).

With this routine, not only we get the full range of motions to which the contour tracking algorithm will be usually subjected but also due to the proximity between structures, have an idea into how well the contours are attracted and converge to edges and other image features in the presence of other potential candidates to a valid tracking.

4.2.2.3 Performance Results

The previous routine was applied to three image acquisition procedures using three different sets of active contour control parameters: one set for the base configuration, one with less detail than the base configuration ($h = 3$) and one for bigger detail than the base configuration ($h = 9$). These acquisitions were made for a constant number of contour points ($N_p = 256$) and with an ultrasound focus depth configuration of 10 cm, for better visibility of the phantom's structures, for which the pixel spacing parameters used are chosen according to Table 4-1.

In order to visualise the results of the tracking algorithm, at each frame the converged contours in their polygonal form are overlaid in the acquired. Also, this polygonal plot is complemented with a frame of reference and the its position and orientation features. The numerical values of these features and current frame rate of the algorithm are presented in text at the side of the contour's workspace, as shown in Figure 4-9.

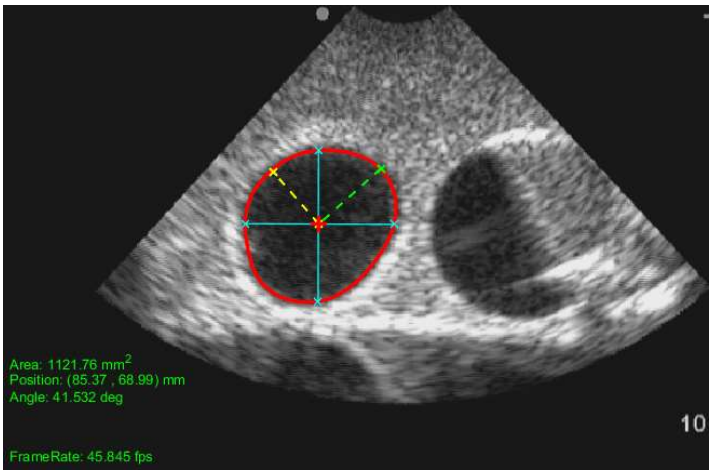

Figure 4-9: Visual representation of the tracking algorithm's GUI.

For the qualitative approach, in all the acquisition routines the behaviour of the active contours tracking program was captured for each moment coincident with the desired motion and is presented on Figure 4-10. The captures from Figure 4-10(a)-(c) show the response of the algorithm induced with an in-plane translation for the different applied models, the captures in Figure 4-10(d)-(f) , show the response of the algorithm induced by the out-of-plane translation of the probe that result in an in-plane vertical translation of the contour in the image plane. Finally, the captures in Figure 4-10(g)-(i) show the response of the algorithm to in-plane rotation induced by a tilting motion of the US probe on the image plane.

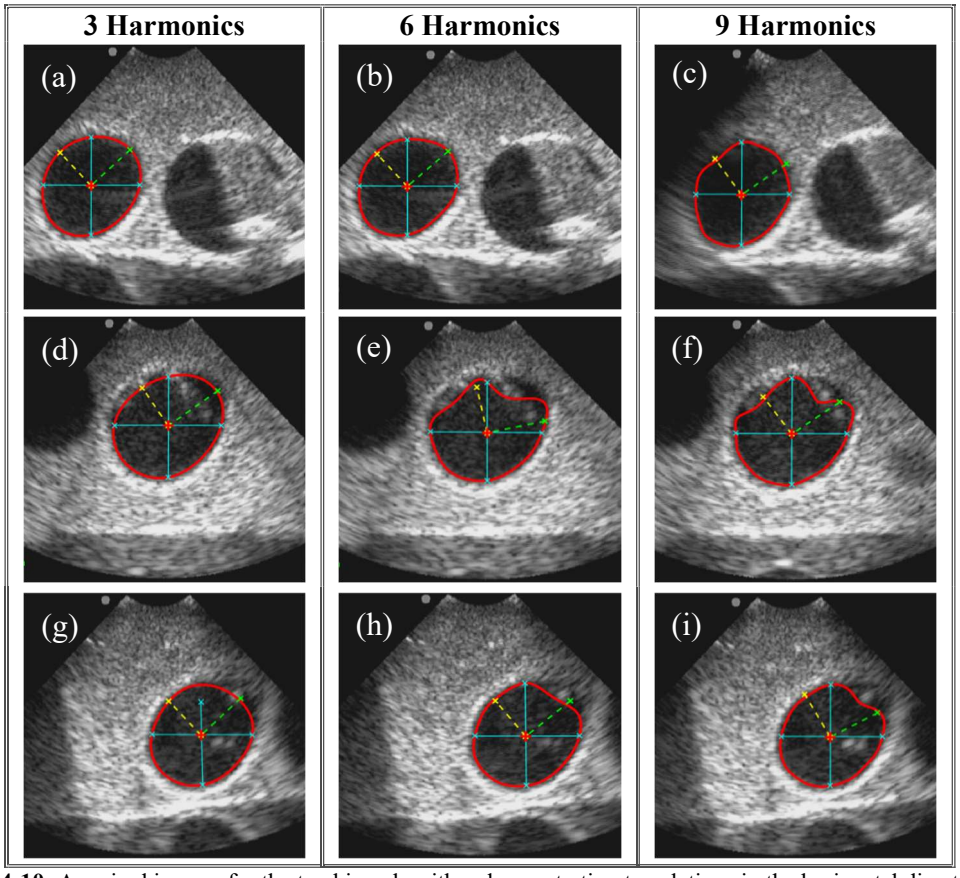

Figure 4-10: Acquired images for the tracking algorithm demonstrating translations in the horizontal direction (a)-(c), translations in the vertical direction(d)-(f) and in-plane tilting (g)-(i) for contour models with 3,6 and 9 harmonics.

The contours adaptation to out-of-plane translation and rotations motions leading to contraction and extension can also be observed. In Figure 4-11(a)-(c), are presented the deformations of the contour due to out-of-plane rotation resulting in the dilation of the contour to an ellipsoid shape. In Figure 4-11(d)-(f) are represented the deformations of a contour due to an out-of-plane translation, showing the ability for the contour to contract in a uniform scenario.

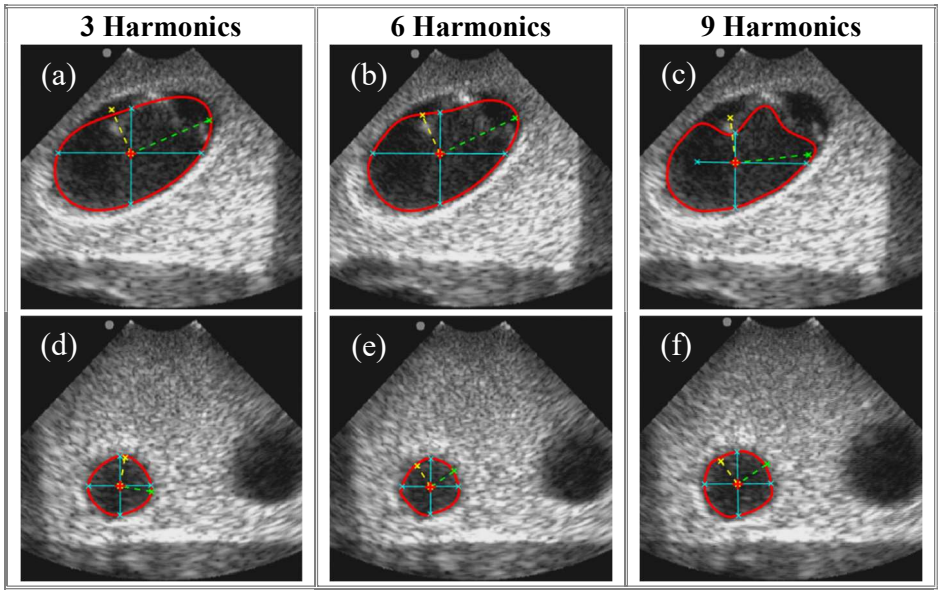

Figure 4-11: Acquired images for the tracking algorithm demonstrating deformation via out-of-plane rotation (a)-(c) and contraction via out-of-plane translation (d)-(f) for contour models with 3,6 and 9 harmonics.

Another important property to look at, in a quantitative approach, is the time each algorithm iteration takes to execute. This is an important aspect to explore since it is the main limiting factor for a real-time tracking approach. For this purpose, the processing time at each algorithm iteration for each acquisition frame was measured and the mean computing time and maximum frame rate for models combining 3,6 and 9 harmonics and 128 and 256 contour points was computed. The values for computation time are displayed on Table 4-4 and the values for mean frame rate are presented on Table 4-5.

Table 4-4: Results of the measurement of the computation time of the tracking algorithms in ms.

Number of Points	3 Harmonics (ms)	6 Harmonics (ms)	9 Harmonics (ms)
128	20.6	22.1	23.6
	±0.8	±1.4	±1.1
256	21.2	22.3	24.3
	±1.4	±1.4	±1.5

Table 4-5: Results of the temporal performance testing of the algorithm as the mean frame rate.

Number of Points	3 Harmonics (fps)	6 Harmonics (fps)	9 Harmonics (fps)
128	48.54	45.25	42.37
256	47.17	44.84	41.15

CHAPTER 5 - DISCUSSION AND CONCLUSION

In this chapter, the previously presented results are discussed. It begins with discussion of the calibration results and comments on the improvement of such routine in future implementations and develops the discussion of the algorithm's tracking ability and robustness to different contour models as well as its real-time ability. In the end, this chapter draws the closing conclusions of the dissertation and establishes the current limitations of the algorithm proposed in this dissertation, providing however solutions that may be implemented in a future work.

5.1 Discussion

Following the last chapter, the experimental setups and results for image calibration and algorithm tracking, the discussions of their respective results are separated in the two sections that follow.

5.1.1 Image Calibration

The execution of a simplified image calibration routine allows the algorithm to be applied to a referenced image space, where the acquisition of contour geometric features is in millimetres. From the results presented on Table it can be observed that the pixel spacing parameters can be estimated with a simplistic approach delivering results that are aligned with what is expected. The values demonstrate that separated estimation for vertical and horizontal patterns offers a more effective spacing estimation, entailing lower error and delivering a calibration where pixels are approximately square, as compared to the diagonal pattern approach.

The main advantage of using an image calibration technique based on a N-wire phantom, as it was done in this dissertation, is that it is a multifaceted and expandable approach allowing future implementation of full 3D calibration procedures using a robotic arm with an US probe tracking strategy, as described in [33], with the implementation of Z-wire phantom patterns that enable the extrapolation of out-of-plane parameters.

5.1.2 Tracking Algorithm Testing

In terms of qualitative tracking ability, the active contours have shown desirable qualities for an effective tracking algorithm, such as compliance with the motions induced by probe movement in the tracked structure and robustness when that structure changes shape and location quickly in between image frames. As seen from the captures of Figure 4-10. the active contours-based algorithm can easily and accurately adapt to in-plane motions of the US probe, being able to track the selected structure through the horizontal translations, as shown in Figure 4-10(a)-(c), vertical translations, as shown in Figure 4-10(d)-(f) and through in-plane tilting , as shown in Figure 4-10.(g)-(i), for the three proposed models.

From the captures presented on Figure 4-11 we can observe that the algorithm is sensitive and capable of tracking either the deformations induced by out-of-plane rotation, as presented in Figure 4-11(a)-(c), or contractions induced by out-of-plane translations, as presented in Figure 4-11(d)-(e). For the out-of-plane motions, however, there are varying results depending on the detail of the model used, due to more drastic changes in the topology of the contours.

In general it can be observed that the captures correspondent to models with a lower number of harmonics, as in Figure 4-10 (a), (d), (f) and Figure 4-11 (a), (d), show a faster shape adaptation to both

contour motion and deformation, due to the simplicity of the model. For models using a higher number of harmonics, as in Figure 4-10(c), (e), (g) and Figure 4-11. (c), (e), the captures show a more accurate boundary description of the structure's shape due to higher sensibility to the action of image potentials.

Despite their good results there are performance issues for both models on the extremities, lower and higher harmonics, relates with the model's sensibility. For lower harmonics we have stiffer contours, meaning that there is the possibility for them to skip over the potential barriers imposed by the image potentials leading many times to the loss of convergence. For higher harmonics, the contours are looser and as such more sensible to image potentials, meaning that the contour will tend to converge to less significant image features.

In terms of quantitative temporal performance, it can be observed from the values of mean computing time and frame rate of the algorithm presented in Table 4-4 and Table 4-5 and that this implementation of the tracking algorithm can easily accompany an ultrasound acquisition systems that with a frame rate of 25 fps. For either low or high detail models, varying in harmonics and contour points, the algorithm is able to establish and keep a frame rate in the interval of 40 to 50 fps, meaning that even for faster systems the algorithm would still be useful as a real-time approach. By comparison with an algorithm of the same nature, as the one detailed in [26], the implementation details of the algorithm presented in this dissertation make it 1.5 to 2 times more efficient.

5.2 Conclusions

This dissertation presents and implements the basic framework for a parametric active contour-based tracking algorithm using Fourier descriptors. Along with the description of MATLAB implementation, this dissertation also as the aim of testing and validating the active-contour algorithm. To this end, a simple calibration of US imaging through a N-Wire phantom has been performed. Finally, the performance of the algorithm has been tested using an agar-agar US phantom with an acquisition routine covering all basic probe motions, with respective acquisition of captures of contour behaviour for qualitative analysis and computation time, for qualitative analysis.

An efficient implementation of the semi-automatic tracking algorithm is done, with the aim of constructing an automatic US diagnostics robotic system. It is shown not only its compatibility with a real-time approach but also its adaptability and robustness in tracking specific structures during an ultrasound procedure with an Agar-agar ultrasound phantom. Thus, validating the use of parametric active contours based on Fourier descriptor as the principal component of the tracking program.

This contour algorithm in its present state still has some limitations that need to be further tackled for a fully automatic tracking approach. These limitations are a source of study for future iterations of the tracking program, such as:

- The lack of multi-contour tracking, meaning the procedure needs to be generalized to multi-contours based on object-centred programming;
- The target region is initially given by the physician, although such regions can be autonomously detected (or suggested) by implementing region-based segmentation;
- The current image potentials are a limiting factor, not enabling evolution to a specific interest region. As such the program requires the introduction of region-based potentials that use region seeding to compute new potentials;
- The lack of range of the image potentials, that forces the system to be compensated by surface force. The program requires a processing tool to enlarge the action region of the potential that is compatible with real time approaches;

- Inability to handle extreme topology events, such as self-intersection and separation, requiring the implementation of a topology resolution routine.

In a future iteration of this algorithm all these limitations will be taken into account to further develop a more precise and robust US tracking tool.

REFERENCES

[1] Adriana Gonzales *et al.*, "TER: a system for robotic tele-echography," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2208, 2001.

[2] L. Al Bassit, "OTELO project: mObile Tele-Echography using an ultra-Light rObot," no. June, 2014.

[3] N. Smith-Guerin, L. Al Bassit, G. Poisson, C. Delgorge, P. Arbeille, and P. Vieyres, "Clinical validation of a mobile patient-expert tele-echography system using ISDN lines," *Proc. IEEE/EMBS Reg. 8 Int. Conf. Inf. Technol. Appl. Biomed. ITAB*, vol. 2003–Janua, no. May, pp. 23–26, 2003.

[4] L. Santos and R. Cortesão, "A Dynamically Consistent Hierarchical Control Architecture for Robotic-Assisted Tele-Echography," no. Iros, 2010.

[5] L. Santos and R. Cortesão, "A dynamically consistent hierarchical control architecture for robotic-assisted tele-echography with motion and contact dynamics driven by a 3D time-of-flight camera and a force sensor," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2015–June, no. June, pp. 2931–2937, 2015.

[6] R. Cortesao, J. Park, and O. Khatib, "Real-Time Adaptive Control for Haptic Telemanipulation With Kalman Active Observers," *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, vol. 3, no. 5, pp. 987–999, 2003.

[7] R. Cortesão and M. Dominici, "Robot Force Control on a Beating Heart," *IEEE/ASME Trans. Mechatronics*, vol. 22, no. 4, pp. 1736–1743, 2017.

[8] P. Chatelain, A. Krupa, and N. Navab, "3D ultrasound-guided robotic steering of a flexible needle via visual servoing," *Proc. - IEEE Int. Conf. Robot. Autom.*, 2011.

[9] P. Chatelain, A. Krupa, and M. Marchal, "Real-time needle detection and tracking using a visually servoed 3D ultrasound probe," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1679–1681, 2013.

[10] A. Krupa, J. Chevrie, M. Babel, S. Misra, and N. Shahriari, "Flexible Needle Steering in Moving Biological Tissue With Motion Compensation Using Ultrasound and Force Feedback," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2338–2345, 2018.

[11] A. Krupa, G. Fichtinger, and G. D. Hager, "Full motion tracking in ultrasound using image speckle information and visual servoing," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 2458–2464, 2007.

[12] A. Krupa, G. D. Hager, and G. Fichtinger, "Real-time tissue tracking with B-mode ultrasound using speckle and visual servoing," *Med. image Comput. Comput. Interv. MICCAI ...International Conf. Med. Image Comput. Comput. Interv. JID - 101249582*, vol. 10, no. Pt 2, pp. 1–8, 2013.

[13] P. Chatelain, A. Krupa, and N. Navab, "Optimization of ultrasound image quality via visual servoing," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2015–June, no. June, pp. 5997–6002, 2015.

[14] W. Bachta and A. Krupa, "Towards ultrasound image-based visual servoing," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2006, pp. 4112–4117, 2006.

[15] C. Nadeau and A. Krupa, "Intensity-based direct visual servoing of an ultrasound probe," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 5677–5682, 2011.

[16] C. Nadeau, A. Krupa, and J. Gangloff, “Automatic tracking of an organ section with an ultrasound probe: Compensation of respiratory motion,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6891 LNCS, no. PART 1, pp. 57–64, 2011.

[17] A. Krupa, “Automatic calibration of a robotized 3D ultrasound imaging system by visual servoing,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2006, pp. 4136–4141, 2006.

[18] P. Chatelain, A. Krupa, and N. Navab, “Confidence-driven control of an ultrasound probe,” *IEEE Trans. Robot.*, vol. 33, no. 6, pp. 1410–1424, 2017.

[19] S. Hutchinson and F. Chaumette, “Visual Servo Control Part I : Basic Approaches,” *IEEE Robot. Autom. Mag.*, no. December, pp. 82–90, 2006.

[20] R. Mebarki, A. Krupa, and C. Collewet, “Automatic guidance of an ultrasound probe by visual servoing based on B-mode image moments,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5242 LNCS, no. PART 2, pp. 339–346, 2008.

[21] F. Chaumette and S. A. Hutchinson, “Visual Servo Control, Part II: Advanced Approaches,” *IEEE Robot. Autom. Mag.*, vol. 14, no. March, pp. 109–118, 2007.

[22] R. Mebarki, A. Krupa, and F. Chaumette, “Image moments-based ultrasound visual servoing,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 113–119, 2008.

[23] M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: Active contour models,” *Int. J. Comput. Vis.*, vol. 331, 1988.

[24] C. Collewet, “Polar snakes: A fast and robust parametric active contour model,” *Proc. - Int. Conf. Image Process. ICIP*, no. 1, pp. 3013–3016, 2009.

[25] F. Precioso, M. Barlaud, T. Blu, and M. Unser, “Robust real-time segmentation of images and videos using a smooth-spline snake-based algorithm,” *IEEE Trans. Image Process.*, vol. 14, no. 7, pp. 910–924, 2005.

[26] T. Li, A. Krupa, and C. Collewet, “A robust parametric active contour based on Fourier descriptors,” *Proc. - Int. Conf. Image Process. ICIP*, no. 1, pp. 1037–1040, 2011.

[27] A. Krupa, C. Collewet, and C. De Beaulieu, “Un contour actif robuste basé sur les descripteurs de Fourier,” 2011.

[28] D. Terzopoulos and D. Metaxas;, “Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 7, 1991.

[29] J. Ivins and J. Porrill, “Active region models for segmenting textures and colours,” *Image Vis. Comput.*, vol. 13, no. 5, pp. 431–438, 1995.

[30] O. Soldea, M. Unel, and A. Ercil, “Recursive computation of moments of 2D objects represented by elliptic Fourier descriptors,” *Pattern Recognit. Lett.*, vol. 31, no. 11, pp. 1428–1436, 2010.

[31] F. Chaumette, “Image Moments: A General and Useful Set of Features for Visual Servoing,” *IEEE Trans. Robot.*, vol. 20, no. 4, pp. 713–723, 2004.

[32] N. Otsu, “A Treshold Selection Method from Gray-Level Histograms,” *IEEE Trans. Syst. Man. Cybern.*, vol. 9, no. 1, pp. 62–66, 1979.

[33] L. Mercier, T. Langø, F. Lindseth, and D. L. Collins, “A review of calibration techniques for freehand 3-D ultrasound systems,” *Ultrasound Med. Biol.*, vol. 31, no. 4, pp. 449–471, 2005.

[34] M. Earle, G. De Portu, and E. Devos, “Agar ultrasound phantoms for low-cost training without refrigeration,” *African J. Emerg. Med.*, vol. 6, no. 1, pp. 18–23, 2016.

APPENDICES

This sections contains the appendices refering to the active contours based tracking program. The original MATLAB code written through this project is included in this chapter to demonstrate de computational logic and implementation of each section of the algorithm.

Appendix I - Code for the image acquisition routine with the algorithm

Appendix II - Code for the algorithm as a standalone tool in an acquired video

Appendix III - Code for the different blocks of the tracking algorithm

Appendix IV - Code for the image calibration computation routine

Appendix I - Code for the image acquisition routine with the algorithm

Complete Program - ImageAcquisitionTest.m

```
%=========================================================================
%   Test script for the Tracking Algorithm with Real-Time Image Acquisition
%=========================================================================

%% Function Over-Head resolution
  gpuArray(); % First Execution of GPUarray in order to spare the function
              % over-head time

%% Calibration Procedure
CalibrationTest;

%% Initial Configuration of the Video Input Object Handle

vidInput = videoinput('winvideo', 1, 'YUY2_640x480');
vidInput.ReturnedColorspace = 'grayscale';
% vidInput.SelectedSourceName = 'svideo';
triggerconfig(vidInput, 'manual');

MaxFrames = 1e5;
FrameRateInterval = 10;

dX = FinalStrc.Horizontal.Medium.MeanD;
dY = FinalStrc.Vertical.Medium.MeanD;
dXY = [dX dY];

%% Initial Configuration of the Fourier Active Contours Parameters

%  Desired configuration

nHarmonics = 2;                 % Number of usable harmonics
nPoints = 128*2;                % Number of sampling points in the contour

nIterations = 200;              % Number of maximum iterations per cycle
dispError = sqrt(2)*sqrt(dX^2+dY^2);    % Displacement error criteria

% Base configuration

nHarmonics_Base = 6;
nPoints_Base = 128*2;

force = 56.0886;%79.5838;%39.5865;% ;
alpha = 0.0027;
beta = 0.4020;
sensi = 0.0309;
rho = 2.3693;
p = 0.80;

gamma_Base =  force/(sensi*pi);                     % Viscosity coefficient

k1_Base = ((alpha*beta*gamma_Base)/((beta+1)*nHarmonics_Base^2));% Extension coefficient
k2_Base = ((alpha*gamma_Base)/((beta+1)*nHarmonics_Base^4));     % Curvature coefficient
k3_Base = (-rho*(k1_Base+k2_Base));                 % Area energy coefficient

% Reparametrization ratios

rH = nHarmonics_Base/nHarmonics;
rN = nPoints/nPoints_Base;

r_k1 = rH^(3/2)*rN;
r_k2 = rH^(7/2)*rN^3;
r_k3 = rH^(7/2)*(1+beta*nHarmonics^2)/(1+beta*nHarmonics_Base^2);
r_gamma = sqrt(1/rH);

% Parameter Computation

gamma = gamma_Base*r_gamma;

k1 = k1_Base*r_k1;
k2 = k2_Base*r_k2;
```

```
k3 = k3_Base*r_k3;

deltaT = (1-p)*gamma/(k1*nHarmonics^2+k2*nHarmonics^4+abs(k3)*nHarmonics);   % Sampling rate of the algorithm

% Control Matrix Computations

[DXdqi, DYdqi] = fourierJacobian(nHarmonics,nPoints);
[C,K] = conditionMatrices2(gamma,k1,k2,k3,nHarmonics);

%% Initial Configuration of the Pre-Processing Parameters

sigma = 6;
nHood = strel('disk',8);

ROImask =  gpuArray(~cell2mat(struct2cell((load('ROImaskUltrasound')))));
ROImask =  imcomplement(preProcessUS2(ROImask,10,strel('disk',2)));

Wroi =  0.8;
Wline = 0.25;
Wedge = 1.15;
Wterm = 0.15;
Neigh = 0;

%% Initial Configuration of the Figure Display Handle

videoResolution = fliplr(vidInput.VideoResolution);
imageReference = imref2d(videoResolution,dX,dY);
figureHandle1 = figure('Position',[20 250 640*1.5 480*1.5]);
set(figureHandle1,'WindowStyle','modal');

BreakButton = uicontrol('Style', 'pushbutton', 'String', 'Break',...
   'Position', [410 20 70 50],'Value',1);
CloseButton = uicontrol('Style', 'pushbutton', 'String', 'Close',...
   'Position', [480 20 70 50],'Callback','close(gcbf)','Visible','off');
set(CloseButton,'BackgroundColor',[0 0.75 0]);
set(BreakButton,'BackgroundColor',[0.75 0 0]);

ImageHandle = imshow(zeros(videoResolution,'double'),imageReference);hold on

% Initializing the plotting components


LineHandle1 = line(zeros(nPoints+1,1),zeros(nPoints+1,1),'lineWidth',2.5,'Color','r');
LineHandle2 = line(zeros(1,2),zeros(1,2),'lineWidth',1.5,'Color','g','LineStyle','--','Marker','x');
LineHandle3 = line(zeros(1,2),zeros(1,2),'lineWidth',1.5,'Color','y','LineStyle','--','Marker','x');
LineHandle4 = line(zeros(1,2),zeros(1,2),'lineWidth',1.5,'Color','c','LineStyle','--','Marker','x');
LineHandle5 = line(zeros(1,2),zeros(1,2),'lineWidth',1.5,'Color','c','LineStyle','--','Marker','x');
MarkerHandle = line(0,0,'Marker','+','LineWidth',2.5,'MarkerSize',10,'Color','r');


% Initializing Contour Data Display components

TextFormatFPS = 'FrameRate: %.3f fps';
TextFPSHandle = text(25*dX,(videoResolution(1)-50)*dY,'','Color','g','LineWidth',12);

TextFormatArea = 'Area: %.0f mm^2';
TextFormatPosition = 'Position: (%.0f , %.0f) mm';
TextFormatAngle = 'Angle: %.3f deg';
TextAreaHandle = text(25*dX,350*dY,'','Color','g','LineWidth',12);
TextPositionHandle = text(25*dX,365*dY,'','Color','g','LineWidth',12);
TextAngleHandle = text(25*dX,380*dY,'','Color','g','LineWidth',12);

% Setting their Initial Visibility (OFF)
set(LineHandle1,'Visible','off');
set(LineHandle2,'Visible','off');
set(LineHandle3,'Visible','off');
set(LineHandle4,'Visible','off');
set(LineHandle5,'Visible','off');
set(MarkerHandle,'Visible','off');

hold off

%% Chronometer Vectors

loopTime = zeros(1,MaxFrames);
```

```
%% Running the Image Aquisition Routine with per loop trigger

isKeyPressed = 0;
frameNumber = 0;
HandSelect = 1;

start(vidInput); % Starts the image aquisition for the vidInput device object
while (true)

    % Updating the frame index
    frameNumber = frameNumber + 1;

    % Getting the frame from vidInput Device

    tic;
    vidFrame = (im2double(gpuArray(getsnapshot(vidInput)))); % Grab frame in-loop and convert to double
    AcqTime = toc;

    set(ImageHandle, 'CData', gather(vidFrame));

    tic;
    if (frameNumber==1)

        % Contour Detection and Selection

        if(HandSelect == 1)
            hold on
            [SelectLineHandle,XData,YData] = freehanddraw(gca,'color','r','linewidth',2.5);
            set(SelectLineHandle,'Visible','off');
            ContOut = interparc(nPoints,XData,YData,'linear');
            hold off
        else
            hold on
            [vidFrame,lineHandles,labelHandles] = preProcessUS2(vidFrame,sigma,nHood,'dark',~ROImask1);
            lineHandles = plotContours(lineHandles,labelHandles,dXY);
            ContOut = selecLineInPlot(lineHandles);
            hold off
        end
        ContStart = interparc(nPoints,ContOut(:,1),ContOut(:,2) ,'linear');
        qInit = fourierDescriptorSet(ContStart,nHarmonics);
    end

    vidFrame = preProcessUS2(vidFrame,sigma,nHood);
    [EimX,EimY] = imagePotentialEnergy2(vidFrame,ROImask2,Wroi, Wline, Wedge, Wterm, Neigh, []);

    qInit = FourierSnakes5(qInit,nPoints,C,K,gather(EimX),gather(EimY),DXdqi,DYdqi,deltaT,dispError,nIterations,dXY);

    % Even-Spaced Interpolation

    ContStart = retrieveContour(qInit,nPoints);
    ContInt = ContStart(1:2:nPoints,:);
    ContStart = interparc(nPoints,ContInt(:,1),ContInt(:,2),'linear');
    qInit = fourierDescriptorSet(ContStart,nHarmonics);

    % Feature Computation

    [Area,massC,Orientation] = contourFeatures(qInit);

    majP1 = round(massC + Orientation(2)*[cos(Orientation(1)) sin(Orientation(1))]);
    majP2 = round(massC + Orientation(2)*[cos(Orientation(1)+pi) sin(Orientation(1)+pi)]);

    minP1 = round(massC + Orientation(3)*[cos(Orientation(1)+pi/2) sin(Orientation(1)+pi/2)]);
    minP2 = round(massC + Orientation(3)*[cos(Orientation(1)+3*pi/2) sin(Orientation(1)+3*pi/2)]);

    xCross = find(ContStart(:,2)<=massC(2)+2*sqrt(2)*dY & ContStart(:,2)>=massC(2)-2*sqrt(2)*dY);
    mX = mean(xCross);
    minXCross = xCross(xCross<mX); maxXCross = xCross(xCross>mX);
    minXPoint = mean(ContStart(minXCross,:),1);
    maxXPoint = mean(ContStart(maxXCross,:),1);

    yCross = find(ContStart(:,1)<=massC(1)+2*sqrt(2)*dX & ContStart(:,1)>=massC(1)-2*sqrt(2)*dX);
    mY = mean(yCross);
    minYCross = yCross(yCross<mY); maxYCross = yCross(yCross>mY);
    minYPoint = mean(ContStart(minYCross,:),1);
    maxYPoint = mean(ContStart(maxYCross,:),1);
```

```
    set(LineHandle1,'XData',[ContStart(:,1); ContStart(1,1)]);
    set(LineHandle1,'YData',[ContStart(:,2); ContStart(1,2)]);

    set(LineHandle2,'XData',[majP1(1);massC(1)]);
    set(LineHandle2,'YData',[majP1(2);massC(2)]);
    set(LineHandle3,'XData',[massC(1);minP2(1)]);
    set(LineHandle3,'YData',[massC(2);minP2(2)]);

    set(LineHandle4,'XData',[minXPoint(1);maxXPoint(1)]);
    set(LineHandle4,'YData',[minXPoint(2);maxXPoint(2)]);
    set(LineHandle5,'XData',[minYPoint(1);maxYPoint(1)]);
    set(LineHandle5,'YData',[minYPoint(2);maxYPoint(2)]);

    set(MarkerHandle,'XData',massC(1));
    set(MarkerHandle,'YData',massC(2));

    set(TextAreaHandle,'String', sprintf(TextFormatArea,Area));
    set(TextPositionHandle,'String', sprintf(TextFormatPosition,massC(1),massC(2)));
    set(TextAngleHandle,'String', sprintf(TextFormatAngle,radtodeg(-Orientation(1))));

    set(LineHandle1,'Visible','on');
    set(LineHandle2,'Visible','on');
    set(LineHandle3,'Visible','on');
    set(LineHandle4,'Visible','on');
    set(LineHandle5,'Visible','on');
    set(MarkerHandle,'Visible','on');


    loopTime(frameNumber) = toc + AcqTime;

    % Instantaneous FrameRate Computation

    if(rem(frameNumber,FrameRateInterval+1)==0)
        FrameRate = 1/(mean(loopTime(frameNumber-FrameRateInterval:frameNumber)));
        FrameRateS = sprintf(TextFormatFPS,FrameRate);
        set(TextFPSHandle,'String',FrameRateS);
    end

    % Loop breaking routine


    isKeyPressed = ~get(BreakButton,'Value');
    if (isKeyPressed)
        set(CloseButton,'Visible','on');
        break
    end
    pause(1/60)

end
delete(vidInput); % Cleans the vidInput Device handle

effectiveFrameRate = 1/mean(loopTime(loopTime>0));
disp(['Average Frame Rate: ', num2str(effectiveFrameRate),' fps']);
```

Appendix II - Code for the standalone algorithm for an acquired video

Complete Program - VideoReadTest.m

```
%==========================================================================
%   Test script for the video reader routine in MATLAB with in-line
%   Despckeling, Gaussian Smoothing and Image Potential Computation
%==========================================================================

%% Function Over-Head Time Resolution
gpuArray();

%% Calibration Procedure
CalibrationTest;


%% Initial Configuration of the Video Object Handle

vidObject = VideoReader('CalibPhantom_Video_10.avi');
frameRate = inf;
FrameRateInterval = 10;
startPoint = 0 ;
stopTime = inf;
totalFrames = round(vidObject.FrameRate*(vidObject.Duration-startPoint),0);

dX = FinalStrc.Horizontal.Medium.MeanD;
dY = FinalStrc.Vertical.Medium.MeanD;

dXY = [dX dY];
set(vidObject,'CurrentTime',startPoint)

%% Initial Configuration of the Fourier Active Contours Parameters

%  Desired configuration

nHarmonics = 6;                 % Number of usable harmonics
nPoints = 128*2;                % Number of sampling points in the contour

nIterations = 200;              % Number of maximum iterations per cycle
dispError = sqrt(2)*sqrt(dX^2+dY^2);    % Displacement error criteria

% Base configuration

nHarmonics_Base = 6;
nPoints_Base = 128*2;

force = 56.0886;%79.5838;%39.5865;% ;
alpha = 0.0027;
beta = 0.4020;
sensi = 0.08;
rho = 2.8;
p = 0.95;

gamma_Base =  force/(sensi*pi);                     % Viscosity coefficient

k1_Base = ((alpha*beta*gamma_Base)/((beta+1)*nHarmonics_Base^2));% Extension coefficient
k2_Base = ((alpha*gamma_Base)/((beta+1)*nHarmonics_Base^4));     % Curvature coefficient
k3_Base = (-rho*(k1_Base+k2_Base));                 % Area energy coefficient

% Reparametrization ratios

rH = nHarmonics_Base/nHarmonics;
```

```
rN = nPoints/nPoints_Base;

r_k1 = rH^(3/2)*rN;
r_k2 = rH^(7/2)*rN^3;
r_k3 = rH^(7/2)*(1+beta*nHarmonics^2)/(1+beta*nHarmonics_Base^2);
r_gamma = sqrt(1/rH);

% Parameter Computation

gamma = gamma_Base*r_gamma;

k1 = k1_Base*r_k1;
k2 = k2_Base*r_k2;
k3 = k3_Base*r_k3;

deltaT = (1-p)*gamma/(k1*nHarmonics^2+k2*nHarmonics^4+abs(k3)*nHarmonics);   % Sampling rate of the algorithm

% Control Matrix Computations

[DXdqi, DYdqi] = fourierJacobian(nHarmonics,nPoints);
[C,K] = conditionMatrices2(gamma,k1,k2,k3,nHarmonics);

%% Initial Configuration of the Pre-Processing Parameters

sigma = 6;
nHood = strel('disk',8);
nHoodROI1 = strel('square',7);
nHoodROI2 = strel('square',7);

ROImask1 =  gpuArray(~cell2mat(struct2cell((load('ROImaskUltrasound')))));
ROImaskInt = ~imerode(~ROImask1,nHoodROI1);
ROImask2 =  imcomplement(preProcessUS2(ROImaskInt,8,nHoodROI2));

Wroi =  3;
Wline = 0.50;
Wedge = 1.25;
Wterm = 0.15;
Neigh = 0;

%% Initial Configuration of the Figure Display Hsndle

% Initializing the Image Handle GUI

videoResolution = [vidObject.Height vidObject.Width];
imageReference = imref2d(videoResolution,dX,dY);
figureHandle1 = figure('Position',[20 250 640*1.5 480*1.5]);
set(figureHandle1,'WindowStyle','modal');

BreakButton = uicontrol('Style', 'pushbutton', 'String', 'Break',...
    'Position', [410 20 70 50],'Value',1);
CloseButton = uicontrol('Style', 'pushbutton', 'String', 'Close',...
    'Position', [480 20 70 50],'Callback','close(gcbf)','Visible','off');
set(CloseButton,'BackgroundColor',[0 0.75 0]);
set(BreakButton,'BackgroundColor',[0.75 0 0]);

ImageHandle = imshow(zeros(videoResolution,'double'),imageReference);hold on

% Initializing the plotting components


LineHandle1 = line(zeros(nPoints+1,1),zeros(nPoints+1,1),'lineWidth',2.5,'Color','r');
LineHandle2 = line(zeros(1,2),zeros(1,2),'lineWidth',1.5,'Color','g','LineStyle','--','Marker','x');
LineHandle3 = line(zeros(1,2),zeros(1,2),'lineWidth',1.5,'Color','y','LineStyle','--','Marker','x');
```

```
LineHandle4 = line(zeros(1,2),zeros(1,2),'lineWidth',1,'Color','c','LineStyle','-','Marker','x');
LineHandle5 = line(zeros(1,2),zeros(1,2),'lineWidth',1,'Color','c','LineStyle','-','Marker','x');
MarkerHandle = line(0,0,'Marker','+','LineWidth',2.5,'MarkerSize',10,'Color','r');


% Initializing Contour Data Display components

TextFormatFPS = 'FrameRate: %.3f fps';
TextFPSHandle = text(25*dX,(videoResolution(1)-50)*dY,'','Color','g','LineWidth',12);

TextFormatArea = 'Area: %.2f mm^2';
TextFormatPosition = 'Position: (%.2f , %.2f) mm';
TextFormatAngle = 'Angle: %.3f deg';
TextAreaHandle = text(25*dX,350*dY,'','Color','g','LineWidth',12);
TextPositionHandle = text(25*dX,365*dY,'','Color','g','LineWidth',12);
TextAngleHandle = text(25*dX,380*dY,'','Color','g','LineWidth',12);

% Setting their Initial Visibility (OFF)
set(LineHandle1,'Visible','off');
set(LineHandle2,'Visible','off');
set(LineHandle3,'Visible','off');
set(LineHandle4,'Visible','off');
set(LineHandle5,'Visible','off');
set(MarkerHandle,'Visible','off');

hold off


%% Chronometer Vectors

loopTime = zeros(1,totalFrames);

%% Video Display

frameNumber = 0;
HandSelect = 0;

while (hasFrame(vidObject) && vidObject.CurrentTime < stopTime)

    frameNumber = frameNumber + 1;

    vidFrame = gpuArray(im2double(readFrame(vidObject)));
    tic;
    pause(1/frameRate);
    AcqTime = toc;
    set(ImageHandle, 'CData', gather(vidFrame));

    if (frameNumber==1)

        % Contour Detection and Selection

        if(HandSelect == 1)
            hold on
            [SelectLineHandle,XData,YData] = freehanddraw(gca,'color','r','linewidth',2.5);
            set(SelectLineHandle,'Visible','off');
            ContOut = interparc(nPoints,XData,YData,'linear');
            hold off
        else
            hold on
            [vidFrame,lineHandles,labelHandles] = preProcessUS2(vidFrame,sigma,nHood,'dark',~ROImask1);
            lineHandles = plotContours(lineHandles,labelHandles,dXY);
            ContOut = selecLineInPlot(lineHandles);
            hold off
```

```
    end
    tic
    ContStart = interparc(nPoints,ContOut(:,1),ContOut(:,2) ,'linear');
    qInit = fourierDescriptorSet(ContStart,nHarmonics);
end

tic;

vidFrame = preProcessUS2(vidFrame,sigma,nHood);
[EimX,EimY] = imagePotentialEnergy2(vidFrame,ROImask2,Wroi, Wline, Wedge, Wterm, Neigh, []);

qInit = FourierSnakes5(qInit,nPoints,C,K,gather(EimX),gather(EimY),DXdqi,DYdqi,deltaT,dispError,nIterations,dXY);

% Even-Spaced Interpolation

ContStart = retrieveContour(qInit,nPoints);
ContInt = ContStart(1:2:nPoints,:);
ContStart = interparc(nPoints,ContInt(:,1),ContInt(:,2),'linear');
qInit = fourierDescriptorSet(ContStart,nHarmonics);

% Feature Computation

[Area,massC,Orientation] = contourFeatures(qInit);

majP1 = round(massC + Orientation(2)*[cos(Orientation(1)) sin(Orientation(1))]);
majP2 = round(massC + Orientation(2)*[cos(Orientation(1)+pi) sin(Orientation(1)+pi)]);

minP1 = round(massC + Orientation(3)*[cos(Orientation(1)+pi/2) sin(Orientation(1)+pi/2)]);
minP2 = round(massC + Orientation(3)*[cos(Orientation(1)+3*pi/2) sin(Orientation(1)+3*pi/2)]);

xCross = find(ContStart(:,2)<=massC(2)+3*dY & ContStart(:,2)>=massC(2)-3*sqrt(2)*dY);
mX = mean(xCross);
minXCross = xCross(xCross<mX); maxXCross = xCross(xCross>mX);
minXPoint = mean(ContStart(minXCross,:),1);
maxXPoint = mean(ContStart(maxXCross,:),1);

yCross = find(ContStart(:,1)<=massC(1)+3*dX & ContStart(:,1)>=massC(1)-3*dX);
mY = mean(yCross);
minYCross = yCross(yCross<mY); maxYCross = yCross(yCross>mY);
minYPoint = mean(ContStart(minYCross,:),1);
maxYPoint = mean(ContStart(maxYCross,:),1);

set(LineHandle1,'XData',[ContStart(:,1); ContStart(1,1)]);
set(LineHandle1,'YData',[ContStart(:,2); ContStart(1,2)]);

set(LineHandle2,'XData',[majP1(1);massC(1)]);
set(LineHandle2,'YData',[majP1(2);massC(2)]);
set(LineHandle3,'XData',[massC(1);minP2(1)]);
set(LineHandle3,'YData',[massC(2);minP2(2)]);

set(LineHandle4,'XData',[minXPoint(1);maxXPoint(1)]);
set(LineHandle4,'YData',[minXPoint(2);maxXPoint(2)]);
set(LineHandle5,'XData',[minYPoint(1);maxYPoint(1)]);
set(LineHandle5,'YData',[minYPoint(2);maxYPoint(2)]);

set(MarkerHandle,'XData',massC(1));
set(MarkerHandle,'YData',massC(2));

set(TextAreaHandle,'String', sprintf(TextFormatArea,Area));
set(TextPositionHandle,'String', sprintf(TextFormatPosition,massC(1),massC(2)));
set(TextAngleHandle,'String', sprintf(TextFormatAngle,radtodeg(-Orientation(1))));

set(LineHandle1,'Visible','on');
```

```
    set(LineHandle2,'Visible','on');
    set(LineHandle3,'Visible','on');
    set(LineHandle4,'Visible','on');
    set(LineHandle5,'Visible','on');
    set(MarkerHandle,'Visible','on');


    loopTime(frameNumber) = toc + AcqTime;

    % Instantaneous FrameRate Computation

    if(rem(frameNumber,FrameRateInterval+1)==0)
        FrameRate = 1/(mean(loopTime(frameNumber-FrameRateInterval:frameNumber)));
        FrameRateS = sprintf(TextFormatFPS,FrameRate);
        set(TextFPSHandle,'String',FrameRateS);
    end

    % Loop breaking routine

    isKeyPressed = ~get(BreakButton,'Value');
    if (isKeyPressed)
        pause(0.010);
        set(CloseButton,'Visible','on');
        break
    end

end

loopTime(1) = 0;
effectiveFrameRate = 1/mean(loopTime(loopTime>0));
effectiveTime = mean(loopTime(loopTime>0))*1000;
stdTime = std(loopTime(loopTime>0))*1000;
disp(['Average Frame Rate: ', num2str(effectiveFrameRate),' fps']);
disp(['Average Computing Time: ', num2str(effectiveTime),' ms']);
disp(['Deviation Computing Time: ', num2str(stdTime),' ms']);
```

Appendix III - Code for the different blocks of the algorithm

1. Code used in the image processing step;
2. Code used in the active contour step
3. Code used in the geometric features step

III- 1. – Code used in the image processing step

Filtering and Contour Extraction Function – preProcess.m

```
function [ gpuImageOut,varargout] = preProcessUS2( image,sigma,speckleNhood,varargin)
%%=========================================================================
% This function implements the pre-processing necessary for the
% Fourier-Descriptor Snakes contour tracking algorithm.
%
%   Inputs:
%   - image: [MxN Double/GPUArray] - Image to be processed;
%   - sigma: [Double] - Parameter controling the Gaussian filtering kernels
%                       standard deviation;
%   - speckleNhood: [STREL] -  Neighboorhood definition for the speckle
%                           filtering kernel ( defined with the strel()
%                           function)
%   - varagin:
%       (1) - [String] - Parameter defining the interest structure to be
%                   'dark' or 'bright';
%       (2) - [MxN Binary] - Binary mask of the region of interest in
%                       which the function will operate
%
%   Outputs:
%   - gpuImageOut: [MxN GPUArray] - Processed image
%   - varargout:
%       (1) - [Nx1 Cell] - List of the detected contours in the image;
%       (2) - [Nx1 Cell] - Listo of labels for the detected contours;
%
%=========================================================================

%% Argument Sentences

if(nargin == 3)
   brightDark = 'dark';
end
if (nargin > 3)
   brightDark = varargin{1};
end
if (nargin > 4)
   ROImask = varargin{2};
else
   ROImask = zeros(size(image));
end

%% Converting and pushing image to GPU

if (~(isa(image,'gpuArray')))
   gpuImage = im2double(gpuArray(image));

elseif (~ isa(image,'double'))
   gpuImage = im2double(image);

else
   gpuImage = image;
end
```

```
%% Speckle Noise reduction

%Speckle filter
gpuImage = multiplicativeSpeckleFilter( gpuImage, speckleNhood);

% Gaussian Blurring
gpuImage = gaussianBlurringFilter( gpuImage,sigma );

% Ouputing filtered image
gpuImageOut = gpuImage;

%% Global Thresholding + Morphological + Contour Extraction

if (nargout > 1 && nargout <=3)

   % Global threshold computation
   gpuImage = ROImask.*gpuImage;
   thresh = multithresh(gather(gpuImage(gpuImage>0)),8);

   if (strcmp('dark',brightDark))

      % Applying global threshold
      gpuImage = (gpuImage <= thresh(1));

      % Morphological operations
      gpuImage = imopen(gpuImage,speckleNhood);
      gpuImage = imerode(gpuImage,speckleNhood);
      gpuImage = imdilate(gpuImage,speckleNhood);

   elseif(strcmp('bright',brightDark))

      % Applying global threshold
      gpuImage = (gpuImage >= thresh(end));

      % Morphological operations
      gpuImage = imopen(gpuImage,speckleNhood);
      gpuImage = imerode(gpuImage,speckleNhood);
      gpuImage = imdilate(gpuImage,speckleNhood);

   else
      error('brightDark must be either: "dark" or "bright"')
   end

   % Contour extraction operations

   [contours,labels] = bwboundaries(gather(gpuImage),4,'noholes');

   varargout(1) = {contours};

   if nargout == 3
      varargout(2) = {labels};
   end

end

end
```

Image Potential Computation Function- imagePotentialEnergy.m

```
function [varargout] = imagePotentialEnergy2(image,ROImask,Wroi, Wline, Wedge, Wterm, Neigh, CC)
%%=========================================================================
% This function implements the computation of the image gradient along a
% contour for the Fourier-Descriptor Snakes contour tracking algorithm.
%
%   Inputs:
%   - image: [M x N Double/GPUArray] - Image to be processed;
%   - ROImask: [MxN Binary] - Binary image mask correspondant to the object
%                             of interest
%   -Wroi: [Double] - Weight for the image mask well potential;
%   -Wline: [Double] - Weight for the image line potential;
%   -Wroi: [Double] - Weight for the image edge potential;
%   -Wroi: [Double] - Weight for the image termination potential;
%   -Neigh: [Double] - Radius for the non-singularity potential;
%   -CC: [2x1 Vector] - Location for the non-singularity potential;

%
%   Outputs:
%    - varargout:
%          -(1) - [M x N Double] Sum of weighted image potential;
%          -(2) - [EimX, EimY] = imagePotentialEnergy();
%                 - EimX - [M x N Double] Horizontal derivative of the
%                 weighted image potential;
%                 - EimY - [M x N Double] Vertical derivative of the
%                 weighted image potential;%
%
%=========================================================================
%% Argument Sentences

if (~(isa(image,'gpuArray')))
   image = gpuArray(image);
   warning('Processing takes more time with non-gpuArray images');
else
   image = image.*ROImask;
end

if (Wline == 0 || Wedge == 0 || Wterm == 0 || Neigh == 0)
   zeroArray = gpuArray.zeros(size(image));
end

%% Line, Edge and Termination Potentials Computation

Eroi = Wroi*imcomplement(ROImask);

if (Wline ~= 0)
   Eline = Wline*image;
else
   Eline = zeroArray;
end

if (Wedge ~=0)
   kernDeriv = fspecial('sobel');
   [ImgX,ImgY,ImgGradMag] = fastXYGradients(image,kernDeriv);

   Eedge = Wedge*ImgGradMag;
else
   Eedge = zeroArray;
end

if (Wterm ~= 0)
   if (Wedge == 0)
      kernDeriv = fspecial('sobel');
      [ImgX,ImgY,ImgGradMag] = fastXYGradients(image,kernDeriv);
```

```
    end

    [ImgXX,ImgXY] = fastXYGradients(ImgX,kernDeriv);
    [ImgYY] = fastXYGradients(ImgY,kernDeriv,'y');

    ImgCurv = (ImgYY.*ImgX.^2 -2*ImgXY.*ImgX.*ImgY + ImgXX.*ImgY.^2)./(1+ImgGradMag.^3);

    Eterm = Wterm*ImgCurv;
else
    Eterm = zeroArray;
end

%% Constraint Potentials Computation

% Non-Singularity Potential
if (Neigh ~= 0)
    NonSing = NonSingPotential(size(image), Neigh,CC);
else
    NonSing = zeroArray;
end

%% Image energy potential computation

imagePotential = Eroi + Eline + Eedge + Eterm + NonSing;

if nargout == 1

    varargout{1} = imagePotential;

elseif nargout == 2

    if (Wedge == 0 && Wterm == 0)
        kernDeriv = fspecial('sobel');
    end

    [imagePotentialX, imagePotentialY] = fastXYGradients(imagePotential,kernDeriv);

    varargout{1} = imagePotentialX;
    varargout{2} = imagePotentialY;

else
    error('Too much output arguments');
end

end
```

Fast Computation of Image Gradients Function – fastXYGradients.m

```
function [ varargout ] = fastXYGradients( image, kernDeriv ,varargin )
%%=========================================================================
% This function implements the computation of image derivatives on GPU
%
%   Inputs:
%   - image: [MxN Double/GPUArray] - Image to be processed;
%   - kernDeriv: [STREL] -  Neighboorhood definition for the derivative
%                           kernel ( defined with the strel()
%                           function)
%   - varagin:
%       (1) - [String] - Parameter defining the type of derivative to be
%                        computed ('x','y','mag');
%
%   Outputs:
%   - varargout:
%       (1) - [MxN Double] - Computed specified potential;
%       (2) - [ImDerivX,ImDerivY] = fastXYgradients(image,kernDeriv);
%             - ImDerivX [M x N Double] - Horizontal Image Derivative
%             - ImDerivY [M x N Double] - Vertical Image Derivative
%=========================================================================
%% Argument Sentences

narginchk(2,3);

if (~(isa(image,'gpuArray')))
   gpuImage = gpuArray(image);
else
   gpuImage = image;
end

if (nargout == 1)
   if(isempty(varargin))
      type = 'y';
   else
      type = varargin{1};
   end
end

%% Local Variables Initialization

padSize = filterPadSize(kernDeriv);

%% Fast computation of gradients

gpuImageInt = padarray(gpuImage,padSize,'symmetric');

if nargout == 1
   switch (type)
      case 'x'
         varargout = {conv2(gpuImageInt,kernDeriv','valid')};
      case 'y'
         varargout = {conv2(gpuImageInt,kernDeriv,'valid')};
      case 'mag'
         varargout = {-abs(conv2(gpuImageInt,kernDeriv,'valid')...
         + 1i*(conv2(gpuImageInt,kernDeriv','valid')))};
   end
elseif nargout == 2
   varargout(1) = {conv2(gpuImageInt,kernDeriv','valid')};
   varargout(2) = {conv2(gpuImageInt,kernDeriv,'valid')};
elseif nargout == 3
   gradientX = conv2(gpuImageInt,kernDeriv','valid');
   gradientY = conv2(gpuImageInt,kernDeriv,'valid');
   varargout(1) = {gradientX};
```

```
    varargout(2) = {gradientY};
    varargout(3) = {-abs(gradientX + 1i*gradientY)};
else
    error('Too much output arguments');

end
end
```

III- 2. – Code used in the active contour step

Fourier Active Contour Function– FourierSnakes.m

```
function [ qFinal] = FourierSnakes5(qInit,nPoints,C,K,EimX,EimY,DXdqi,DYdqi,deltaT,DispError,Niter,dXY)
%%=========================================================================
%
%   This function evolves an active contour in the FourierDescriptor model
%   into the point of force equilibrium between the internal forces and the
%   external forces derived from an image potential
%
%   Inputs:
%    - q:       [4h+2 x 1]Vector - Set of parameters describing the contour
%               in the Elliptic Fourier Descriptor framework developed in
%               [QUOTE]
%
%   Outputs:
%    - Momts:   [Nth x Nth]Matrix - Set of 2D contour moments up to the
%               N-th order
%=========================================================================

%% Initialization of Local Variables

i = 1;
qSave = qInit;

%% Euler Method Iteration Process

while (1)

   qkNext  = FourierSnakesStep4(qInit,nPoints,C,K,EimX,EimY,DXdqi,DYdqi,deltaT,dXY);

   if (mod(i,10) == 0)

      [ ~, DispMeasure] = contourRelativeDisplacement(qSave,qInit,nPoints,'mean');

      if (DispMeasure <= DispError)
         break;
      else
         qSave = qkNext;
      end
   end

   if ( i >= Niter)
      break;
   else
      qInit = qkNext;

   i = i +1;
   end
end

qFinal = qkNext;

end
```

Active Contour Model Iteration Function– FourierSnakesStep.m

```
function [ qNext ] = FourierSnakesStep4( qCurrent,Np,C,K,EimX,EimY,DXdqi,DYdqi,deltaT,dXY)
%%=========================================================================
%
%   This function implements the Euler method step for the Fourier
%   Descriptor active contour model.
%
%   Inputs:
%    - qCurrent:[4h+2 x 1]Vector - Set of parameters describing the contour
%              in the Elliptic Fourier Descriptor framework
%
%   Outputs:
%    - qNext:  [Nth x Nth]Matrix - Set of 2D contour moments up to the
%              N-th order
%=========================================================================

%% Computation of the Image and Surface forces acting on the current contour

Cont = retrieveContour(qCurrent,Np);
PixelBins = round([Cont(:,1)/dXY(1) Cont(:,2)/dXY(2)],0);
Qim = imgForcesComp2( EimX,EimY,DXdqi,DYdqi,PixelBins);

% Estimating the next contour configuration through Explicit Euler Method

I = eye(size(K));
dTinvC = deltaT*diagonalInverse(C);
qNext = (I-dTinvC*K)*qCurrent + dTinvC*Qim;

end
```

Active Contour Control Matrices Computation Function – conditionMatrices.m

```
function [C,K] = conditionMatrices2(gamma,k1,k2,k3,h)
%%=========================================================================
%
%   This function computes the initialization behaviour gain matrices for
%   FD-Snake's curve evolution.
%
%   Inputs:
%    - gamma:   Double - Parameter that describes the viscosity of
%               the medium the curve is set in, it limits the evolution
%               velocity of the curve.
%    - k1:      Double - Parameter that describes the elasticity of
%               the curve, it limits the extension evolution of the curve
%    - k2:      Double - Parameter that describes the resistance of
%               the curve to bending force, it limits the evolution of the
%               curvature of the curve.
%    - k3:      Double - Parameter that describes the weight of
%               the ballon force, it powers the expanding and contracting
%               evolution of the of the curve.
%    - h:       Integer - Number of harmonics the contour is described
%               by.
%
%   Outputs:
%    - C:   [4h+2 x 4h+2]Matrix - Diagonal matrix that describes the
%           viscosity term of the curve evolution in the Fourier parameter
%           space
%    - K:   [4h+2 x 4h+2]Matrix - Diagonal matrix that expresses the global
%           behaviour of the contour in terms of energy:
%
%           - K1:  [4h+2 x 4h+2]Matrix - Diagonal matrix that describes the
%            extension evolution term of the curve evolution in the Fourier
%            parameter space
%           - K2:  [4h+2 x 4h+2]Matrix - Diagonal matrix that describes the
```

```
%           resistance of the curve to bending force term of the curve
%           evolution in the Fourier parameter space
%         - K3:  [4h+2 x 4h+2]Matrix - Diagonal matrix that describes the
%           balloon force, it powers the expanding and contracting
%           evolution of the of the curve in the Fourier parameter space
%
%=========================================================================
%% Initialization of local variables

% Permutation matrix for computation of ballon force

Perm = zeros(4*h+2);

for i = 1:h
   Perm(1+i,(3*h+2)+i) = 1;
   Perm((h+1)+i,(2*h+2)+i) = -1;
   Perm((2*h+2)+i,(h+1)+i) = -1;
   Perm((3*h+2)+i,1+i) = 1;
end

%% Diagonal terms computation

% Harmonic dependant terms

H = repmat([0 1:h 1:h],[1 2]);

% Diagonal of C

DiagnC = pi*ones(1,4*h+2);
DiagnC(1) = 2*DiagnC(1);
DiagnC(2*h+2) = 2*DiagnC(2*h+2);

% Diagonal of K1

DiagnK1 = pi*H.^2;

% Diagonal of K2

DiagnK2 = pi*H.^4;

% Diagonal of K3

DiagnK3 = pi*H;
AreaScaleTerm = 1;


%% Final Matrix Computation

C = gamma*diag(DiagnC);
K1 = k1*diag(DiagnK1);
K2 = k2*diag(DiagnK2);
K3 = k3*Perm*AreaScaleTerm*diag(DiagnK3);
K = K1 + K2 +K3;

end
```

Active Contour Parameter Derivatives Function – fourierJacobian.m

```
function [ DXdqi, DYdqi] = fourierJacobian( nHarmonics, nPoints)
%%=========================================================================
%
%   This function computes de derivatives along the parameters of a Fourier
%   Descriptor Snake, based on the article: [1]. The set of
%   these two matrices constitutes the Jacobian of the transformation
%   between the cartesian space and the EFD parameter space.
%
%   Inputs:
%    - nHarmonics:   Integer - Number of harmonics used to describe the
%                    contour
%    - nPoints:     Integer - Number of points in the contour
%
%   Outputs:
%    - DXdqi:   [4h+2 x Np]Matrix - Matrix that describes de derivatives of
%               the Fourier Snake in the X-component along the parameters
%               for every point of the contour
%    - DYdqi:   [4h+2 x Np]Matrix - Matrix that describes de derivatives of
%               the Fourier Snake in the Y-component along the parameters
%               for every point of the contour
%
%=========================================================================
%% Initialization of Local Variables

DXdqi = zeros(4*nHarmonics+2,nPoints);
DYdqi = zeros(4*nHarmonics+2,nPoints);

%% Computation of harmonic Sines and Cosines

harmonicsMat = repmat((1:nHarmonics)',[1 nPoints]);
paramMat = repmat(linspace(0,1-(1/nPoints),nPoints),[nHarmonics 1]);
argumentMat = 2*pi*harmonicsMat.*paramMat;

cossinesMat = cos(argumentMat);
sinesMat = sin(argumentMat);
constMat = (1/2)*ones(1,nPoints);

%% Attribution of the values to the derivative vectors

% X component derivative

DXdqi(1,:) = constMat;
DXdqi(2:nHarmonics+1,:) = cossinesMat;
DXdqi(nHarmonics+2:2*nHarmonics+1,:) = sinesMat;

% X component derivative

DYdqi(2*nHarmonics+2,:) = constMat;
DYdqi(2*nHarmonics+3:3*nHarmonics+2,:) = cossinesMat;
DYdqi(3*nHarmonics+3:4*nHarmonics+2,:) = sinesMat;

end
```

Fourier Descriptor Computation Function – fourierDescriptorSet.m

```
function [ q ] = fourierDescriptorSet( contourIn, nHarmonics )
%%=========================================================================
%
%   This function implements the Fourier Description for contour curves
%   based on the article: [1] A. Krupa, C. Collewet, and C. De Beaulieu,
%   “Un contour actif robuste basé sur les descripteurs de Fourier,” 2011.
%
%   Inputs:
%    - contourIn:   [N x 2]Vector - Object contour extracted from image in
%                   cooordinates
%    - Nharmonics:  Integer - Number of harmonics to preserve from the
%                   contour
%
%   Outputs:
%    - q:   [4h+2 x 1]Vector - Set of generalized coordinates describing
%           the contour in the Fourier Descriptor framework developed in
%           [1]
%
%=========================================================================
%% Argument sentences

if(isa(contourIn,'gpuArray'))
   flagGpu = true;
else
   flagGpu = false;
end

% if(nHarmonics > 64)
%     warning('For better preformance choose less parameters');
% end

% if(~isPower2(nHarmonics))
%     warning('For better preformance powers of 2 are advised');
% end

% if(nHarmonics >= length(contourIn)/2)
%     warning('Aliasing: Number of harmonics is beyond the Nyqist parameter');
% end

%% Initialization of local variables

if (flagGpu)
   q = gpuArray.zeros(4*nHarmonics + 2,1);
else
   q = zeros(4*nHarmonics + 2,1);
end

%% Computing the Fourier Transform of the x and t components of the contour

fftContour = ifft(contourIn);
RfftContour = real(fftContour);
IfftContour = imag(fftContour);

%% Sorting Fourier Components into generalized coordinates vector

q(1) = RfftContour(1,1);
q(2:(nHarmonics+1)) = 2*RfftContour((2:nHarmonics+1),1);
q((nHarmonics+2):(2*nHarmonics+1)) = 2*IfftContour((2:nHarmonics+1),1);
q(2*nHarmonics+2) = RfftContour(1,2);
q((2*nHarmonics+3):(3*nHarmonics+2)) = 2*RfftContour((2:nHarmonics+1),2);
q((3*nHarmonics+3):(4*nHarmonics+2)) = 2*IfftContour((2:nHarmonics+1),2);

end
```

Contour Reconstruction from Descriptors Function – retrieveContour.m

```
function [ retContour ] = retrieveContour( qIn , nPoints ,varargin)
%%=========================================================================
%
%   This function implements the reconstruction of boundary curves from Fourier Descriptors
%
%   Inputs:
%    - qIn:         [4h+2 x 1]Vector - Set of parameters describing the
%                   contour in the Fourier Descriptor framework developed
%                   in [QUOTE]
%    - nPoints:   [1 x 1]Integer - Number of points to reconstruct the
%                   contour
%    - varargin:  Double - Flag to define if the contour is closed
%
%   Outputs:
%    - retContour:   [Np x 2]Vector - Object contour extracted from image
%                     in the regular cartesian cooordinates
%
%=========================================================================
%% Argument sentences

lengthQ = length(qIn);

if (~(rem(lengthQ,4) == 2))
    error('Invalid size of contour descriptor vector');
end

closeContour = false;

if (~isempty(varargin))
    closeContour = varargin{1};
end

%% Initialization of local variables

nHarmonics = floor(lengthQ/4);

if(isa(qIn,'double'))
    ffContR = zeros(nHarmonics + 1,2);
else
    ffContR = gpuArray.zeros(nHarmonics + 1,2);
end

%% Unsorting of the Fourier Descriptor Set

ffContR(1,:) = [qIn(1)  qIn(2*nHarmonics+2)];
ffContR(2:end,:) =  [qIn(2:nHarmonics+1) + 1i*qIn((nHarmonics+2):(2*nHarmonics+1))...
    qIn((2*nHarmonics+3):(3*nHarmonics+2)) + 1i*qIn((3*nHarmonics+3):(4*nHarmonics+2))];

%% Inverse Fourier computation of the contour

retContour = real(fft(ffContR,nPoints));

if closeContour
    retContour = interparc(nPoints-1,retContour(:,1),retContour(:,2),'linear');
    retContour(end+1,:) = retContour(1,:);
end

end
```

III- 3.– Code used in the geometric features step

Contour Features Computation Function – contourFeatures.m

```
function [ varargout ] = contourFeatures( qIn )
%%=========================================================================
%
%   This function computes the geometric features of a contour based on the
%   on its geometric moments, computed from the Fourier Description of the
%   contour
%
%   Inputs:
%    - qIn:       [4h+2 x 1]Vector - Set of parameters describing the
%                 contour in the EFD framework
%
%   Outputs:
%    - varagout:  [Area] - 1 output - Double - Area enclosed by the contour
%                 [__,massC] - 2 outputs - [1x2]Matrix - Centre of mass of
%                 the contour - [X Y]
%                 [_,_,Pose] - 3 outputs - [1x3]Matrix - In order, the main
%                 contour orientation angle, major and minor axis -
%                 [alpha,majAxis,minAxis]
%=========================================================================

%% Argument Sentences

lengthQ = length(qIn);
if nargout > 3
   error('Undefined output feature');
end
if ((mod(lengthQ,4)-2) ~= 0)
   error('Invalid size of contour descriptor vector');
end

Nth = nargout-1; % Defines maximum order to which the features
                 % need to be computed

%% Contour Features Computation

MomentsMat = contourFMoments3(qIn,Nth);

% FEATURE 1: Area
Area = MomentsMat(1,1); % Raw 0-th Order Moment

varargout{1} = Area;

if Nth > 0

% FEATURE 2: Center of Mass
massCX = MomentsMat(2,1)/MomentsMat(1,1); %Centered 1-st Order Moment
massCY = MomentsMat(1,2)/MomentsMat(1,1); %Centered 1-st Order Moment

massC = [massCX, massCY];

varargout{2} = massC;

end

if Nth > 1
% Centered 2nd Order Moments
myu20 = MomentsMat(3,1)/MomentsMat(1,1) - massCX^2;
myu02 = MomentsMat(1,3)/MomentsMat(1,1) - massCY^2;
myu11 = MomentsMat(2,2)/MomentsMat(1,1) - massCX*massCY;
```

```
% FEATURE 3: Orientation and Elongation

majorAxis = abs(sqrt(2*(myu02+myu20+sqrt((myu20-myu02)^2+4*myu11^2))));
minorAxis = abs(sqrt(2*(myu02+myu20-sqrt((myu20-myu02)^2+4*myu11^2))));
alphaAngle = wrapToPi((1/2)*atan(2*myu11/abs(myu20-myu02)));
varargout{3} = [alphaAngle,majorAxis,minorAxis];
end

end
```

Contour Boundary Moments from Fourier Descriptors – contourFMoments.m

```
function [momentsMat] = contourFMoments3(qIn,Nth)
%%=========================================================================
%
%   This function implements the Fourier Descriptor 2D contour moments
%   computation.
%
%   Inputs:
%    - q:       [4h+2 x 1]Vector - Set of parameters describing the contour
%               in the EFD framework
%    - Nth:     Integer - Order up to which the moments will be computed
%
%   Outputs:
%    - momentsMat:  [Nth x Nth]Matrix - Set of 2D contour moments up to the
%                   N-th order
%=========================================================================

%% Argument sentences

lengthQ = length(qIn);

if ((mod(lengthQ,4)-2) ~= 0)
   error('Invalid size of contour descriptor vector');
end

%% Initialization of local variables

h = floor(lengthQ/4);
Np = 1;
Nm = (Nth+1 + (Nth+1)^2)/2;
count = 0;

ak = [qIn(1);qIn(2:h+1)];
bk = [0; qIn(h+2:2*h+1)];
ck = [qIn(2*h+2); qIn(2*h+3:3*h+2)];
dk = [0; qIn(3*h+3:4*h+2)];

M = zeros(Nm,2);
momentsMat = zeros (Nth+1,Nth+1);

% Generation of the moment idexing

for N = 0:Nth
   for m = 0:Nth
      for n = 0:Nth
         if n+m == N
            count = count +1;
            M(count,:) = [m n];
         end
      end
   end
```

```
end


alpha = zeros(Nth+1,Nth+1,h+1,h+1);
beta = zeros(Nth+1,Nth+1,h+1,h+1);
gamma = zeros(Nth+1,Nth+1,h+1,h+1);

Nbound =(2+Nth)*(h+1);

% Intermediate computations matrices

CosM = zeros(Nth+1,Nth+1,Nbound);
SinM = zeros(Nth+1,Nth+1,Nbound);

% IntM1 = zeros(1,h+1);
% IntM2 = zeros(1,h+1);
% IntM3 = zeros(h+1,h+1);

% Indexing Matrices initialization

siInd = [h+1,1];
indiF = zeros(siInd);
indiB = zeros(siInd);
indiS = zeros(siInd);
indiK = (0:h)';

siInd2 = [h+1,h+1];
indiF2 = zeros(siInd2);
indiB2 = zeros(siInd2);
indiS2 = zeros(siInd2);

indiH = 0:h;
IndiJ = repmat(indiH,[h+1 1]);
IndiI = repmat(indiH',[1 h+1]);

akdk = ak*dk';
ckbk = ck*bk';
akck = ak*ck';
ckak = ck*ak';
dkbk = dk*bk';
bkdk = bk*dk';
bkck = ckbk';

%% Computation of sine and cossine projections of moments

for m = 1:Nm

   p = M(m,1);
   q = M(m,2);
   ord = p+q;

   %Computation of the Harmonic Crossed Coeficients

   ac = (akdk./(p+1)) - (ckbk./(q+1));
   bc = (-akck./(p+1)) + (ckak./(q+1));
   cc = (dkbk./(p+1)) - (bkdk./(q+1));
   dc = (-ckbk./(p+1)) + (akdk./(q+1));

   alpha(p+1,q+1,:,:) = IndiJ.*ac;
   beta(p+1,q+1,:,:) = IndiJ.*bc + IndiI.*cc;
   gamma(p+1,q+1,:,:) = IndiI.*dc;

   % Recusrive computation of Cossine and Sine correlation moments
```

```
subN = Nbound - ord*(h+1);


if p == 0 && q == 0

   % Initial Values Computation (i=0)

   CosM(p+1,q+1,1) = Np ;
   %SinM(p+1,q+1,1) = 0;

   % Subsquent Values (i>0)

   %CosM{p+1,q+1,2:end) = 0;
   %SinM{p+1,q+1,2:end) = 0;

elseif p == 0 && q > 0

   % Initial Values Computation (i=0)

   IntM1 = (ck.*reshape(CosM(1,(q+1)-1,indiK+1),siInd) + dk.*reshape(SinM(1,(q+1)-1,indiK+1),siInd));

   CosM(p+1,q+1,1) = sum(IntM1);
   %SinM(p+1,q+1,1) = 0;

   % Subsquent Values (i>0)

   for i = 1:subN-1

      for k = 0:h
         indiB(k+1) = abs(k-i)+1;
         indiF(k+1) = k+i+1;
         indiS(k+1) = sign(k-i);
      end

      IntM1 = indiK.*(ck.*(reshape(CosM(1,(q+1)-1,indiB),siInd) - reshape(CosM(1,(q+1)-1,indiF),siInd))...
               - dk.*(reshape(SinM(1,(q+1)-1,indiF),siInd) - indiS.*reshape(SinM(1,(q+1)-1,indiB),siInd)));

      IntM2 = indiK.*(-ck.*(reshape(SinM(1,(q+1)-1,indiF),siInd) + indiS.*reshape(SinM(1,(q+1)-1,indiB),siInd))...
               + dk.*(reshape((CosM(1,(q+1)-1,indiF)),siInd) + reshape(CosM(1,(q+1)-1,indiB),siInd)));

      CosM(p+1,q+1,i+1) = (q/(2*i))*sum(IntM1);
      SinM(p+1,q+1,i+1) = (q/(2*i))*sum(IntM2);
   end

elseif p > 0 && q == 0

   % Initial Values Computation

   IntM1 = (ak.*reshape(CosM((p+1)-1,1,indiK+1),siInd) + bk.*reshape(SinM((p+1)-1,1,indiK+1),siInd));

   CosM(p+1,q+1,1) = sum(IntM1);
   %SinM(p+1,q+1,1) = 0;

   % Subsquent Values

   for i = 1:subN-1

      for k = 0:h
         indiB(k+1) = abs(k-i)+1;
         indiF(k+1) = k+i+1;
         indiS(k+1) = sign(k-i);
```

```
        end

        IntM1 = indiK.*(ak.*(reshape(CosM((p+1)-1,1,indiB),siInd) - reshape(CosM((p+1)-1,1,indiF),siInd))...
            - bk.*(reshape(SinM((p+1)-1,1,indiF),siInd) - indiS.*reshape(SinM((p+1)-1,1,indiB),siInd)));

        IntM2= indiK.*(-ak.*(reshape(SinM((p+1)-1,1,indiF),siInd) + indiS.*reshape(SinM((p+1)-1,1,indiB),siInd)))...
            + bk.*((reshape(CosM((p+1)-1,1,indiB),siInd) + reshape(CosM((p+1)-1,1,indiF),siInd)));


        CosM(p+1,q+1,i+1) = (p/(2*i))*sum(IntM1);
        SinM(p+1,q+1,i+1) = (p/(2*i))*sum(IntM2);
    end


elseif p > 0 && q > 0

    % Initial Values Computation

    for i = 0:h
        for j = 0:h
            indiF2(i+1,j+1) = i+j+1+j*subN;
            indiB2(i+1,j+1) = abs(i-j)+1+j*subN;
            indiS2(i+1,j+1) = sign(i-j);
        end
    end
    IntCos = repmat((reshape(CosM((p+1)-1,(q+1)-1,1:subN),[subN 1])),[1 h+1]);
    IntSin = repmat((reshape(SinM((p+1)-1,(q+1)-1,1:subN),[subN 1])),[1 h+1]);

    IntM3 = (akck + bkdk).*IntCos(indiB2)...
        + (akck - bkdk).*IntCos(indiF2)...
        + (-akdk+ bkck).*indiS2.*IntSin(indiB2)...
        + (akdk + bkck).*IntSin(indiF2);

    CosM(p+1,q+1,1) = (1/2)*sum(sum(IntM3));
    %SinM(p+1,q+1,1) = 0;

    %Subsequent Values

    for i = 1:subN-1

        for k = 0:h
            indiB(k+1) = abs(k-i)+1;
            indiF(k+1) = k+i+1;
            indiS(k+1) = sign(k-i);
        end

        IntM1 = indiK.*(p*ak.*(reshape(CosM((p+1)-1, q+1,indiB),siInd) - reshape(CosM((p+1)-1,q+1,indiF),siInd))...
            - p*bk.*(reshape(SinM((p+1)-1,q+1,indiF),siInd) - indiS.*reshape(SinM((p+1)-1,q+1,indiB),siInd))...
            + q*ck.*(reshape(CosM(p+1,(q+1)-1,indiB),siInd) - reshape(CosM(p+1,(q+1)-1,indiF),siInd))...
            - q*dk.*(reshape(SinM(p+1,(q+1)-1,indiF),siInd) - indiS.*reshape(SinM(p+1,(q+1)-1,indiB),siInd)));

        IntM2 = indiK.*(-p*ak.*(reshape(SinM((p+1)-1,q+1,indiF),siInd) + indiS.*reshape(SinM((p+1)-
1,q+1,indiB),siInd))...
            + p*bk.*(reshape(CosM((p+1)-1,q+1,indiF),siInd) + reshape(CosM((p+1)-1,q+1,indiB),siInd))...
            - q*ck.*(reshape(SinM(p+1,(q+1)-1,indiF),siInd) + indiS.*reshape(SinM(p+1,(q+1)-1,indiB),siInd)))...
            + q*dk.*(reshape(CosM(p+1,(q+1)-1,indiF),siInd) + reshape(CosM(p+1,(q+1)-1,indiB),siInd));

        CosM(p+1,q+1,i+1) = (1/(2*i))*sum(IntM1);
        SinM(p+1,q+1,i+1) = (1/(2*i))*sum(IntM2);
    end

end
```

```
    %Full moment computation

    for i = 0:h
        for j =0:h
            indiF2(i+1,j+1) = i+j+1+2*(h+1)*j;
            indiB2(i+1,j+1) = abs(i-j)+1+2*(h+1)*j;
            indiS2(i+1,j+1) = sign(i-j);
        end
    end
    IntCos = repmat((reshape(CosM(p+1,q+1,1:2*(h+1)),[2*(h+1) 1])),[1 h+1]);
    IntSin = repmat((reshape(SinM(p+1,q+1,1:2*(h+1)),[2*(h+1) 1])),[1 h+1]);

    IntM3 = reshape(alpha(p+1,q+1,:,:),[h+1 h+1]).*(IntCos(indiB2) + IntCos(indiF2))...
        + reshape(beta(p+1,q+1,:,:),[h+1 h+1]).*(IntSin(indiF2) - indiS2.*IntSin(indiB2))...
        + reshape(gamma(p+1,q+1,:,:),[h+1 h+1]).*(IntCos(indiB2) - IntCos(indiF2));

    momentsMat(p+1,q+1) = (pi/(2*Np))*sum(sum(IntM3));
end

end
```

Appendix IV - Code for the image calibration computation routine

Image Calibration Program -CalibrationTest.m

```
%==========================================================================
%   Test script for the Image Calibration with N-Wire Geometry
%==========================================================================

%% Image Loading and Organization

fileData = dir('C:\Users\guiac\Desktop\Estágio de Tese\Programs\Images\CalibPhantom/*.png');
fileLength = length(fileData);

imageStcArray = struct.empty;

for i = 1:fileLength

   fileNameString = fileData(i).name;
   nameList = strsplit(fileNameString,{'_','.'});

   imageStcArray(i).name = nameList{1};
   switch(nameList{2})
      case('7,5')
         depth = 'Short';
      case('10')
         depth = 'Medium';
      case('13')
         depth = 'Large';
   end
   imageStcArray(i).depth = depth;
   imageStcArray(i).shape = nameList{3};
   imageStcArray(i).image = gpuArray(im2double(imread(fileNameString)));

end

%% N-Wire Geometry Parameters

tolCNC = 0.1;              % Typical production tolerance

spaceX = 15;                % Horizontal spacing
spaceY = 10;                % Vertical spacing
dXY = [spaceX spaceY];

sigma = 4;
nHood = strel('disk',3);

ROImask =  gpuArray(cell2mat(struct2cell((load('ROImaskUltrasound')))));


%% Distance Computations Procedure

for i = 1:fileLength

    pixelCentreDistance = calibrationFunction(imageStcArray(i).image,imageStcArray(i).shape,dXY,
sigma,nHood,ROImask);
    imageStcArray(i).result = pixelCentreDistance;

end

%% Calibration Results
```

```
ResultStrc = struct;

for i = 1:fileLength

    depth = imageStcArray(i).depth;
    shape = imageStcArray(i).shape;

    if (~isfield(ResultStrc,shape))
        ResultStrc.(shape) = struct;
    end

    if(~isfield(ResultStrc.(shape),depth))
        ResultStrc.(shape).(depth) = imageStcArray(i).result;
    else
        ResultStrc.(shape).(depth) = vertcat( ResultStrc.(shape).(depth),imageStcArray(i).result);
    end


end

%% Mean Calibration Distance Results
 FinalStrc = struct;

Shapes = {'Diagonal','Horizontal','Vertical'};
Depths = {'Short','Medium','Large'};

for i = 1:length(Shapes)
    for j = 1:length(Depths)

        Shape = Shapes{i};
        Depth = Depths{j};

        lengthD = length(ResultStrc.(Shape).(Depth));

        switch(Shape)
            case('Vertical')

                realDistance = ResultStrc.(Shape).(Depth);
                meanD = mean(realDistance);
                stdD = std(realDistance);
                sY = dXY(2)/meanD(2);

                FinalStrc.(Shape).(Depth).MeanD = dXY(2)/meanD(2);
                FinalStrc.(Shape).(Depth).StdD = sqrt(tolCNC^2/meanD(2)^2 + (dXY(2)^2*stdD(2)^2)/(meanD(2)^4));
            case('Horizontal')

                realDistance = ResultStrc.(Shape).(Depth);
                meanD = mean(realDistance);
                stdD = std(realDistance);

                FinalStrc.(Shape).(Depth).MeanD = dXY(1)/meanD(1);
                FinalStrc.(Shape).(Depth).StdD = sqrt(tolCNC^2/meanD(1)^2 + (dXY(1)^2*stdD(1)^2)/(meanD(1)^4));
            case('Diagonal')

                realDistance = ResultStrc.(Shape).(Depth);
                meanD = mean(realDistance);
                stdD = std(realDistance);

                FinalStrc.(Shape).(Depth).MeanD = (dXY/2)./meanD;
                FinalStrc.(Shape).(Depth).StdD = sqrt(tolCNC^2./meanD(2)^2 + ((dXY./2).^2.*stdD.^2)./(meanD.^4));
        end
    end
end
```