Ozyegin University

Department of Computer Science
Faculty of Computer Science

Bachelor Thesis

# A Standalone Application for a 2D Ultrasound Guided Medical Robotic System

Enes Senel

*Supervisors* Erhan Oztop and Ozkan Bebek

May 2016

# Abstract

Ultrasound (US) is one of the most commonly used medical imaging techniques in percutaneous needle procedures. However, US images are inherently noisy and contain excessive number of artifacts. Hence, detecting and tracking the needle tip in US images during the needle insertion is challenging. In this project, we focused on the development of a standalone application which aims to provide an easy-to-use application for segmenting and tracking biopsy needles in 2D US images to use in real world practices. This report presents the visual tracking of the needles using sum of squared differences (SSD) and normalized cross correlation (NCC), a Gabor filter based curve detection algorithm to localize curve shaped needles without any external sensor, 3D estimation of the needle shape using 2D transverse images and a standalone desktop software which provides a robust, easy-to-use interface to run needle localization and tracking algorithms. Also, GPU experiments are conducted to analyze the performance of the algorithms. Finally, the experiments showed that algorithms run in real-time with high accuracy.

# Contents

# 1 Introduction

Percutaneous needle biopsy is a frequently performed medical operation. To have a successful operation, precise insertion of the needle is crucial. Inaccurate placement of the needle may lead to collections from wrong locations or cause some damage to the tissues. Robotic assisted needle insertion is a good way to achieve more precise and robust operation.

There are different types of medical imaging available. Magnetic resonance imaging (MRI) provides clear images, however, it requires the usage of non-magnetic needles and also workspace is limited. Also, needles can be clearly visible in computer tomography (CT) and fluoroscopy images but they are harmful because of the radiation. On the other hand, Ultrasound (US) provides radiation-free and harmless imaging and it can be used with a small probe without requiring a large workspace.

Detecting the needle tip position correctly is very important to operate using the robotic system. Needle tip localization and visual tracking of the needle tip are the main tasks to detect needle tip location using 2D Ultrasound images. Needle tip localization can be used to detect needle axis and estimate the needle tip using Gabor filter and a probability estimation method. On the other hand, during the insertion, probe can move or needle can disappear for some time period. Also, since ultrasound images contain artifacts, it is important to track the needle tip to prevent mis-detections and target losses.

Generally, a biopsy needle appears as a straight line in 2D ultrasound images. However, some operations such as prostate biopsies may include curve shaped needles. To detect the curve needles in 2D US images, a Gabor filter based localization algorithm is proposed. The difference of this algorithm than the others in the literature is that this algorithm does not use any external sensor or localization algorithm.

This project also includes a 3D needle shape estimation method using the visual tracking of transverse 2D ultrasound images. It provides a robust 3D shape estimation without any localization.

Finally, algorithms are embedded in a standalone application to offer a functional, easy-to-use and robust software. Also, algorithms are modified to work with GPUs.

## 1.1 Thesis Structure

**Chapter 2**

This chapter presents the theoretical background of needle tip tracking using *Sum of Squared Differences(SSD)* and *Normalized Cross Correlation(NCC)*. Template update strategies and target loss detection technique are also presented.

**Chapter 3**

Curve needle detection using Gabor filter is explained in this chapter. In most of the operations needle shape is very close to a straight line, however, some operations such as prostate operations may include curve shaped needles. Gabor filter is utilized to detect curve needle shapes and algorithm is explained in detail.

**Chapter 4**

Estimating needle shape is a widely used approach in US guided systems. We propose a tracking based needle shape detection method using transverse ultrasound images. The idea of the proposed method is the ability of estimating 3D needle shape without using any external sensor or localization algorithm.

**Chapter 5**

This chapter will briefly explain the basic implementation details for the needle tracking algorithms.

**Chapter 6**

One of the main goals of this project is having a standalone application which can run needle tip localization and tracking algorithms in real time. This chapter will present the application which was created by using Qt framework.

**Chapter 7**

In this chapter, GPU utilization using OpenCL and CUDA is discussed.

## 1.2 Previous Work

Real-time Needle Tip Localization was developed in 2015 and published in ICAR '15 [3] . The paper extended the work presented in and [4] and [5] by implementing real time needle tip tracking in 2D US images. In our previous studies, the needle axis localization and needle tip estimation in 2D US images were proposed, and the needle axis is localized using a Gabor filter.
In the previous study, the needle tip is also tracked using 2D US imaging, and additionally, the needle tip estimation noise is smoothened using a Kalman filter. The

processing time of the proposed localization method is reduced by approximately 56% using the bin packing method, therefore the algorithm can be executed in real time.

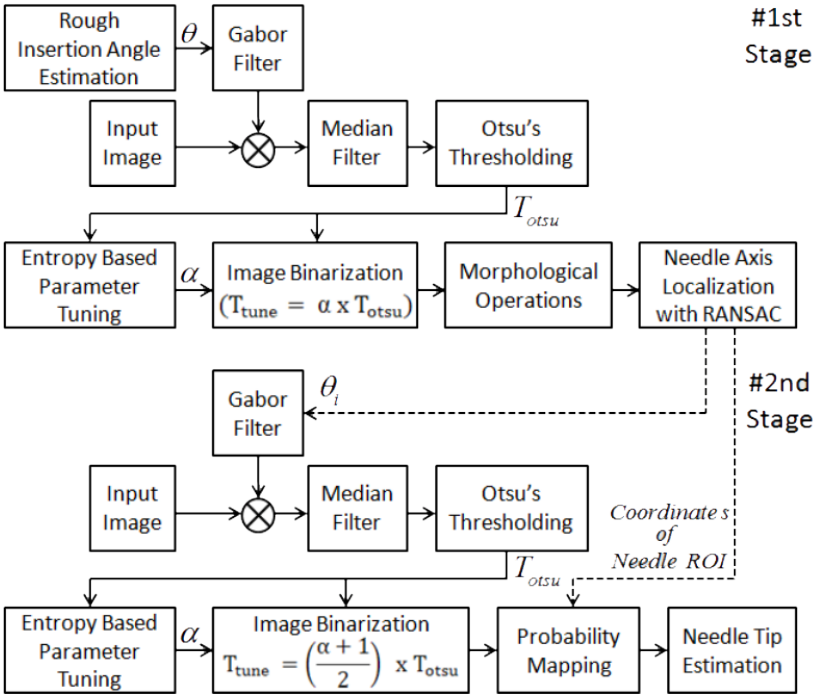



**Fig. 1.1.:** Flowchart of the needle tip localisation algorithm.

Needle tip localization algorithm is implemented using both MATLAB and C++ with OpenCV [1] . A bin packing method to parallelize the execution for multiple frames is also proposed previously. The execution time of sequential and parallel methods are 35.01±3.04 (ms), 13.28±3.30, respectively.

## 1.3 Project Scope and Current Work

This project involves three main building blocks as:

### Algorithm Implementations

Needle tip localization and tracking algorithms are theoretically researched previously and MATLAB implementations are completed in 2015. In the first part of the project, implementations of visual tracking with SSD and NCC were completed.

[1] OpenCV is an open-source computer vision framework. (http://www.opencv.org)

### User Interface and Software

After the completion of all algorithm implementations and also the GPU implementations, the library is completely embedded to a user interface. This user interface will allow users to run algorithms on different datasets. Users are able to select different parameters and try in real-time along with different exporting and sharing options.

### GPU Programming

Needle tip algorithms are evaluated by utilizing GPUs using NVIDIA CUDA[2] and OpenCL [3] support.

[2] A framework for utilizing GPU cores to provide highly parallelizable programs.
[3] An open-source GPU utilizing library which works with both NVIDIA and non-NVIDIA GPUs.

# 2 Visual Tracking of the Needle Tip

Needle Tip Tracking is an important task in medical imaging because of the curiciality of the exact needle location. Since needle tip localization method takes all image into consideration, it is possible to estimate wrong tip locations due to change in the appearance. However, visual tracking of the needle tip can prevent these wrong estimations and provides more robust results.
Visual tracking is achieved by minimizing the error or maximizing the similarity between current and template images, $I(\mathbf{w}(\mathbf{x},\mathbf{p}))$ and $T(\mathbf{x})$, respectively. **w(x,p)** is the warp function that transforms the **x**=(x, y) which is the vector that contains the image pixels.

For the tracking with SSD [1] and NCC [7], following algorithms are proposed.

## 2.1 Tracking Algorithms

### 2.1.1 Visual Tracking using SSD

In this section, we describe the first and second optimization methods of SSD for image registration. SSD between $I$(**w**(**x**,**p**)) and $T$(**x**) can be computed as:

$$SSD(\mathbf{p}) = \sum_{\mathbf{x}} (I(\mathbf{w}(\mathbf{x},\mathbf{p})) - T(\mathbf{x}))^2 \tag{2.1}$$

$I$(**w**(**x**,**p**)) and $T$(**x**) contains the intensity values of N pixels in the the current and reference images, respectively. Parameters, **p**, which minimizes the similarity values between $I$(**w**(**x**,**p**)) and $T$(**x**) are calculated using optimization methods. For this purpose, $I$(**w**(**x**,**p**)) and $T$(**x**) are rewritten in the rectified vector format as:

$$I(\mathbf{w}(\mathbf{x};\mathbf{p})) = \begin{bmatrix} I(\mathbf{w}(\mathbf{x}_1;\mathbf{p})) \\ I(\mathbf{w}(\mathbf{x}_2;\mathbf{p})) \\ \vdots \\ I(\mathbf{w}(\mathbf{x}_N;\mathbf{p})) \end{bmatrix}, \quad T(\mathbf{x}) = \begin{bmatrix} T(\mathbf{x}_1) \\ T(\mathbf{x}_2) \\ \vdots \\ T(\mathbf{x}_N) \end{bmatrix}$$

In order to determine $\mathbf{p}$, firstly Taylor series expansion of $I(\mathbf{w}(\mathbf{x},\mathbf{p}))$ is performed:

$$I(\mathbf{w}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) \approx I(\mathbf{w}(\mathbf{x};\mathbf{p})) + \frac{\partial I(\mathbf{w}(\mathbf{x};\mathbf{p}))}{\partial \mathbf{p}}\Delta\mathbf{p} + \frac{1}{2}\frac{\partial^2 I(\mathbf{w}(\mathbf{x};\mathbf{p}))}{\partial \mathbf{p}^2}\Delta\mathbf{p}^2 + HOT \tag{2.2}$$

The equation above works for multiple dimensions ($I : \mathbf{R}^n \mapsto \mathbf{R}$). In that case, the first and second derivatives can be replaced with Jacobian ($J(\mathbf{p})$) and Hessian ($H(\mathbf{p},\Delta\mathbf{p})$) matrices of $I(\mathbf{w}(\mathbf{x},\mathbf{p}))$, respectively.

$$J(\mathbf{p}) = \frac{\partial I(\mathbf{w}(\mathbf{x};\mathbf{p}))}{\partial \mathbf{p}}, \quad H(\mathbf{p},\Delta\mathbf{p}) = \frac{\partial^2 I(\mathbf{w}(\mathbf{x};\mathbf{p}))}{\partial \mathbf{p}^2} \tag{2.3}$$

Using (2.3), the Taylor series expansion of $I(\mathbf{w}(\mathbf{x},\mathbf{p}))$ can be rewritten as:

$$T(\mathbf{x}) = I(\mathbf{w}(\mathbf{x};\mathbf{p})) + J(\mathbf{p})\Delta\mathbf{p} + H(\mathbf{p},\Delta\mathbf{p})\Delta\mathbf{p}^2 \tag{2.4}$$

By ignoring the $H(\mathbf{p},\Delta\mathbf{p})\Delta\mathbf{p}^2$, the motion parameters can be estimated as follows:

$$\Delta\mathbf{p} = -(J(\mathbf{p}))^{\dagger}(I(\mathbf{w}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})) \tag{2.5}$$

where $()^{\dagger}$ is the pseudo-inverse of the matrix. Equation (2.5) estimates the motion parameters using first order optimization [2]. However, second order optimization of SSD converges faster and is more robust to noise. Also, it can be easily calculated by estimating $H(\mathbf{p},\Delta\mathbf{p})$ as [8]:

$$H(\mathbf{p},\Delta\mathbf{p}) \approx J(\mathbf{p}) - J(\mathbf{p}_0) \tag{2.6}$$

where $J(\mathbf{p}_0)$ is the Jacobian matrix of $T(\mathbf{x})$. By rewritting (2.4) using (2.6), $\Delta\mathbf{p}$ can be computed with second order optimization as follows:

$$\Delta\mathbf{p} = -2(J(\mathbf{p}) + J(\mathbf{p}_0))^{\dagger}(I(\mathbf{w}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})) \tag{2.7}$$

$J(\mathbf{p})$ and $J(\mathbf{p}_0)$ are computed as follow:

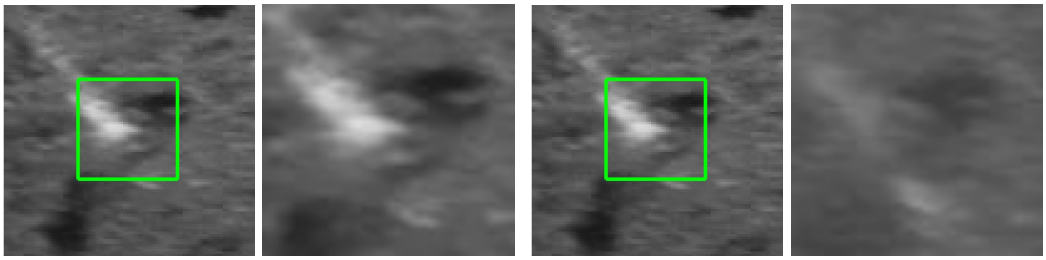
$$J(\mathbf{p}) = \nabla I\frac{\partial \mathbf{w}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}}, \quad J(\mathbf{p}_0) = \nabla T\frac{\partial \mathbf{w}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}}$$

where

$$\frac{\partial \mathbf{w}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

and $\nabla I$, $\nabla T$ are the gradient of $I(\mathbf{w}(\mathbf{x},\mathbf{p}))$ and $T(\mathbf{x})$, respectively.

Parameters, $\mathbf{p}$, are iteratively obtained by accumulating $\Delta\mathbf{p}$ in each iteration. Before the iteration loop starts, $J(\mathbf{p}_0)$ is computed and then $\Delta\mathbf{p}$ is computed in each



(a) Ref. Image (b) $1^{st}$ Template (c) Ref. Image (d) $2^{nd}$ Template

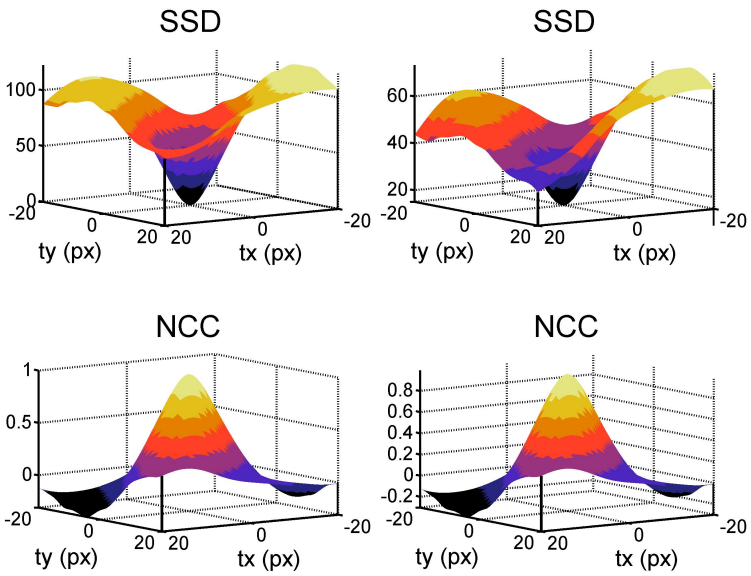



**Fig. 2.1.:** 3D plots of SSD and NCC values with respect to translations between two images. The left column contains similarity plots between (a) Reference image and (b) $1^{st}$ template image. The right column contains the similarity plots between (c) Reference image and (d) $2^{nd}$ template image.

iteration using (2.7). The iterations last until $\Delta\mathbf{p}$ is smaller than the threshold value, $\epsilon$, or the predefined maximum iteration number is reached.

### 2.1.2 Visual Tracking using NCC

In this section, we will describe how to track the needle tip using NCC. NCC between the I($\mathbf{w}$($\mathbf{x}$,$\mathbf{p}$) and T($\mathbf{x}$) can be computed as:

$$NCC(\mathbf{p}) = \frac{\sum_{\mathbf{x}}(I(\mathbf{w}(\mathbf{x},\mathbf{p})) - \overline{I}) \cdot (T(\mathbf{x}) - \overline{T})}{\sqrt{\sum_{\mathbf{x}}(I(\mathbf{w}(\mathbf{x},\mathbf{p})) - \overline{I})^2} \cdot \sqrt{\sum_{\mathbf{x}}(T(\mathbf{x}) - \overline{T})^2}}$$

I(**w**(**x**,**p**) and T(**x**) are vectors contains N pixels in the current and reference image, respectively and they are written in rectified vector format as:

$$I(\mathbf{w}(\mathbf{x},\mathbf{p})) = \begin{bmatrix} I(\mathbf{w}(\mathbf{x}_1,\mathbf{p})) \\ I(\mathbf{w}(\mathbf{x}_2,\mathbf{p})) \\ \vdots \\ I(\mathbf{w}(\mathbf{x}_N,\mathbf{p})) \end{bmatrix}, \quad T(\mathbf{x}) = \begin{bmatrix} T(\mathbf{x}_1) \\ T(\mathbf{x}_2) \\ \vdots \\ T(\mathbf{x}_N) \end{bmatrix}$$

The main purpose of visual tracking is finding **p** parameters which maximize the NCC value between I(**w**(**x**,**p**) and T(**x**) using Newton optimization method. In order to determine p, firstly Taylor series expansion of similarity function (f(**p**)) is performed:

$$f(\mathbf{p}+\Delta\mathbf{p}) \approx f(\mathbf{p}) + f'(\mathbf{p})\,\Delta\mathbf{p} + \frac{1}{2}f''(\mathbf{p})\,(\Delta\mathbf{p})^2 + HOT \tag{2.8}$$

According to Newton optimization method, derivariton of similiarity is equal to zero. For this purpose, derviation of (2.8) is computed as:

$$\frac{\partial f(\mathbf{p}+\Delta\mathbf{p})}{\partial\Delta\mathbf{p}} = \frac{\partial f(\mathbf{p})}{\partial\Delta\mathbf{p}} + \frac{\partial(f'(\mathbf{p})\,\Delta\mathbf{p})}{\partial\Delta\mathbf{p}} + \frac{1}{2}\frac{\partial(f''(\mathbf{x})\,(\Delta\mathbf{p})^2)}{\partial\Delta\mathbf{p}}$$

$$0 = 0 + f'(\mathbf{p}) + f''(\mathbf{p})\Delta\mathbf{p} \tag{2.9}$$

The equation above works for multiple dimensions ($f : \mathbf{R}^n \mapsto \mathbf{R}$). In that case the first and derivatives of f can be replaced with Gradient (**g**) and Hessian (**H**), respectively .

$$\mathbf{g} = f'(\mathbf{p}), \quad \mathbf{H} = f''(\mathbf{p}) \tag{2.10}$$

Using (2.10), derivation of Taylor series expansion of f(**p**) can be rewritten:

$$0 = \mathbf{g} + \mathbf{H}\cdot\Delta\mathbf{p} \tag{2.11}$$

Then, the motion parameters can be calculated using Gradient and Hessian of f(**p**) as follows:

$$\Delta\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g} \tag{2.12}$$

The rest of this section, how to calculate Gradient and Hessian of f(**p**) will be explained. For simplification, (2.20) is divided into three parts:

$$\mathbf{a} = \sum_{\mathbf{x}} (I(\mathbf{w}(\mathbf{x},\mathbf{p})) - \overline{I}) \cdot (T(\mathbf{x}) - \overline{T})$$
$$\mathbf{b} = \sqrt{\sum_{\mathbf{x}} (I(\mathbf{w}(\mathbf{x},\mathbf{p})) - \overline{I})^2}$$
$$\mathbf{c} = \sqrt{\sum_{\mathbf{x}} (T(\mathbf{x}) - \overline{T})^2}$$

Using equation above, NCC similarity function can be rewritten as:

$$NCC(\mathbf{p}) = \frac{\mathbf{a}}{\mathbf{b} \cdot \mathbf{c}} \tag{2.13}$$

Firstly, gradient of NCC is calculated using first derivative of (2.13) with respect to motion parameters as:

$$g = \frac{\partial}{\partial \mathbf{p}} \left[ \frac{\mathbf{a}}{\mathbf{b} \cdot \mathbf{c}} \right] = \frac{\mathbf{a}' \cdot (\mathbf{b} \cdot \mathbf{c})}{(\mathbf{b} \cdot \mathbf{c})^2} - \frac{\mathbf{b}' \cdot (\mathbf{a} \cdot \mathbf{c})}{(\mathbf{b} \cdot \mathbf{c})^2} - \frac{\mathbf{c}' \cdot (\mathbf{a} \cdot \mathbf{b})}{(\mathbf{b} \cdot \mathbf{c})^2} \tag{2.14}$$

Template is not changing during the iterations so **c**' is directly equal to zero. Due to this reason, the right most term of the equation above can be directly eliminated and gradient can be calculated as:

$$g = \frac{\mathbf{a}'}{\mathbf{b} \cdot \mathbf{c}} - \frac{\mathbf{b}' \cdot \mathbf{a}}{\mathbf{b}^2 \cdot \mathbf{c}} \tag{2.15}$$

**a**' and **b**' are vectors that contain the derivative of **a** and **b** with respect to motion parameters (2.16).

$$\mathbf{a}' = \left[ \frac{\partial \mathbf{a}}{\partial \mathbf{p}_1}, \frac{\partial \mathbf{a}}{\partial \mathbf{p}_2}, \ldots, \frac{\partial \mathbf{a}}{\partial \mathbf{p}_6} \right] \; and \; \mathbf{b}' = \left[ \frac{\partial \mathbf{b}}{\partial \mathbf{p}_1}, \frac{\partial \mathbf{b}}{\partial \mathbf{p}_2}, \ldots, \frac{\partial \mathbf{b}}{\partial \mathbf{p}_6} \right] \tag{2.16}$$

Each element of **a**' and **b**' can be computed as:

$$\frac{\partial \mathbf{a}}{\partial \mathbf{p}_v} = \sum_{\mathbf{x}} \left[ \frac{\partial I(\mathbf{w}(\mathbf{x},\mathbf{p}))}{\partial \mathbf{p}_v} - \frac{1}{\mathbf{N}} \sum_{\mathbf{x}} \frac{\partial I(\mathbf{w}(\mathbf{x},\mathbf{p}))}{\partial \mathbf{p}_v} \right] \cdot [T(\mathbf{x}) - \overline{T}]$$

$$\frac{\partial \mathbf{b}}{\partial \mathbf{p}_v} = \frac{1}{\mathbf{b}} \sum_{\mathbf{x}} \left[ \frac{\partial I(\mathbf{w}(\mathbf{x},\mathbf{p}))}{\partial \mathbf{p}_v} - \frac{1}{\mathbf{N}} \sum_{\mathbf{x}} \frac{\partial I(\mathbf{w}(\mathbf{x},\mathbf{p}))}{\partial \mathbf{p}_v} \right] \cdot [I(\mathbf{w}(\mathbf{x},\mathbf{p})) - \overline{I}]$$

Secondly the hessian of the NCC is computed with respect to motion parameters and the matrix form of Hessian is shown below:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial p_1 \partial p_1} & \frac{\partial^2 f}{\partial p_1 \partial p_2} & \cdots & \frac{\partial^2 f}{\partial p_1 \partial p_6} \\ \frac{\partial^2 f}{\partial p_2 \partial p_1} & \frac{\partial^2 f}{\partial p_2 \partial p_2} & \cdots & \frac{\partial^2 f}{\partial p_2 \partial p_6} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial p_6 \partial p_1} & \frac{\partial^2 f}{\partial p_6 \partial p_2} & \cdots & \frac{\partial^2 f}{\partial p_6 \partial p_6} \end{bmatrix} \tag{2.17}$$

Each element of **H** is computed as:

$$\begin{aligned} H_{uv} = & \frac{1}{\mathbf{b} \cdot \mathbf{c}} \frac{\partial^2 \mathbf{a}}{\partial \mathbf{p}_u \partial \mathbf{p}_v} - \frac{1}{\mathbf{b}^2 \cdot \mathbf{c}} \frac{\partial \mathbf{a}}{\partial \mathbf{p}_v} \frac{\partial \mathbf{b}}{\partial \mathbf{p}_u} - \frac{1}{\mathbf{b}^2 \cdot \mathbf{c}} \frac{\partial \mathbf{a}}{\partial \mathbf{p}_u} - \\ & \frac{\mathbf{a}}{\mathbf{b}^2 \cdot \mathbf{c}} \frac{\partial^2 \mathbf{b}}{\partial \mathbf{p}_u \partial \mathbf{p}_v} + \frac{\mathbf{a}}{\mathbf{b}^3 \cdot \mathbf{c}} \frac{\partial \mathbf{b}}{\partial \mathbf{p}_u} \frac{\partial \mathbf{b}}{\partial \mathbf{p}_v} \end{aligned} \tag{2.18}$$

The first derivative of **a** and **b** can be computed using (2.17) and second derivative of them can be computed as:

$$\frac{\partial^2 \mathbf{a}}{\partial \mathbf{p}_u \partial \mathbf{p}_v} = \sum_{\mathbf{x}} \left[ \frac{\partial^2 I(\mathbf{w}(\mathbf{x}, \mathbf{p})}{\partial \mathbf{p}_u \partial \mathbf{p}_v} - \frac{1}{\mathbf{N}} \sum_{\mathbf{x}} \frac{\partial^2 I(\mathbf{w}(\mathbf{x}, \mathbf{p})}{\partial \mathbf{p}_u \partial \mathbf{p}_v} \right] \cdot [T(\mathbf{x}) - \overline{T}]$$

$$\begin{aligned} \frac{\partial^2 \mathbf{b}}{\partial \mathbf{p}_u \partial \mathbf{p}_v} = & -\frac{1}{\mathbf{b}^2} \frac{\partial \mathbf{b}}{\partial \mathbf{p}_u} \sum_{\mathbf{x}} \left[ \frac{\partial I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_v} - \frac{1}{\mathbf{N}} \sum_{\mathbf{x}} \frac{\partial I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_v} \right] \cdot \\ & [I(\mathbf{w}(\mathbf{x}, \mathbf{p})) - \overline{I}] \\ & + \frac{1}{\mathbf{b}} \sum_{\mathbf{x}} \left[ \frac{\partial^2 I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_v \partial \mathbf{p}_u} - \frac{1}{\mathbf{N}} \sum_{\mathbf{x}} \frac{\partial^2 I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_v \partial \mathbf{p}_u} \right] \cdot \\ & [I(\mathbf{w}(\mathbf{x}, \mathbf{p})) - \overline{I}] \\ & + \frac{1}{\mathbf{b}} \sum_{\mathbf{x}} \left[ \frac{\partial I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_v} - \frac{1}{\mathbf{N}} \sum_{\mathbf{x}} \frac{\partial I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_v} \right] \cdot \\ & \sum_{\mathbf{x}} \left[ \frac{\partial I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_u} - \frac{1}{\mathbf{N}} \sum_{\mathbf{x}} \frac{\partial I(\mathbf{w}(\mathbf{x}, \mathbf{p}))}{\partial \mathbf{p}_u} \right] \end{aligned} \tag{2.19}$$

Parameters, **p**, are iteratively calculated. Before iterations start, **H** and its inverse are computed. In the each iteration, **g** is firstly calculated and then $(\Delta \mathbf{p})$ is calculated using (2.12). Iterations last until $(\Delta \mathbf{p})$ become smaller than predefined threshold value, $\epsilon$, or the maximum iteration number is reached.

## 2.2 Template Update Strategy

In this section, the needle template update strategy is described. It is one of the significant steps of the needle tip tracking. If the needle template is not updated correctly, the needle tip is lost and the tracking fails. In order to update the needle template correctly, three commonly used strategies are evaluated below.

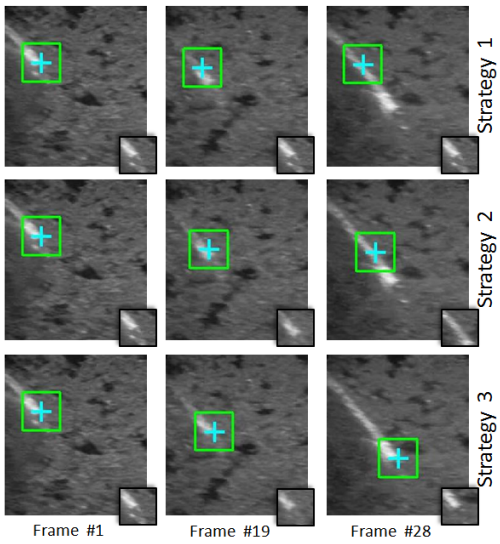



**Fig. 2.2.:** Comparison of the template update strategies. **Strategy 1**: No update. **Strategy 2**: Template updated in each frame. **Strategy 3**: Template update with drift correction.

**Strategy 1: No Update.** The template is not updated during the tracking ($T_{n+1}(\mathbf{x}) = T_1(\mathbf{x})$). However, this strategy is not suitable for needle tip tracking because the appearance of the needle shape changes during the needle insertion. Also, sudden changes in the needle pixels' intensities might occur due to misalignment between the needle and the US probe. Due to these shortcomings, the template image must be updated in every frame.

**Strategy 2: Naive Update.** The template is updated in every frame ($T_{n+1}(\mathbf{x}) = T_n(\mathbf{x})$). Hence, the template becomes invariant to appearance changes during the tracking. After $\mathbf{p}$ is calculated between two images, there is a residue error between template and current images. This error is accumulated during the tracking and template is drifting along motion direction. Eventually, needle tip tracking fails. Similar to Strategy 1, Strategy 2 is not suitable for needle tip tracking because the template is drifting. Therefore, the drift must be corrected to prevent failures during the tracking.

**Strategy 3: Template Update with Drift Correction.** In this Strategy, template is updated in every frame like Strategy 2. But, new template is then registered with the first template of the needle [6]. Hence, the residue error due to image registration between the template and the current images is so minimized that the drift is corrected. This method improves the tracking accuracy significantly and prevents tracking failure. Also, it is a simple method and can be easily implemented. All of the Strategies are compared using 2D US frames while the needle is inserted into agar-gelatine based phantom with a needle insertion robot. The outputs of the Strategies are shown in Fig. 2.2.

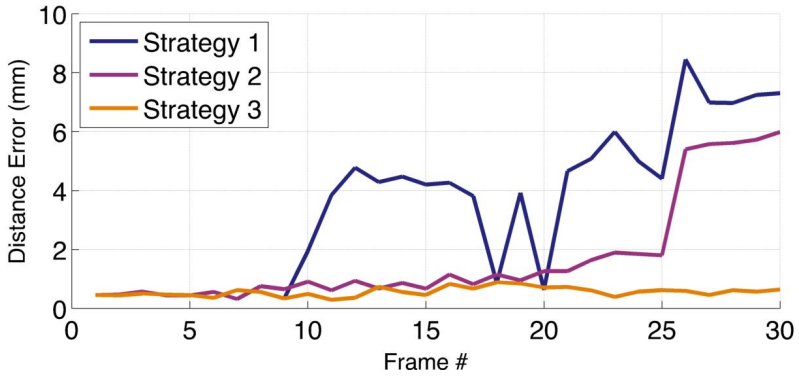



**Fig. 2.3.:** Euclidean distance error versus frame number for template update strategies.

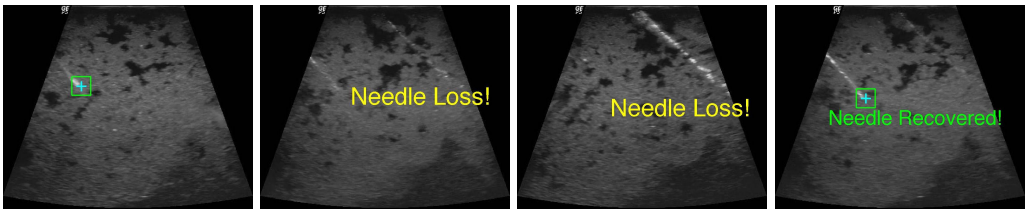
## 2.3 Needle Loss Detection & Recovery

While inserting the needle, there can be misalignment between the US image plane and the needle. In this situation, the needle is in the out-of-US image plane so the needle is lost in the US image. During the needle tip tracking, the needle template is updated in each frame and then the drift is corrected (Strategy 3). If the needle tip loss is not detected, template still will be updated and the updated template won't contain the needle structure. Eventually, tracking will fail.

In order to prevent tracking failure, the target loss must be detected. For this purpose, similarity between current template image ($T_n(\mathbf{x})$) and image obtained after the tracking ($T_{n+1}(\mathbf{x})$) can be used. In our experiments, SSD, SCV, NCC, and MI are evaluated. However, values of SSD, SCV, and MI are not normalized and also thresholding value for detecting target loss using these measures varies by image. On the other hand, NCC value is normalized. Its value is changing between 1 and -1. Also, it is a robust similarity measure (see Fig. 2.1). NCC between $T_n(\mathbf{x})$ and $T_{n+1}(\mathbf{x})$ can be computed as:

$$NCC = \frac{\sum_{\mathbf{x}}(T_{n+1}(\mathbf{x})) - \overline{T}_{n+1}) \cdot (T_n(\mathbf{x}) - \overline{T}_n)}{\sqrt{\sum_{\mathbf{x}}(T_{n+1}(\mathbf{x})) - \overline{T}_{n+1})^2} \cdot \sqrt{\sum_{\mathbf{x}}(T_n(\mathbf{x}) - \overline{T}_n)^2}}$$

where $\overline{T}_{n+1}$ and $\overline{T}_n$ are the mean intensity values of $T_{n+1}(\mathbf{x})$ and $T_n(\mathbf{x})$, respectively. If the NCC value is 1, images are identical whereas if the NCC value is -1, images are completely irrelevant. In our experiments, we observed that if the needle is in the US image plane, NCC value is very close to 1. In contrast, if the needle is lost in the image plane, NCC value drops significantly (see Fig. 2.5). During the needle tracking, 0.8 is determined as threshold value for detecting the target loss.




(a) Frame #31 (b) Frame #37 (c) Frame #52 (d) Frame #56

**Fig. 2.4.:** Result of the needle loss detection & recovery. (a) The needle tip is being visually tracked and the needle is in the image plane. (b)-(c) The needle tip is in out-of-the US image plane and needle loss is detected. (d) The needle tip is again in the US image plane and the needle tip is detected using the last template image of the needle before it got lost.

If the needle loss is detected once, the needle insertion is stopped and current template is not updated so the last appearance of the needle is kept. Then, the needle is searching around last needle tip points. This process is applied to each frame until NCC value reach the 0.75. If NCC value reaches 0.75, the needle is recovered and tracking continues. The example output of the needle loss & recovery is shown in the Fig. 2.4 and NCC values versus frame number including the needle loss & recovery images in Fig. 2.4 is shown in Fig. 2.5.

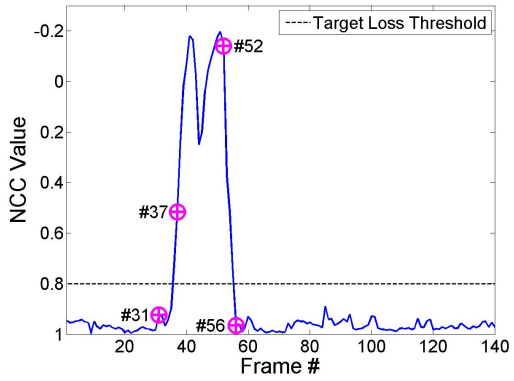


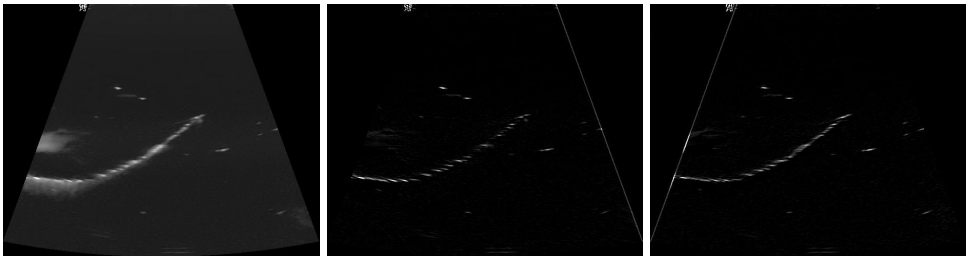

**Fig. 2.5.:** NCC value versus frame number. The NCC value of each frame illustrated in Fig. 2.4 is marked with '⊕' and the corresponding frame number is indicated.

# 3 Curve Gabor Localization

This chapter presents an algorithm for the segmentation of curved biopsy needles in 2D Ultrasound images using Gabor filter. In the previous needle localization algorihm, needle is assumed as a straight line. However, in the biopsy procedures flexible needles are also used. To detect the curved needle correctly, previously proposed Gabor filter based needle localization algorithm should be modified.

In the proposed needle localization method, Gabor filter is used to segment needle pixels. Gabor filter is applied with an orientation angle to segment line shaped needles pixels. In this situation, orientation of needle pixels are very close to with each other. However, orientation of curve shape needle pixels are not equal to with each other. Hence, if Gabor filter is applied with a single orientation angle to segment curved shape needles, only some parts of needle pixels are enhanced while the other parts are suppressed due to orientation of needle pixels. In order to segment all needle pixels of curve shaped needles, Gabor filter is applied with two different orientation angles $(\alpha_1, \alpha_2)$ and then filtered images are summed. As a result of this, all pixels of curved shape needles are segmented. After the segmentation process, summed image is smoothen using a 7x7 sized Median filter and smoothen image is binarized with entropy based parameter tuning method. In our experiments, $\alpha_1 and \alpha_2$ are selected as $135°$ and $225°$, respectively because pixels with opposite directions can be segmented using these angles. Now, needle axis can be localized.

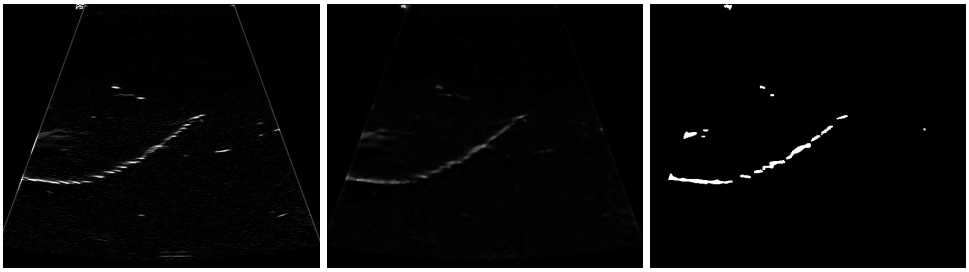
According to solid mechanics, needle curvature can be modeled with a third order polynomial. In order to localize needle axis, polynomial fitting can be applied. However, binarized image does not consists of pure needle pixels. There can be noise in the binarized image so polynomial fitting is not a good method to localize the needle axis. RANSAC curve fitting can be used but it is hard to generalize RANSAC for higher dimensions. Fitting a line to binarized pixels using RANSAC is a robust fitting method because it is a simple parametric model. Needle axis can be localized using RANSAC line fitting although geometrical model of needle is curve. For this purpose, binarized image is divided into grids. In each grid, geometrical model of binarized pixel is line (see Fig. 3.1 - (g)). Hence, RANSAC line fitting is applied to each grid and then needle axis is localized with gathering detected lines. After localizing the needle axis, needle tip estimation part (#2st Stage - Fig.1.1) is directly used because this method works with geometrical model. If the geometrical model is known, needle tip can be estimated using probability mapping method. The steps



(a) Raw image of curve needle in distilled water

(b) Gabor filter with $\alpha_1 = 135°$

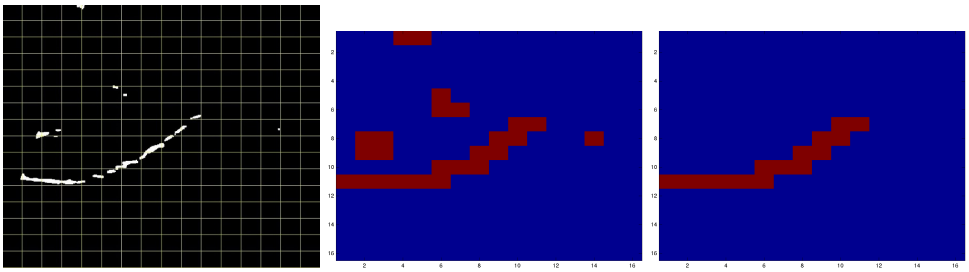
(c) Gabor filter with $\alpha_2 = 225°$



(d) Sum Image

(e) Median filter Output

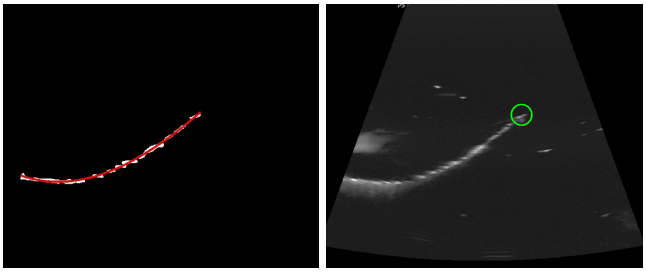
(f) Image binarization



(g) Gridization

(h) Components

(i) Largest Component



(j) Axis localization

(k) Tip estimation

**Fig. 3.1.:** Curve needle axis and its tip localization in 2D US images. (a) Curve needle is imaged in water medium. (b) & (c) Gabor filter are applied with $135°$ and $225°$, respectively. (d) Gabor filter outputs in (b-c) are summed. (e) Median filter is applied to summed image. (f) Median filter output is binarized and morphological operations is applied. (g) Binarized image is divided into grids for needle axis localization. (h) Needle axis is localized by applying RANSAC line fitting to each grid. (i) Needle tip is estimated using probability mapping method.

of proposed localization method for curved shaped needles are shown in Fig. 3.1 and 3D probability map is shown in Fig. 3.2.

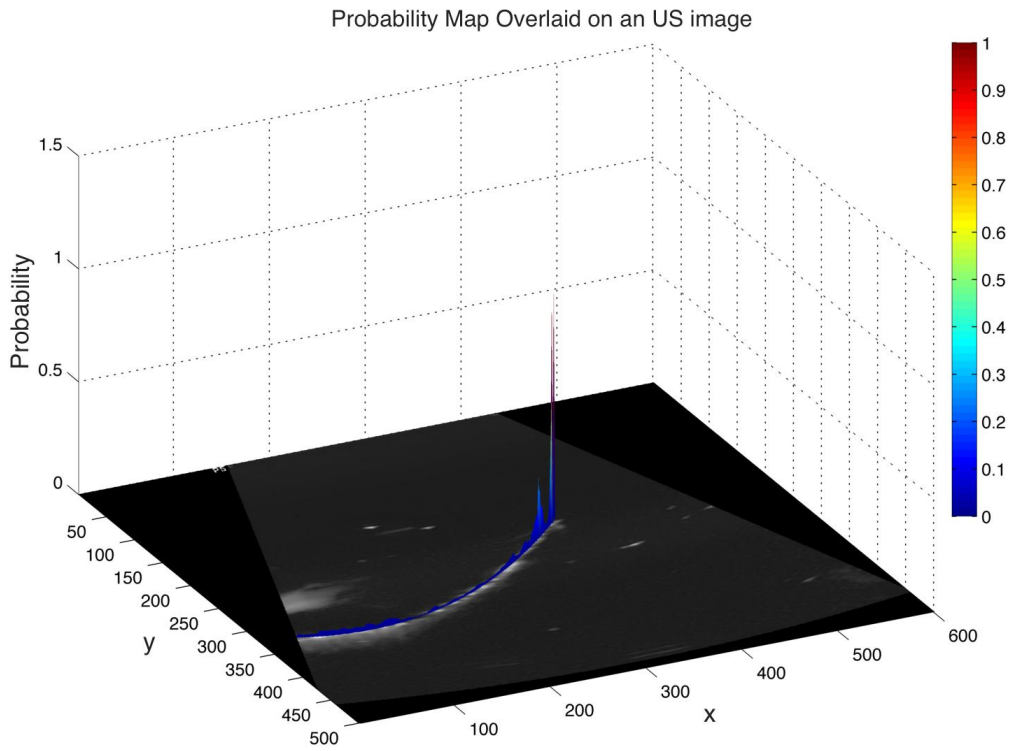
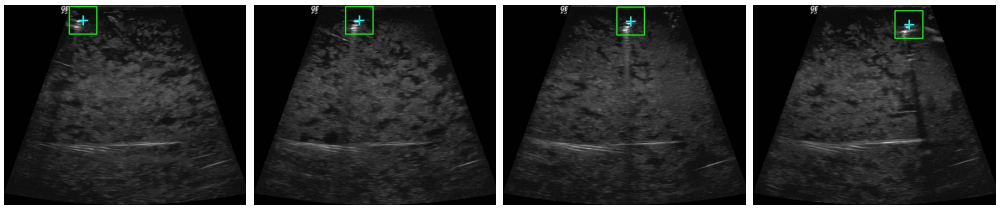



**Fig. 3.2.:** 3D probability map overlaid onto 2D US image shown in Fig. 3.1-(a).

# 4 3D Needle Shape Visualisation

3D needle shape visualization is an important process during the percutaneous needle procedures. In the tissue, needle encounters with external forces which cause bending so the target can be missed or tissue damage might occur. Also, the needle tip reaches the target by bending the needle in a controlled manner in the needle steering. For these reasons, 3D needle shape is required to prevent side effects and get the precise path of the needle tip for needle steering.

3D needle shape can be obtained using external sensors. For this purpose, optical and electromagnetic tracking systems and fiber bragg grating sensor can be used [**FGB**]. However, shape estimation using external sensor is expensive and also requires an external setup. On the other hand, percutaneous needle procedures are generally carried out with the guidance of medical imaging devices, mostly with 2D US. Hence, 3D needle shape can be estimated from 2D US images without using any external sensors.



(a) Frame #1 (b) Frame #35 (c) Frame #54 (d) Frame #65

**Fig. 4.1.:** Results using visual tracking of the needle tip in 2D transverse US images.

In 2D lateral US images, needle shape is very close to a straight line so needle curvature can not be observed. For this reason, needle must be transversely scanned to visualize the needle curvature in 3D. In the 2D transverse US images, the needle has a specific pattern which is called comet tail artifact (CTA). In the literature, 3D needle shape was previously obtained from US images using CTA pattern [9], [10]. In these studies, firstly, 2D US probe was moved along the needle insertion direction and CTA is detected in each frame. Then, a polynomial function was fitted to acquired data. However, this method is very sensitive to sudden changes because the needle tip points are acquired with CTA detection algorithm. In this case, if CTA detection algorithm fails, unrelated points can be detected as the needle. In order to estimate 3D needle shape more accurately, a visual tracking based method is proposed.

In this method, like other methods, US probe was moved along the needle insertion direction. CTA pattern is localized or marked in the first frame. Then, this pattern is visually tracked along the motion and the needle points are acquired. In the final process, a polynomial function is fitted to acquired data. The schematic representation of this method is depicted in Fig. 4.2. According to the solid mechanics, the

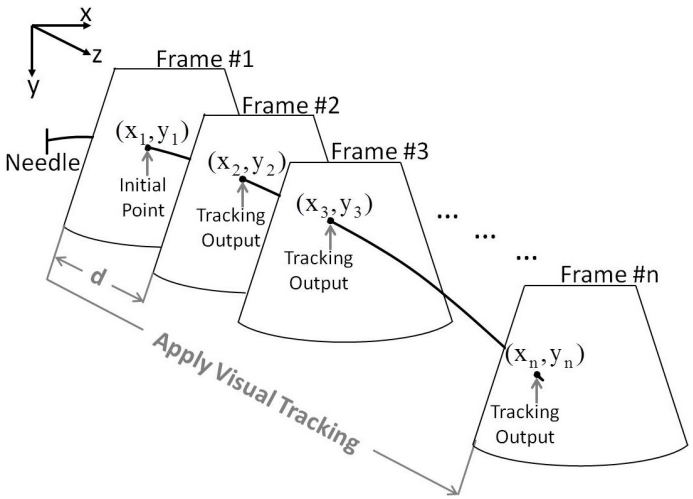



**Fig. 4.2.:** Conceptual representation of 3D needle shape estimation from 2D transverse US images using visual tracking.

needle curvature can be modelled using cubic polynomial function. Hence, cubic polynomial function is chosen to model the 3D needle curvature. Fig. 4.1 shows the results of CTA pattern tracking in 2D transverse US images. In each frame, needle locations in $x-$ and $y-$ axes are acquired. Fig. 4.3 shows the 3D needle shape estimation with a cubic polynomial fitting. As seen in Fig. 4.3, needle curvature which is obtained by the visual tracking of CTA pattern in 2D transverse images are precisely visible. It is shown that the proposed method is a more effective way to estimate 3D needle shape in 2D transverse US images compared to other methods in the literature.

## 4.1 3D Needle Shape Visualization Accuracy

In order to evaluate the accuracy of proposed 3D needle shape visualization method, CT scanner (YXLON MU2000-D) was used (see Appendix B). The accuracy of CT scanner is 10 $\mu$m so 3D needle shape was obtained accurately. Before the scanning with CT and US, a mold was fabricated to fix the needle. Hence, needle was scanned with CT and US without breaking. The mold was fabricated using a 3D printer. In order to determine reference coordinate system, three cylindrical steel shafts (diameter = 4 mm, length = 20 mm) were attached to corners orthogonally as fiducial markers. They are shown in Fig. 4.4.

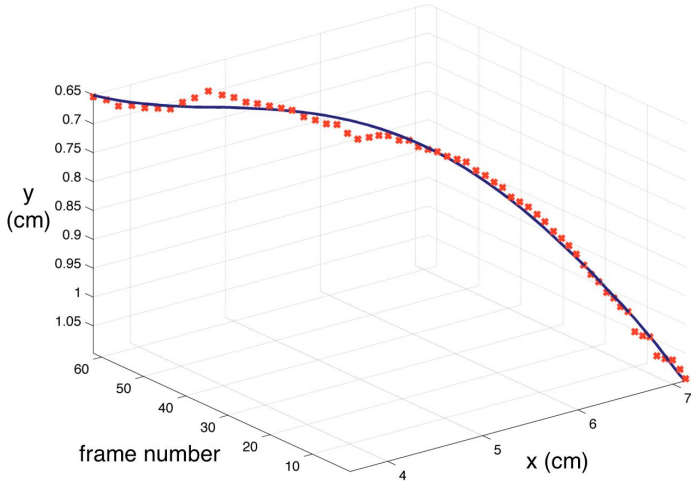



**Fig. 4.3.:** Needle is visually tracked along the needle path in Fig. 4.1 and 3D needle shape is visualized from 2D transverse US images by fitting cubic polynomial function.

The needle was firstly scanned in CT and 3D needle shape was obtained. After the CT scan, needle was embedded into water medium. The US probe was attached to a robotic arm and needle was scanned along the needle path. In the scanning process, the velocity of robotic arm was set to 10 mm/s. CTA pattern was tracked in each frame and location of the needle was obtained. Then, cubic polynomial was fitted to obtained needle locations. At the final stage, 3D needle shapes obtained from CT and US were combined in the same coordinate system (see Fig. 4.4). By combining two shapes in the same coordinate system, euclidian distance between the shapes can be calculated. Euclidian distance error between the shapes of CT and US scans is $1.85 \pm 1.60$ mm.

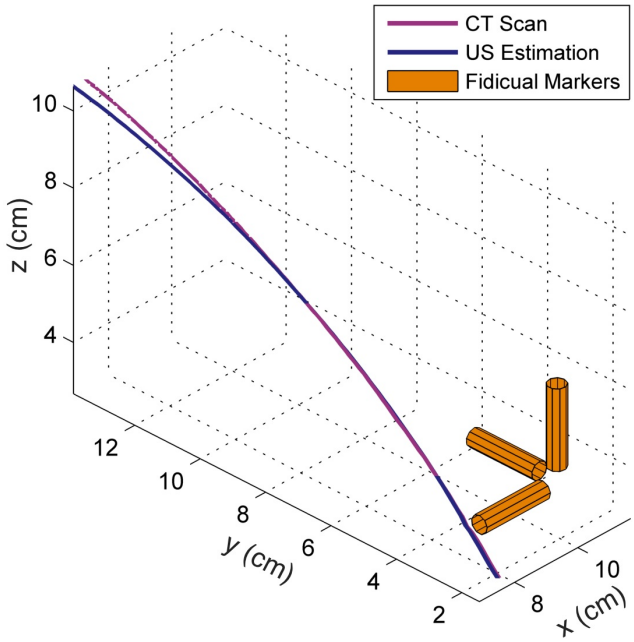



**Fig. 4.4.:** Comparison of 3D needle scan with CT and 3D needle shape estimation from 2D transverse US images.

# 5 Implementations of Algorithms

## 5.1 Implementation Details

Needle Tip Localization, tracking with SSD and tracking with NCC algorithms are implemented using C++ and OpenCV 3.0. CMake 3.3 is used for building purposes. For version control git is used.

Algorithms software includes two main packages.

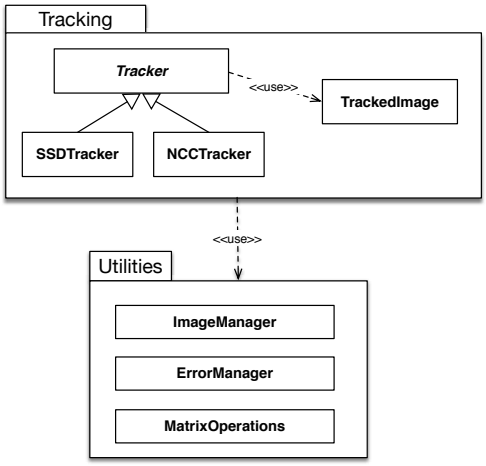



**Fig. 5.1.:** Basic tracking software architecture.

Fig. 5.1 shows the main software architecture of the tracking operations. There is also another needle tip localization package. However, it is not explained here since it has been developed out of the scope of this project.

The **Tracking** and **Utilities** packages are explained below.

**Tracking**
Classes in this package provide the tracking functionality for both SSD and NCC based algorithms. **Tracker** is an abstract class which defines the basic functionalities for the **SSDTracker** and **NCCTracker**. Algorithms that each tracker implements are explained previously in sections 2.1.1 and 2.1.2, respectively.

A **TrackedImage** class is implemented in order to encapsulate the tracked frame's information in order to easily use iwhile tracking following frames and calculating error.

```
class TrackedImage {
   public:
      TrackedImage(cv::Mat image, double x, double y, int w, int h);
      cv::Mat image;
      double trackedX;
      double trackedY;
      int width;
      int height;
};
```

While applying a tracking algorithm, firstly a specific tracker is generated.

```
      Tracker* tracker = new NCCTracker();
```

or

```
      Tracker* tracker = new SSDTracker();
```

For the initial frame, a TrackedImage object is created with the manually defined initial x and y locations. For each frame, tracking takes place by the current tracker. Each completed tracking iteration for every frame returns a TrackedImage object which contains the new position values. A TrackedImage also carries the information like width and height so that the tracker can handle the differences when a parameter changes between consecutive frames somehow.

On the other hand, every time a new image is tracked without a target loss, x and y parameters updated and therefore template is cropped in the next frame correctly.

```
    TrackedImage tracked = tracker->track(imageToTrack,
        previousTrackedImage);

      //update the locations
      initx = trackedImage.trackedX;
      inity = trackedImage.trackedY;
```

For SSD and NCC tracking, mathematical details are given in section **??**. Pseudocodes of these algorithms are as following:

**Algorithm 1** SSDTracking

**Data**: Template Image T
**Result**: Motion Parameters $\Delta p$
**for** *Each image I* **do**
  **while** $\Delta p > \epsilon$ *and not reached to maximum iterations* **do**
    compute **Jacobian**;
    compute $\Delta p$;
  **end**
**end**

**Algorithm 2** NCCTracking

**Data**: Template Image T
**Result**: Motion Parameters $\Delta p$
Compute Hessian Matrix **H**;
**for** *Each image I* **do**
  **while** $\Delta p > \epsilon$ *and not reached to maximum iterations* **do**
    compute **G**;
    compute $\Delta p$;
  **end**
**end**

### Utilities

Utilities package implements **ImageManager**, **MatrixOperations** and **ErrorManager**.

**ImageManager** is used for acquiring images from given paths. Returning the grayscale images and image save operations are responsibilities of this class.
**MatrixOperations** is the namespace package which is used for calculating gradients and handling meshgrid operations. It is important to separate the matrix operations in order to easily provide extensions when new algorithms arrived to be implemented.
**ErrorManager** takes an template image and warped image to calculate the NCC value between them. This manager defines the error thresholds and is used to detect when there is a target loss or not.

# 6 A Standalone Application

Needle tip localization and tracking algorithms and their implementations are explained in chapters 2 and 5, respectively. In order to provide a better and faster user experience, a standalone application is developed.

Our goals in developing this standalone application can be summarized as follows:

1. For researchers: Being able to try proposed localization and tracking algorithms easily for different datasets with different parameters as well as getting detailed reports

2. For medical field: To be able to use algorithms in real robotic assisted biopsy operations

**Qt Library** is used to develop a multi-platform application which can deliver the features with a feasible user interface. Next section will provide brief information about Qt library and then the implementation details and the design of the software will be discussed.

### 6.0.1 Qt Library and Why Qt?

Qt is a comprehensive framework for cross-platform application development. It is possible to develop for different operating systems(Mac OS, Windows, Linux, iOS) as well as different hardwares such as pc's, mobile phones and more. Qt Project has both commercial and open-source versions. Open-source one is available under GPL-v3 [1] , LGPL v3 and LGPL v2 [2] .

#### Why Qt?

There are different frameworks for cross-platform application development such as FLTK and GTK. When a comparison is made between them, Qt outscored the others. The reasons are as follows:

1. Qt has extensive and detailed documentation and bigger community support

[1] GNU General Public Licence - https://www.gnu.org/licenses/gpl.html
[2] GNU Lesser General Public Licence - https://www.gnu.org/licenses/lgpl.html

2. For all platforms Qt provides native development

3. Qt is a more generic framework which is not only developed for GUI development but also can be used for other modules such as multimedia support etc.

## 6.1 Requirements

The application has different requirements for tracking and localization parts along with some common operations such as importing and exporting. This functional and non-functional quality requirements are clearly defined in the next part.

### 6.1.1 Functional Requirements

This requirements are the functional features that must be offered by the standalone application. They ensure the usability of the program and makes it possible to use in real-world applications.

#### General Requirements

These general requirements are common for both localization and tracking algorithms. They are used throughout the software.

**Opening Image Files** The application shall be able to open single or multiple image files.

**Opening Video Files** The application shall be able to import frames from video files.

**Displaying Thumbnails** User interface shall allow user to see multiple frames as thumbnails and select them accordingly. Thumbnails should be updated when a change in an image occur.

**Export Image Files** The user shall be able to export and save single or multiple images.

**Export Video Files** The user shall be able to export and save video files from modified image frames or videos.

**Reset/Clear Options** User shall be able to reset modified frames to initial states or clear all imported or modified data from the application.

### Localization Requirements

This requirements are defined for the needle tip localization specific parts of the software.

**Setting Parameters** Localization algorithm includes parameters for: initial Gabor insertion angle, region of interest width and height and RANSAC line fitting iteration count. The user interface shall provide input areas for user to change these parameters.

**Localizing a single image** User shall be able to apply localization only to a single image.

**Localizing multiple images** User shall be able to apply localization algorithms to multiple frames or a video.

**Localization Details Display** After a successful localization, application shall show localization steps in a different interface.

**Exporting outputs** User shall be able to export the images of intermediate localization steps as well as the final localized image.

### Tracking Requirements

This requirements are defined for the needle tip tracking specific parts of the software.

**Setting Parameters** Tracking algorithm includes parameters for: initial point, region of interest width and height, target loss threshold, target recovery threshold and tracker type (SSD, NCC). The user interface shall provide input areas for user to change these parameters.

**Tracking sequences** User shall be able to track needle tip using frame sequences.

**Targetloss Detection** Application should detect when the target is lost and display a notification.

**Target recovery Detection** Application should detect when the target is acquired and display a notification.

**Tracking Details Display** During the tracking another window should display a real time error plot.

**Error Plot Acquisition** Application should provide options the export current error plot.

### 6.1.2 Non Functional Requirements

This requirement are defined to ensure the software's reliability and robustness.

**Working With Different Types** Application should be able to open .png and .jpg images as well as .avi videos

**Exporting different types** Application should offer the export options for .png and .jpg images as well as .avi videos

**Working Performance** All algorithms should work in real-time.

**Algorithm correctness** Algorithms shall correctly localize and/or track the needle tip.

## 6.2 User Interface

User interface of the program is developed using Qt Creator. Qt Creator is an Integrated Development Environment (IDE) which has a specific debugger and an integrated GUI designer.

In the following parts, details of the user interface and the design/programming decisions will be explained.

### 6.2.1 Design and Usability

User interface of the software aims to provide most of the available options without forcing user to traverse over high number of menus. For this purpose, main window is designed in a way that it can contain all available algorithm options.

#### Main Window

Fig. 6.1 represents the main window of the user interface. All operations are handled in that window and user can easily access most of the options they require during the process.

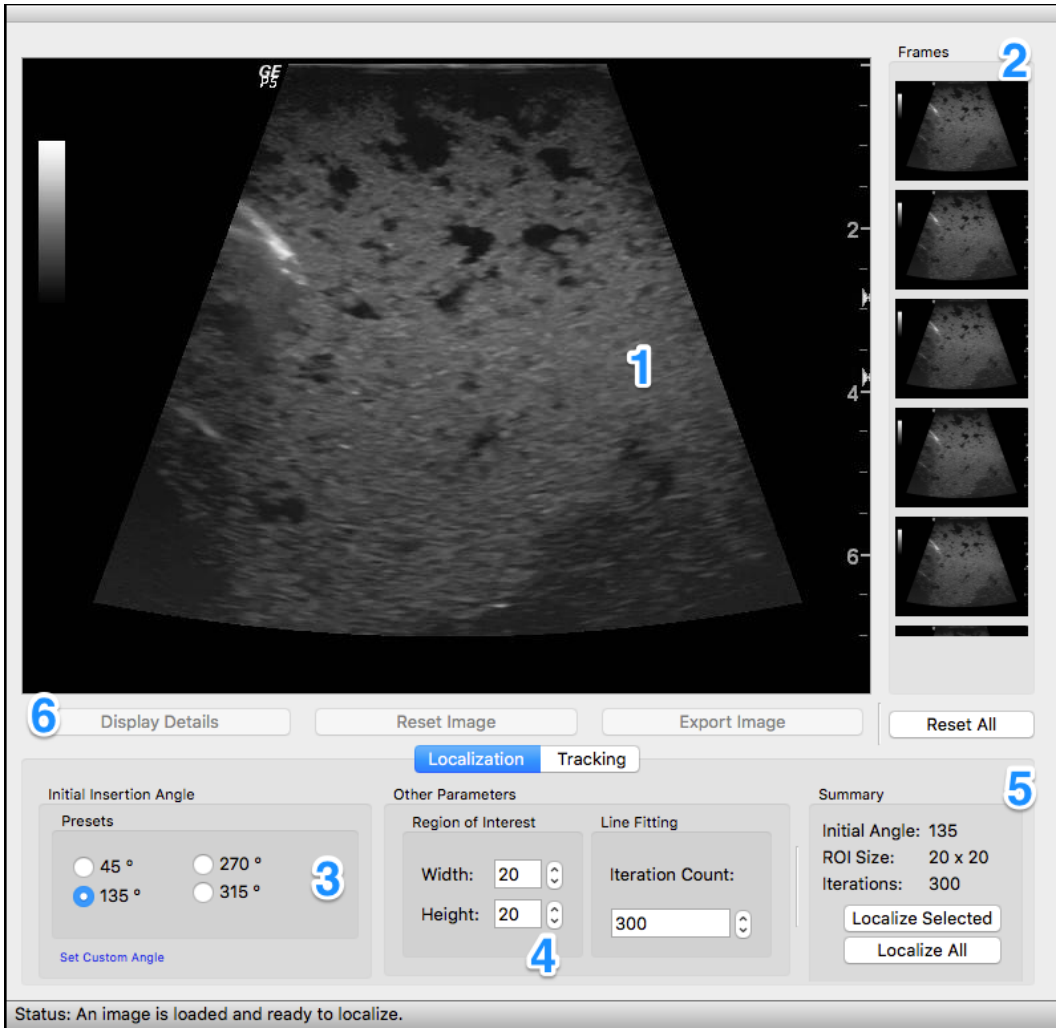



**Fig. 6.1.:** Main Application Interface with Localization Option

**1** This is the main image area. QGraphicsView component is used to display images. If user add multiple images, first image is shown as default.

**2** All images are displayed as thumbnails.

**3** As explained in theFig. 1.1, an initial estimation of the insertion angle is needed. There are four preset angles for the four quadrants of the rectangle. Users can select one according to the direction needle enters or select a custom value using the *Set Custom Angle* option.

**4** Users can specify the ROI size and the iteration count for RANSAC line fitting from this area.

**5** A summary of the selected parameters are shown in this area and user can run the localization algorithm using localization buttons.

6 This area becomes activated after an algorithm is applied. Users can see the detailed reports of results, reset the modified image to original and export the final result.

### Tracking Options Panel

Tracking options panel can be activated using the tabs at the bottom panel of main window. In the application, users can change tabs any time and try both localization and tracking algorithms for the same dataset.

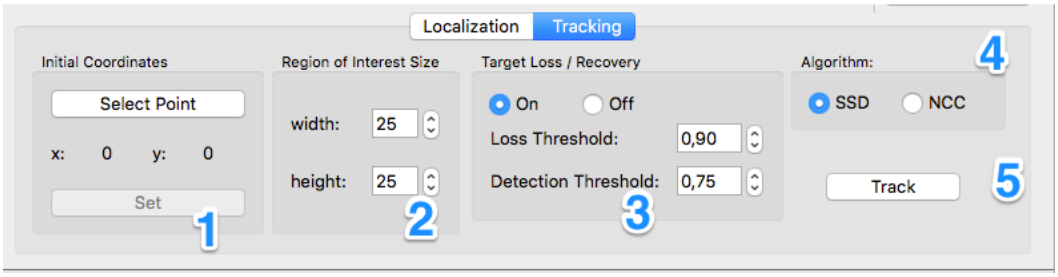



Fig. 6.2.: Tracking Options Panel

1 Initial point is selected using the graphics view. After selecting the point, user can finalize the selection process using this area.

2 This area provides the input for template region of interest size. This is one of the most important input information for the tracking process since the needle is visually tracked using this template and it is very size dependant.

3 Proposed needle target loss and recovery method is implemented and the threshold values can be specified from this area.

4 Selection of the algorithm type between the options of SSD based and NCC based tracking methods.

5 After selecting all parameters user can start the tracking process.

### Localization Details

Proposed needle tip localization algorithm consists of multiple steps as depicted in Fig. 1.1. Output of each step can be displayed separately and exported. Each localized image's information is ready and user can display using the 6. Panel explained in Fig.6.1.

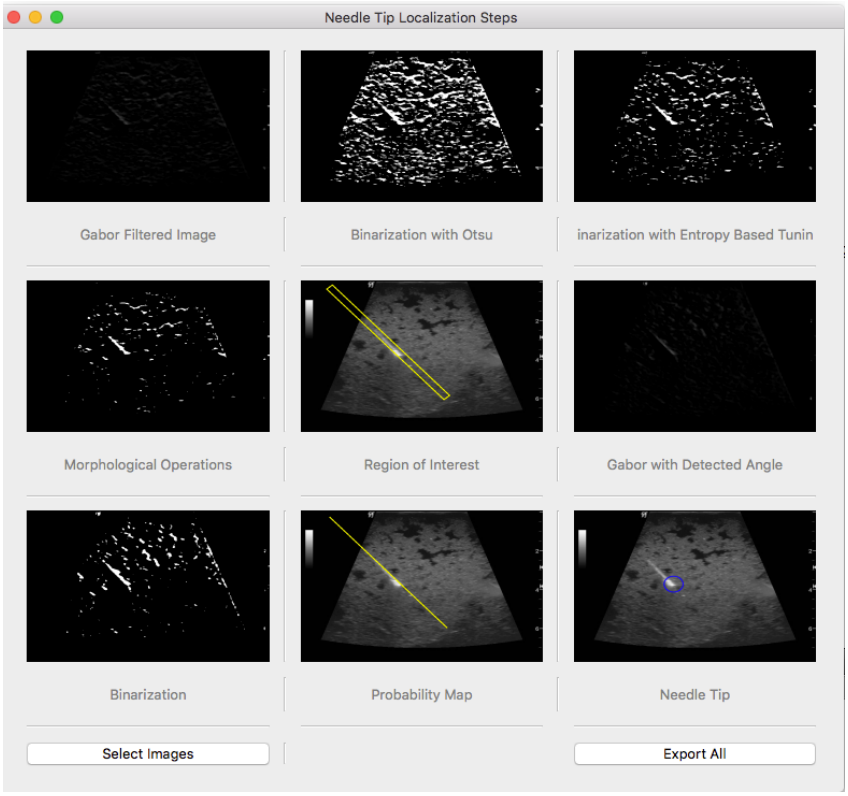



**Fig. 6.3.:** Localization Details Window. Each image displays a step of the localization algorithm

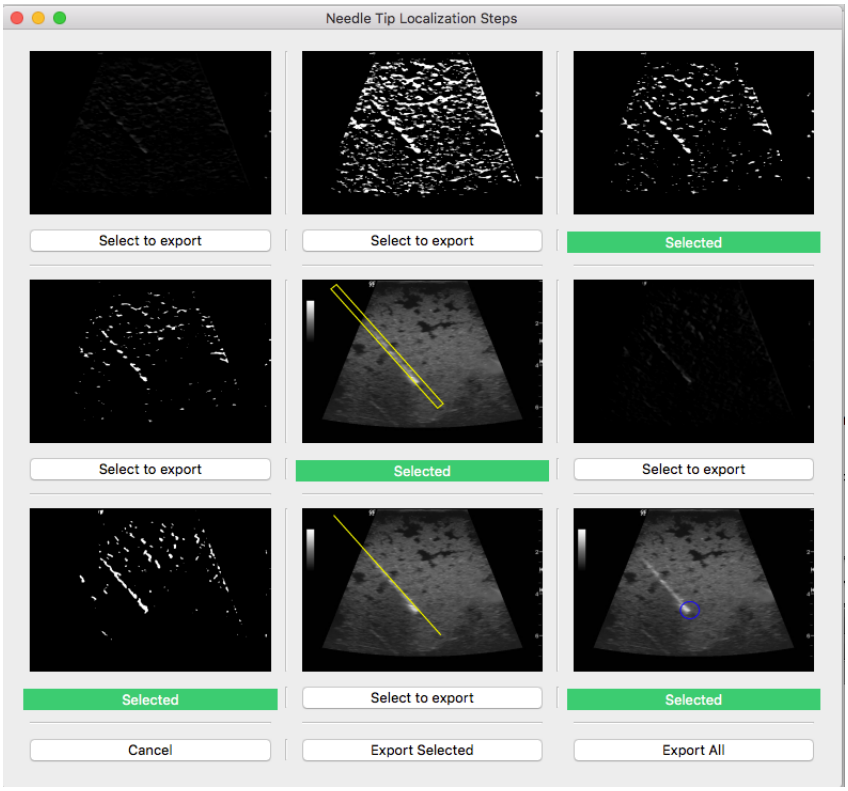
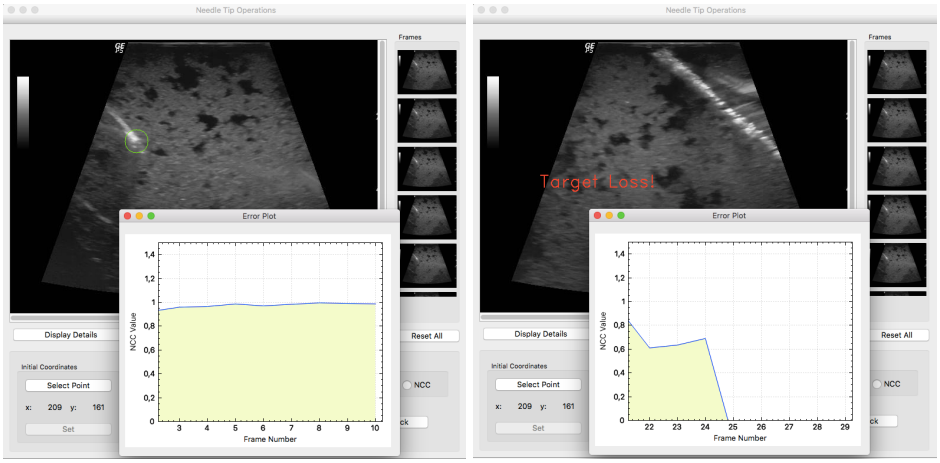



**Fig. 6.4.:** Localization Details with selections to export

### Tracking Details

Tracking part of the algorithm includes NCC value calculation between template and tracked image. This similarity value is used for loss and recovery detection. The value is normalized between 0 and 1. In the program, when user runs the tracking algorithm, real time error plot is also created.




(a) Regular Tracking (b) Target Loss

**Fig. 6.5.:** Tracking application with real-time NCC similarity plot (a) The NCC value close to 1 during the regular tracking (d) When the needle got lost, real time error plot reaches almost to zero and target loss notification is displayed

## 6.2.2 Software Architecture

Application software is divided into six main parts. GUI structure, utilities and algorithms are strictly separated to achieve lower dependency. Contents of these packages can be seen in Fig. 6.6.

### Modules

1. *UI* is the part of the program which is responsible from the view of the components. It includes *Main Window*, which is the main window explained in Fig. 6.1. *Localization Details Widget* and *Tracking Details Widget* are the classes which manage the visual structure of details windows of respective algorithms.

2. *UI Utilities* consists of *Main Window UI Manager* and *Tracking UI Manger* which are the components that work as controllers.

3. *Extensions* contains classes which improve some capabilities of Qt components. They are used for providing a clickable graphics view.

4. *Utilities* package contains classes for managing and transforming data in the program. *FileManager* is responsible from load and save operations as well as file conversions. *Frameset Manager* is used for managing frame groups and keeping track of the modified images. *Qt OpenCV Utilities* is the class which handles the conversions between OpenCV and Qt elements such as QImage - cv::Mat conversion.

5. *Algorithms* part is the part explained in the section 5 and they are completely embedded to the system.

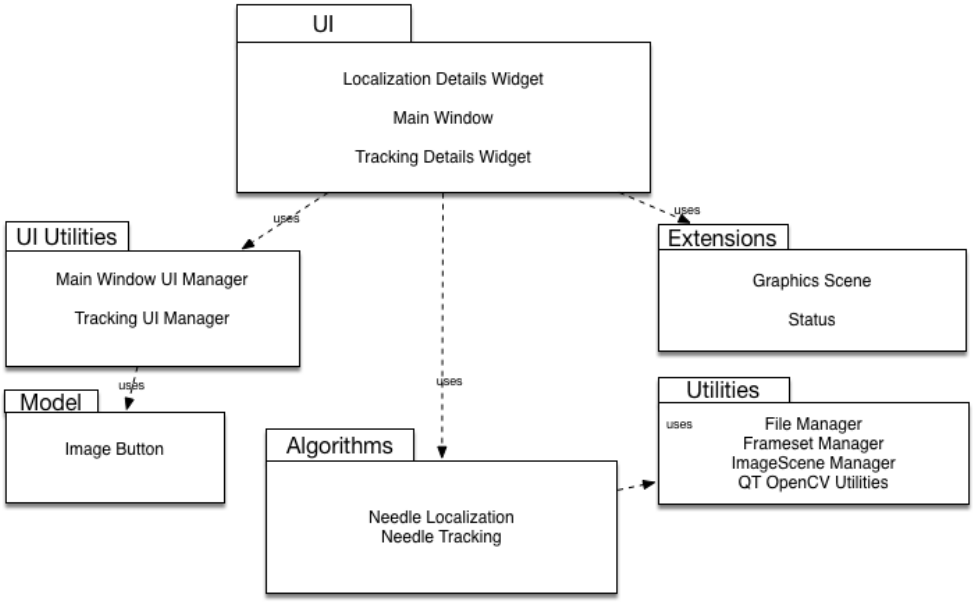



**Fig. 6.6.:** Structure of the Standalone Application

### 6.2.3 Discussion

Developed standalone application is a complete package for applying needle tip localization and tracking algorithms in real time. It provides the necessary options and meets with the requirements. Development with Qt is pretty straightforward, components are clear and well documented. Current application user interface design is more appropriate for research purposes. For enterprise usage, user interface and experience design should be improved. Extra screenshots and some implementation details can be found in Appendix A.

# 7 GPU

Graphical Processing Unit(GPU) is an electronic circuit which is generally used for graphics applications such as 3D Rendering and image creation. For highly parallelizeble tasks with high load such as 3D Rendering, GPUs increased the performance since they are capable of processing large amounts of data at the same time with their 100+ cores. There are vendors such as Nvidia and AMD which are producing GPUs ranging from home level devices to high-level devices like Nvidia Tegra.

#### General Purpose Computing on Graphics Processing Units

In modern software development era, we are using GPUs not only for graphical purposes but also for general-purpose computing. The concept of *General-purpose computing on graphics processing units (CPGPU)* is created by the idea of utilizing GPU's high number of cores and processing power for standard computations which normally are handled by CPU. CPGPU is a concept of software design instead of a hardware system. It is pipeline which we can use to communicate between CPU and GPU in order to allocate complex tasks to the graphical processing unit.

There are different API's for using GPUs in general purpose applications. These frameworks may support different devices and they have different types of limitations. Following part will briefly explain these frameworks.

## 7.1 Frameworks

#### CUDA

*CUDA* is a framework developed by NVIDIA to create an interface which is used for utilizing CUDA supported GPUs for general purpose processing. It is a software layer which makes it possible to access GPU's instructions to execute operations.

CUDA framework includes a *Driver* and a *toolkit*. *Driver* is the low-level software part of CUDA which manages the graphics card. *Toolkit* included components such as nvcc compiler and some libraries which are used for CUDA-compatible software development.

### OpenCL

OpenCL is a framework which aims to provide a computing interface for multiple platforms including CPUs, GPUs and more. With a C-like language, it is possible to create programs that can run on a computation device such as GPU.

CUDA is a NVIDIA spesific environment and it works only with CUDA-Enabled NVIDIA GPUs. However, since OpenCL is a open spesification for heterogeneous devices, it can be used with various types of CPUs and GPUs with different vendors as well as NVIDIA ones.

## 7.2 Programming Logic

The basic common logic behind CUDA and OpenCL programming is to allocate highly parallezible parts of the code to the GPU. In this systems, there are two separate processors with distinct memories. CPU is called the *host* and the GPU is called the *device*. Sequential part of the code executes in the host which is our CPU. On the other hand, parallelizable code is executed in the GPU with multiple threads.

Please note that CUDA and OpenCL programming may have different terminology for some cases. However, main idea is almost the same and content in this report will be explained using OpenCL terminology to prevent seperate complex descriptions for each framework. Important thing is to get the main feeling of how they work. Basic structure of this logic can be seen in Fig. 7.1 [1]

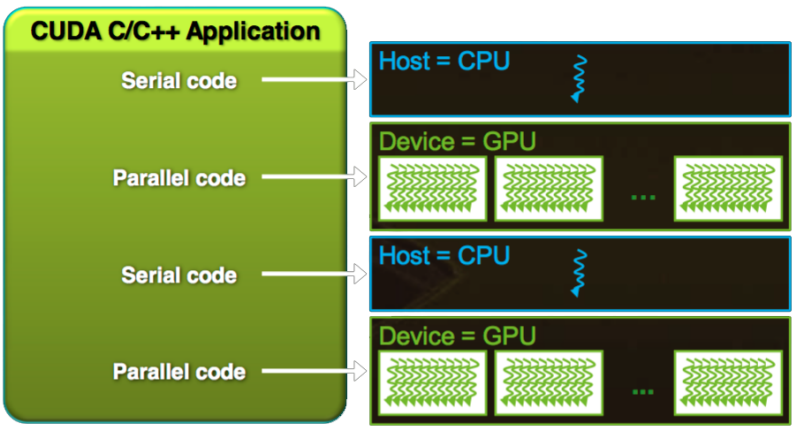



**Fig. 7.1.:** Basic Structure of a CUDA Application

[1] Taken from http://www.nvidia.com/content/gtc-2010/pdfs/2131_gtc2010.pdf

### Host Code and Kernel Code

*Host code* is the code executed in the host device which is the CPU in our case. *Kernel code* is executed in the device which is our GPU and it is executed multiple times in different GPU threads. Basic algorithm starts with initializing a device (e.g GPU). After that process, memory should be allocated in the device and the data will be transferred from host to device. After the calculations are done, the data will be transferred back to host and GPU memory will be deallocated.

OpenCV library provides modules for using OpenCL and CUDA without explicitly writing kernels. In the project, these modules were used.

## 7.3 OpenCV GPU Modules

Previously proposed needle tip localization and tracking implementations work in real time. However, OpenCV's OpenCL and CUDA modules are experimented in order to observe the effects of parallelisation in our algorithms. Considering the pipeline of our tracking algorithms, the main structure of the algorithm is highly sequential. Most of the instructions awaits response from previous calls so it is very challenging to improve performance using parallelization. On the other hand, tracking algorithms include some matrix multiplication and filtering operations that can be run in GPU.

### OpenCV and CUDA

OpenCV has a module called **GPU** for utilizing CUDA in OpenCV implementations. It includes implementations for utility functions, low-level primitives and high-level algorithms.[2]

To use OpenCV with CUDA support, OpenCV is configured with cmake using

```
WITH_CUDA=ON
```

After successful compilation, parallelizable operations in needle tip tracking algorithm is modified using GPU module.

Definition of data variables that will run in GPU:

```
cv::gpu::GpuMat jacobian, normalizedJacobianU, normalizedJacobianV;
```

[2] http://docs.opencv.org/2.4/modules/gpu/doc/introduction.html

At the start of algorithm, matrices that will be used are transferred to GPU's memory. An example is as following:

```
cv::Mat normalizedJacobianU;
cv::Mat normalizedJacobianV;
cv::subtract(jacobianVectorU, cv::mean(jacobianVectorU),
    normalizedJacobianU);
cv::subtract(jacobianVectorV, cv::mean(jacobianVectorV),
    normalizedJacobianV);

//The point where data is transferred to GPU memory
this->normalizedJacobianU.upload(normalizedJacobianU);
```

As an example, calculations of derivatives in regular code and CUDA compatible code are given.

Regular code:

```
aDerivative[0] = (float)
    cv::sum(templateDiff.mul(normalizedJacobianU))[0];
aDerivative[1] = (float)
    cv::sum(normalizedJacobianV.mul(templateDiff))[0];
bDerivative[0] = (float) (1.0/b *
    cv::sum(normalizedJacobianU.mul(warpedDiff))[0]);
bDerivative[1] = (float) (1.0/b *
    cv::sum(normalizedJacobianV.mul(warpedDiff))[0]);
```

CUDA code:

```
cv::gpu::GpuMat a0;
 cv::gpu::multiply(templateDiff,normalizedJacobianU,a0);
 aDerivative[0] = (float) cv::gpu::sum(a0)[0];

 cv::gpu::GpuMat a1;
 cv::gpu::multiply(normalizedJacobianV,templateDiff,a1);
 aDerivative[1] = (float) cv::gpu::sum(a1)[0];

 cv::gpu::GpuMat b0;
 cv::gpu::multiply(normalizedJacobianU,warpedDiff,b0);
 bDerivative[0] = (float) (1.0/b * cv::gpu::sum(b0)[0]);

 cv::gpu::GpuMat b1;
 cv::gpu::multiply(normalizedJacobianV,warpedDiff,b1);

 bDerivative[1] = (float) (1.0/b * cv::gpu::sum(b1)[0]);
 cv::gpu::multiply(templateDiff,normalizedJacobianU,a0);
```

In the final step, data are transferred back to the host.

```
//Transferring back
this->normalizedJacobianU.download(normalizedJacobianU);
```

#### OpenCV OpenCL Module

OpenCV also includes an OCL module which includes functions that accelerates OpenCV functions for compatible OpenCL CPUs and GPUs.

Firstly, an environment variable for *OPENCV_OPENCL_DEVICE* is added to the system to select the GPU to use. It can be assigned as following:

```
:GPU:1
```

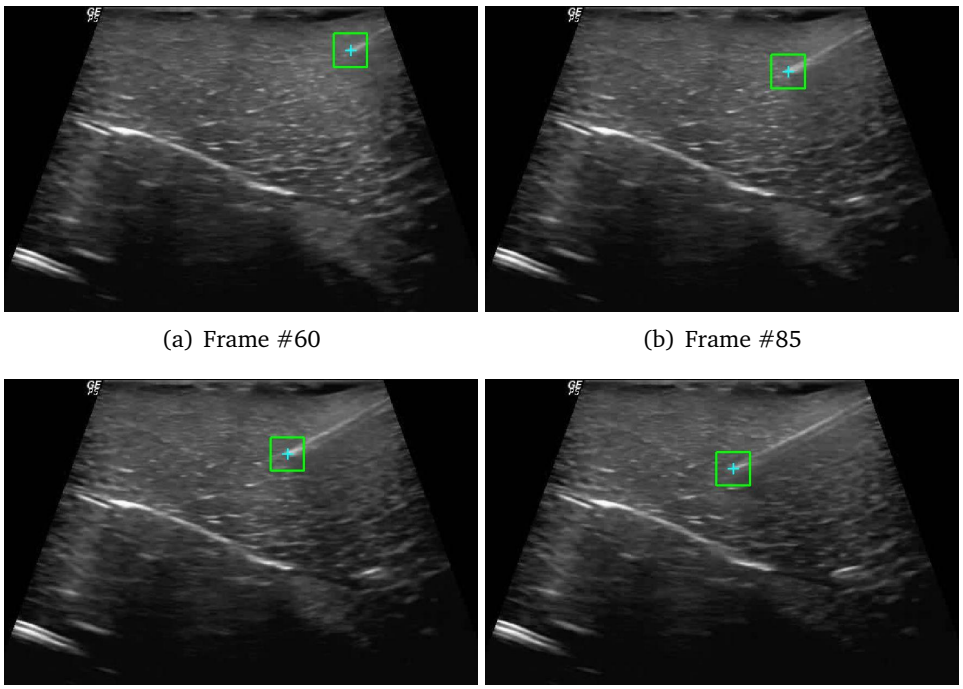
After the initial setup, data transfers and operations are similar to CUDA module. Changing *cv::Gpu::** with *cv::ocl:** allows us to use OCL module.

## 7.4 Results and Discussion

In the experiments, algorithms are tested using Intel HD 5000 and NVIDIA 610M GPUs. Processing times were higher ($32 \pm 10$ ms) than the previously implemented algorithms using only GPU. There might be multiple reasons for this situation.

Firstly, in our algorithms matrix sizes are very small. Maximum sized matrix is a image which has a size of 640*480. Considering this, GPU is not providing a huge speedup during the calculations such as multiplication or filtering. On the other hand, there are data transfer overheads while transferring data from host to GPU or vice versa. Furthermore, as previously mentioned, our algorithms consists lots of sequential parts which requires us to transfer data back and forward again and again. Also, GPUs have lower clock speed than CPUs and for small and less parallelizable operations, this fact negatively impacts our code's performance. As a future plan, all algorithm can be renewed to prevent this overheads and become more parallel.

# 8 Experiments and Results



(a) Frame #60 (b) Frame #85

(c) Frame #100 (d) Frame #120

**Fig. 8.1.:** Ex vivo experiment. Needle is being inserted into chicken breast and its tip is visually tracked in 2D lateral US Images.

## 8.1 Experiments

In this section of the study, execution time of the proposed visual tracking of biopsy needles in 2D US images were measured and its accuracy was evaluated using an optical tracking system. Also, accuracy of 3D needle shape estimation from 2D transverse US images was evaluated using computer tomography (CT). In the following parts, experimental setup is described and then experimental results are given.

### 8.1.1 Experimental Setup

In the experiments, a GE LOGIQ P5 2D US machine and a GE 11L linear 2D US probe were used to acquire images. Size of the acquired images was 640×480 pixels and the images were captured from the US machine using EURESYS PICOLO HD 3G

frame grabber. Experiments were carried on gelatin and agar mixture, agar-based and gelatin-based phantoms, ethanol and distilled water mixture, and distilled water. Phantoms were prepared using the recipes given in [4]. Also, lamb meat and chicken breast were used as ex vivo phantom and they were embedded into gelatin to fixate them. In the needle insertion experiments, 22G×15 cm biopsy needles (WIPAK MEDICAL) and 12G×26 cm stainless steel shafts were used.

The US probe was attached to a 6-DOF industrial robot arm (KUKA KR 6 R9000 sixx), and the arm was used to move the US probe in 3D space to track the actual position of the needle tip in the 2D US image plane. A 5-DOF needle insertion robot was used to insert the biopsy needles. Motors of the needle insertion robot were driven using ESCON 50/5 servo controllers. Encoder data is obtained from Quanser Q8 data acquisition board.

In order to track the motion of all the system components in 3D space, OptiTrack optical motion capture system was used. 11.4 mm diameter optical markers were placed on the US Probe and the needle insertion robot as shown in Fig. B.2. The experimental setup is shown in Fig. 8.2. Mechanical design of the robotic system can be seen in Appendix B.

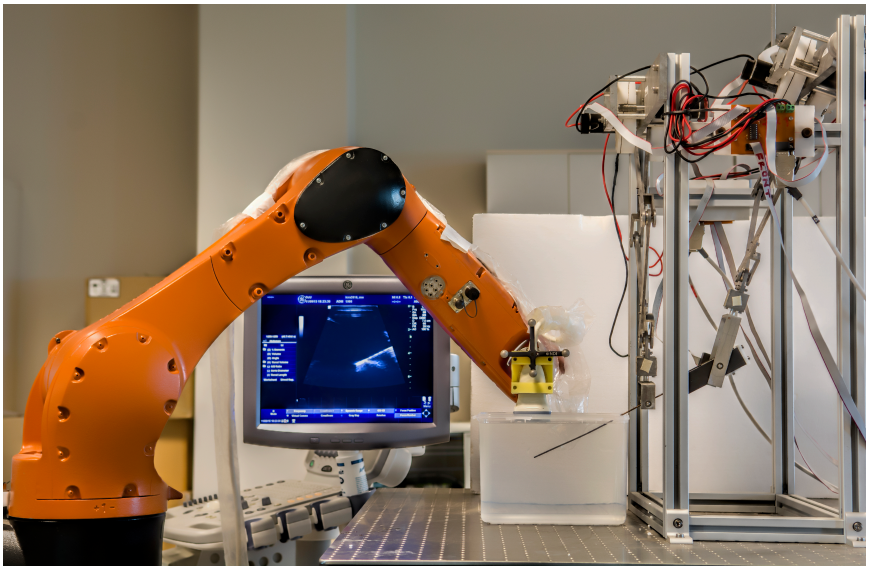


**Fig. 8.2.:** Experimental Setup: Optical markers were attached to needle insertion mechanism and US probe for initial positioning. 2D US probe was mounted onto a robotic arm for visual servoing and the needle was inserting into meat phantom.

### 8.1.2 Experimental Results

#### Needle Tip Tracking Accuracy

In order to evaluate the accuracy of the proposed visual tracking of biopsy needles, optical tracking system was used. Firstly, 2D US was calibrated in 3D space. Using the calibration, needle tip coordinates in 2D US image plane were calculated. Hence, the needle tip coordinates were obtained using an external sensor and these coordinates were accepted as actual needle tip points.

Accuracy measurement of the proposed method was conducted using robotic system. Before needle insertion started, needle and 2D US probe were aligned using the robotic arm. Then, the needle is inserted using needle insertion robot. The velocity of the needle was set to 10 mm/s in the needle insertions. During the needle insertions, the needle tip coordinates were calculated using optical tracking system and visual tracking simultaneously. In the needle insertion experiments, 9614 frames were acquired in total. In order to find the error of the proposed visual tracking of needle tip, the Euclidean distance between coordinates obtained using optical tracking system and visual tracking ($\mathbf{E}$) were calculated for each frame. Also, absolute errors in $x-$ ($\mathbf{E}_x$) and $y-$ ($\mathbf{E}_y$) axes were calculated. The error of visual tracking using different type of similarity measures were tabulated in Table 8.1. As seen from Table 8.1, NCC has the lowest tracking error and $2^{nd}$ order Gauss-Newton optimization reduced the tracking error over $35\%$.

**Tab. 8.1.:** Accuracy of visual tracking of biopsy needles

| **Method** | **Accuracy [mm]** | | |
|---|---|---|---|
| | $\mathbf{E}_x$ | $\mathbf{E}_y$ | $\mathbf{E}$ |
| $1^{st}$ order SSD | 0.54±0.03 | 0.39±0.02 | 0.63±0.07 |
| $2^{nd}$ order SSD | 0.36±0.08 | 0.20±0.08 | 0.42±0.07 |
| NCC | 0.15±0.03 | 0.10±0.02 | 0.22±0.02 |

#### Execution Time

The algorithm was implemented in both C++ using the OpenCV library and MATLAB and ran on 64-bit Windows 7 workstation, which has an Intel Xeon E5-2620 CPU running at 2 GHz and 32 GB RAM. The size of template images was 61×61 during the visual tracking. Threshold value for breaking the iteration loop ($\epsilon$) and the maximum iteration number were set to 0.01 and 50, respectively. The execution

times of visual tracking using $1^{st}$ and $2^{nd}$ order optimization of SSD, SCV, and NCC are tabulated in Table 8.2.

**Tab. 8.2.:** Execution time for the methods used

| **Method** | **Execution Time [ms]** | |
|---|---|---|
| | **C++** | **MATLAB** |
| $1^{st}$ order SSD | $29.74 \pm 10.43$ | $180 \pm 130$ |
| $2^{nd}$ order SSD | $21.86 \pm 6.44$ | $100 \pm 40$ |
| NCC | $27.76 \pm 10.59$ | $660 \pm 170$ |

As seen from Table 8.2, the processing time of visual tracking using SSD and SCV was reduced over $40\%$ using $2^{nd}$ order Gaussian-Newton optimization method. Also, processing time of SSD and SCV based visual tracking method was reduced five folds and processing time of NCC based visual tracking method was reduced twenty folds in C++ implementation.

# 9 Conclusion and Future Work

This study presents a package of algorithms which offers needle tip localization and visual tracking applications along with a standalone application. At the same time, examinations of algorithms for GPU acceleration are conducted. Package contributes to the literature with two different parts. First part includes the algorithmic background for needle tip localization, visual tracking of the needle tip, curve needle detection and 3D needle shape estimation. Second part consists of a software package which combines all algorithms in one easy-to-use software bundle. Accuracies and processing times of algorithms are evaluated. Processing times are calculated using MATLAB, C++ and GPU implementations. Results have shown that all algorithms are running in real-time when they are tested in the robotic experimental setup. On the other hand, accuracies are calculated for shape estimation and tracking. For the shape estimation, results are compared with external CT scan information. This evaluations supports the validity of algorithms.

Whole robotic system can be examined for economic, health, manufacturability and security aspects. First of all, system is not expensive to build and software bundle is created using all open source technologies. This makes it easier to develop and distribute. Since it is a medical robotics project, health is the main concern and having a robust system is very important. Without that, operations can result in catastrophic consequences. Algorithms are developed especially for detecting errors such as target loss. These perspective created a secure system. Manufacturability is related to the economic impact of the system and main purpose is to create a system that is easy to produce. Both hardware and software are compatible with that purpose.

## 9.1 Future Work

The algorithm part of tracking and localization is complete. However, needle curve detection algorithm is prone to wrong detections if it is the beginning of insertion. Another task to do is to create a automatic parameter detection mechanism for target loss and ROI size parameters of tracking. Finally, to increase the parallel localization of multiple images, GPUs can be utilized better. However, all of the algorithm should be refined to be able to run OpenCV operations in GPU device with custom kernels and this is a challenging task.

# Appendix

A

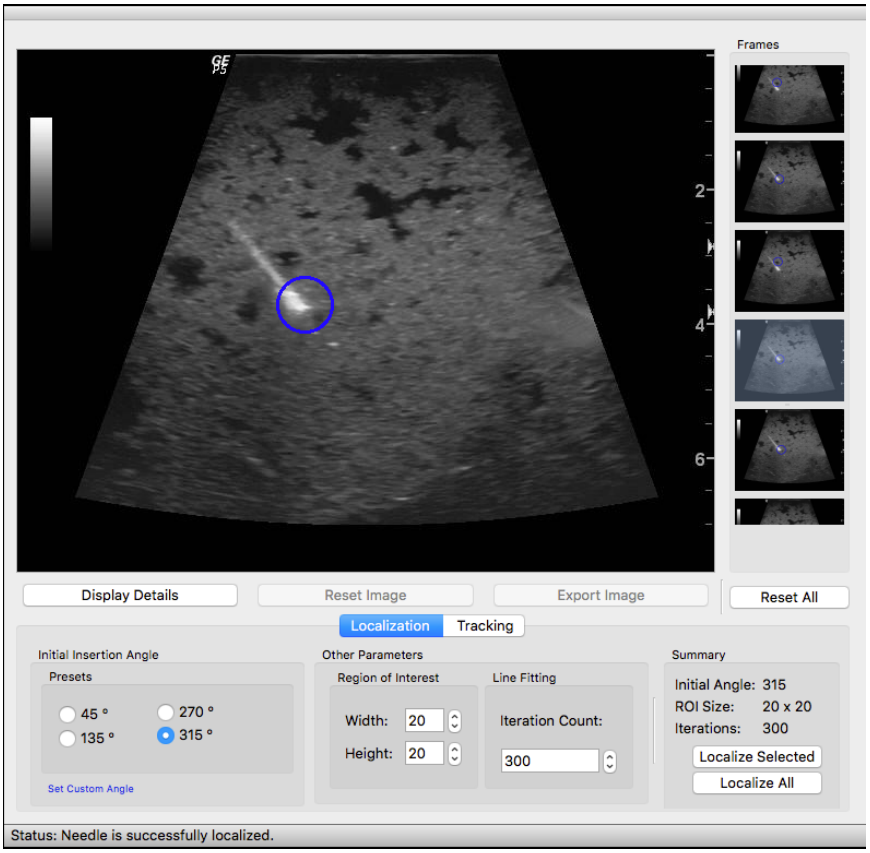



**Fig. A.1.:** Example Localized Image

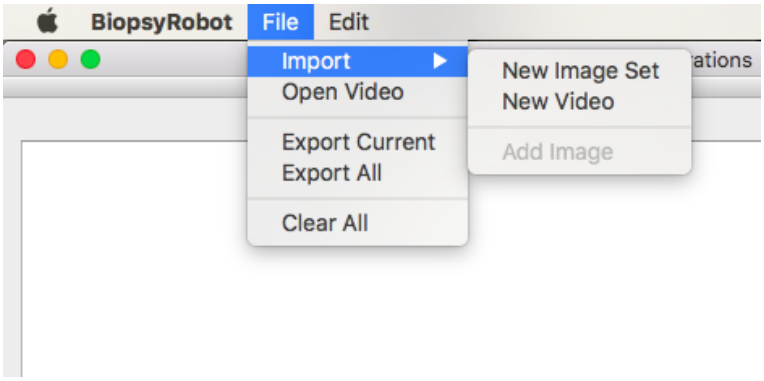



**Fig. A.2.:** Image / Video Import Menu

## A.1 Qt Components

In the project, different components are combined to get experience. Some of the components are explained below.

### QGraphicsView

This component is used for displaying the image in the main screen. A *Scene* is created and added to the view. When a change in the image occurs, scene is updated.

### QListWidget

This component is used for displaying thumbnails. Thumbnails are added as QIcon of ListWidget's items.

### QTabWidget

QTabWidget is used to create the bottom panel which includes the localization and tracking parameters.

### QCustomPlot

For the real time error plot, an external file is used. QCustomPlot is a Qt extension that allows to create cool graphs that can be changed in runtime. [1]

[1] QCustomPlot Webpage: http://www.qcustomplot.com

# Appendix

B

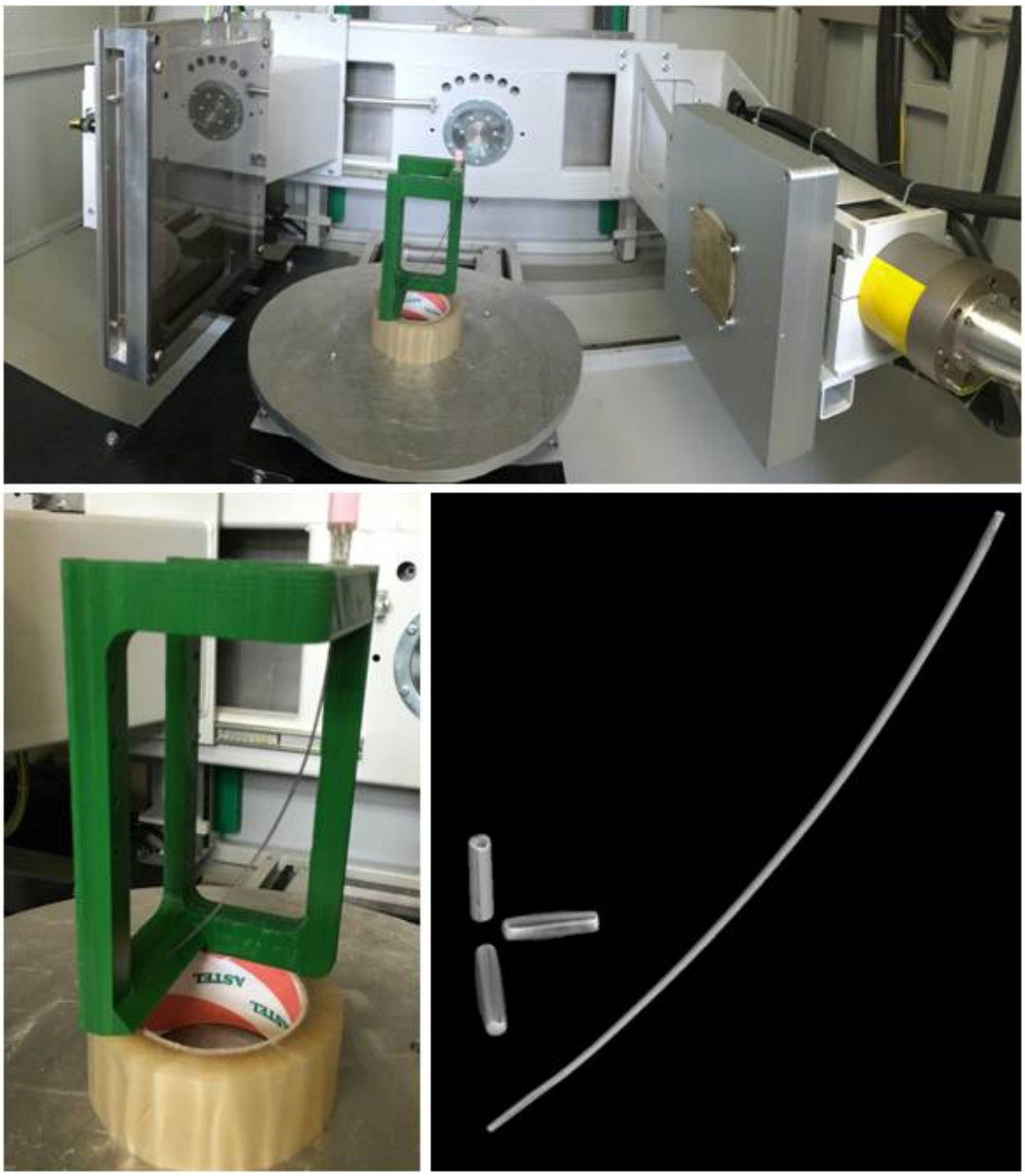


**Fig. B.1.:** Setup of the CT Machine. Green mold is fabricated using a 3D Printer and it allows us to keep the shape of the curved needle steady.

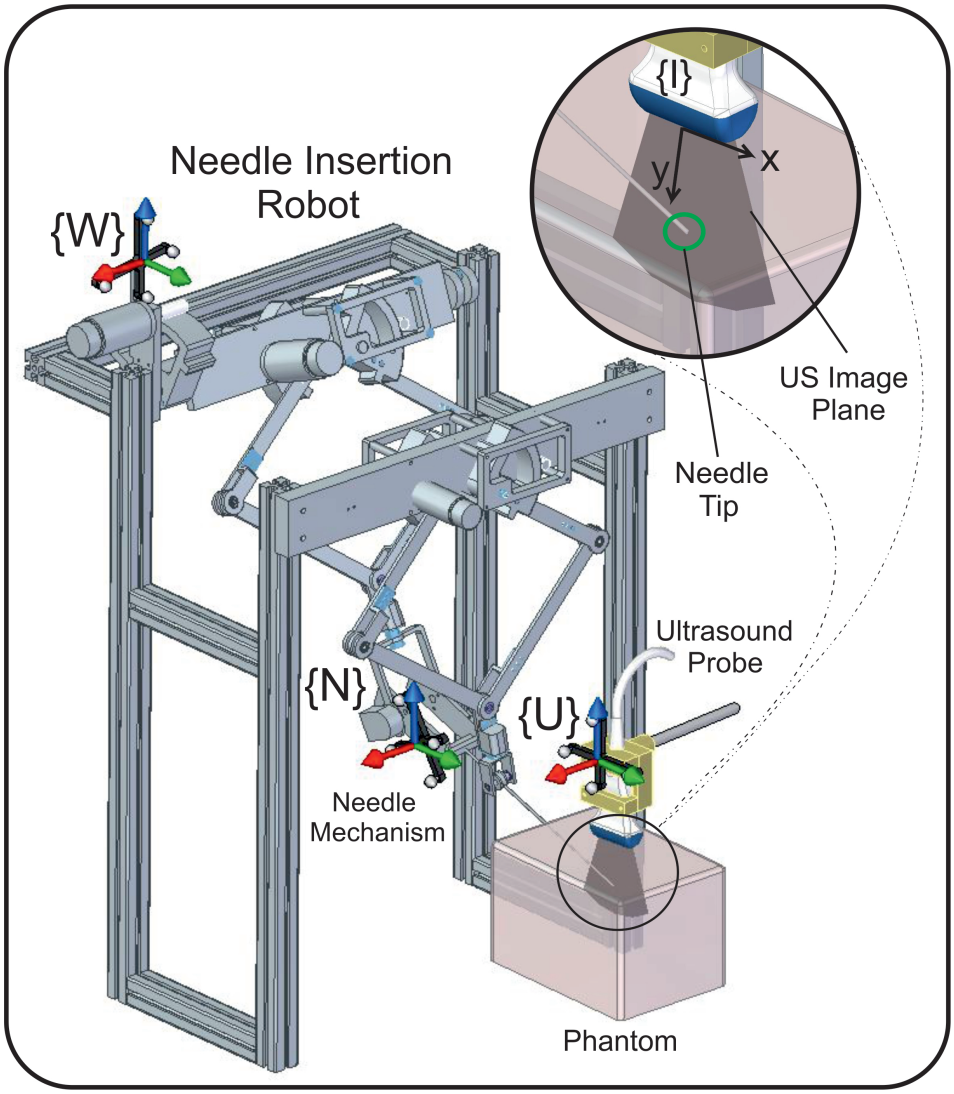



**Fig. B.2.:** 2D US Guided Biopsy Robot Design: Biopsy needle is inserted using needle insertion robot and needle tip is kept in the center of US image plane by servoing the 2D US probe.

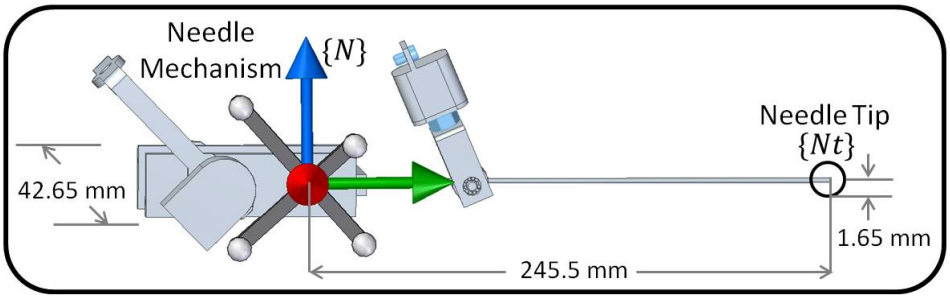



**Fig. B.3.:** Needle mechanism design with optical markers.

# Bibliography


[1] Simon Baker and Iain Matthews. „Lucas-kanade 20 years on: A unifying framework“. In: *International journal of computer vision* (2004) (cit. on p. 5).

[2] Gregory D Hager and Peter N Belhumeur. „Efficient region tracking with parametric models of geometry and illumination“. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (1998) (cit. on p. 6).

[3] M. Kaya, E. Senel, A. Ahmad, O. Orhan, and O. Bebek. „Real-time Needle Tip Localization in 2D Ultrasound Images for Robotic Biopsies“. In: *IEEE International Conference on Advanced Robotics (ICAR)*. 2015 (cit. on p. 2).

[4] Mert Kaya and Ozkan Bebek. „Gabor Filter Based Localization of Needles in Ultrasound Guided Robotic Interventions“. In: *IEEE International Conference on Imaging Systems and Techniques (IST)*. Oct. 2014 (cit. on pp. 2, 38).

[5] Mert Kaya and Ozkan Bebek. „Needle Localization Using Gabor Filtering in 2D Ultrasound Images“. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. May 2014 (cit. on p. 2).

[6] Iain Matthews, Takahiro Ishikawa, and Simon Baker. „The template update problem“. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* (2004) (cit. on p. 12).

[7] Rogeria Richa, Raphael Sznizman, and Gregory Hager. *Robust Similarity Measures for Gradient-Based Direct Visual Tracking*. Tech. rep. 2012 (cit. on p. 5).

[8] Rogério Richa, Raphael Sznitman, Russell Taylor, and Gregory Hager. „Visual tracking using the sum of conditional variance“. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference On*. 2011 (cit. on p. 6).

[9] G.J. Vrooijink, M. Abayazid, and S. Misra. „Real-time three-dimensional flexible needle tracking using two-dimensional ultrasound“. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2. May 2013, pp. 1680–1685 (cit. on p. 17).

[10] Michael Waine, Carlos Rossa, Ronald Sloboda, Nawaid Usmani, and Mahdi Tavakoli. „3D shape visualization of curved needles in tissue from 2D ultrasound images using RANSAC“. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 4723–4728 (cit. on p. 17).

# List of Figures

# List of Tables

# Acknowledgement

Besides my advisors Ozkan Bebek and Erhan Oztop, I also want to thank Mert Kaya for the ultimate guidance through out the project. I want to thank Awais Ahmad for the cool robot diagrams.

This work was supported by the Scientific and Technical Research Council of Turkey (TUBITAK) under Grant No. 112E312.