Vision based tracking and 3D reconstruction of concentric tube robot

Viktor Vörös

Internship at King’s College London

02/09/2019 - 13/12/2019

<https://github.com/viktorvoros/CTR3dReconstruction>

Table of content

[1. Introduction 5](#_Toc26802011)

[2. Literature review 5](#_Toc26802012)

[2.1 Concentric tube robots 5](#_Toc26802013)

[2.2 Vision 6](#_Toc26802014)

[2.3 Stereovision 8](#_Toc26802015)

[2.3.1 Epipolar geometry 8](#_Toc26802016)

[2.3.2 Rectification 9](#_Toc26802017)

[2.3.3 Disparity 9](#_Toc26802018)

[3. Developed algorithm 12](#_Toc26802019)

[3.1 Pre-processing 13](#_Toc26802020)

[3.2 Image processing 13](#_Toc26802021)

[3.2.1 Find origin 14](#_Toc26802022)

[3.2.2 Find centerline points 14](#_Toc26802023)

[3.2.3 Find width of tube along centerline 17](#_Toc26802024)

[3.2.4 Find the threshold values for joint and tip allocation 18](#_Toc26802025)

[3.2.5 Find mid points 20](#_Toc26802026)

[3.3 3D reconstruction 21](#_Toc26802027)

[3.4 Graphical user interface 22](#_Toc26802028)

[3.5 Camera holder design 23](#_Toc26802029)

[4. Validation 24](#_Toc26802030)

[5. Discussion 26](#_Toc26802031)

[6. Future work 26](#_Toc26802032)

[7. Scripts and files 27](#_Toc26802033)

[8. References 29](#_Toc26802034)

**List of Figures**

[Figure 2.1 Kinematic parameters of a 6 DoF concentric tube robot, composed of 3 curved tubes [1]. 5](#_Toc26802035)

[Figure 2.2 Pinhole camera model [5]. 6](#_Toc26802036)

[Figure 2.3 Radial distortion: Undistorted (a), positive (b) and negative radial distortion (c) [5]. 7](#_Toc26802037)

[Figure 2.4 General layout of stereovision [7]. 8](#_Toc26802038)

[Figure 2.5 Epipolar geometry. The epipolar plane OO’P. The images p and p’ of the point P, epipolar lines l and l’ and the epipoles e and e’. The center of the cameras are O and O’ [6]. 8](#_Toc26802039)

[Figure 2.6 Stereo triangulation for rectified images [8]. 9](#_Toc26802040)

[Figure 2.7 The inverse proportional relation between disparity and distance (z). The closer an object, the bigger the disparity. 10](#_Toc26802041)

[Figure 3.1 The block diagram of the algorithm for CTR 3D reconstruction. 12](#_Toc26802042)

[Figure 3.2 Result of pre-processing: The captured left (a) and right (b) image and the resulted rectified, cropped, threshold left (c) and right (d) images. 13](#_Toc26802043)

[Figure 3.3 The origin of the robot marked by red circle on both the left and right image. 14](#_Toc26802044)

[Figure 3.4 Transforming Cartesian coordinates to polar coordinates. 15](#_Toc26802045)

[Figure 3.5 The identified centerline points of the CTR on both images marked by green circles. 16](#_Toc26802046)

[Figure 3.6 Finding centerline points. Original coordinate system with origin O (a); move origin to the robot origin O’ (b); transform Cartesian to polar coordinate system and locate centerline points (r,th) marked by green (c); reverse transform to Cartesian coordinate system with origin O and centerline points with (x,y) coordinates (d). 16](#_Toc26802047)

[Figure 3.7 Applying search range on the centerline of the robot: Determining vector pointing from previous to next point (a) and rotating the vector by 90° going through point with length l (b). The drawn search lines at each centerline point on the left (c) and right (d) image. 17](#_Toc26802048)

[Figure 3.8 The change of width along the robot (a) and the moving average (b) that is used for adaptive thresholding marked by the red lines. 18](#_Toc26802049)

[Figure 3.9 The identified joint and tip points on the left (a) and right (b) images. The first joint is colored with blue, the second with green and the tip is marked by light blue color. 19](#_Toc26802050)

[Figure 3.10 The determined mid points of each tube segment on the left (a) and right (b) view and the additional mid points resulting in 13 points per image on (c-d). 20](#_Toc26802051)

[Figure 3.11 Triangulation to calculate z coordinate 21](#_Toc26802052)

[Figure 3.12 The reconstructed 3D points of the CTR joints 22](#_Toc26802053)

[Figure 3.13 The graphical user interface. 22](#_Toc26802054)

[Figure 3.14 Stereo camera holder. 23](#_Toc26802055)

[Figure 4.1 The experiment layout. 24](#_Toc26802056)

# Introduction

I spent 3,5 months of internship at the School of Biomedical Engineering and Imaging Sciences after my master studies with the aim of getting familiar with computer vision (CV) and its challenges. This project gave me the opportunity to learn the basics of computer vision and the use of openCV through creating an algorithm for tracking and making a 3D reconstruction of a concentric tube robot (CTR) by using 2 web cameras. Since the robot does not have many features, as it consists of 3 long and thin tubes, finding the joint points, where the tubes go into each other was very challenging.

This document is a summary of the knowledge, experience I gained and the methods I tried out during this project, starting with an overview of CTRs and basic principles of stereovision, following with the introduction of the developed algorithm from pre-processing through image processing to 3D reconstruction and the validation of the proposed method.

The developed algorithm and corresponding scripts are explained as well as a small user manual can be found at the end of the document in Chapter 7.

Everything can be found on github: <https://github.com/viktorvoros/CTR3dReconstruction>

# Literature review

The required background is summarized in this chapter about concentric tube robots and stereovision.

## Concentric tube robots

Continuum robots are emerging in surgery, as they can be navigated through the human anatomy to reach hardly accessible sites deep inside the human body.

Representatives of these types of robots are concentric tube robots (CTR) that have enhanced dexterity and flexibility to take on complex 3D shapes, which enables them to be used in sensitive environments. CTRs are composed of multiple concentric, pre-curved, elastic tubes. The steering in 3D space is done by the manipulation of the 2 degrees of freedom (DoF) of each tube, relative translation () and rotation () w.r.t each other [1, 2, 3]. Figure 2.1 represents the kinematic parameters of a 6 DoF concentric tube robot.

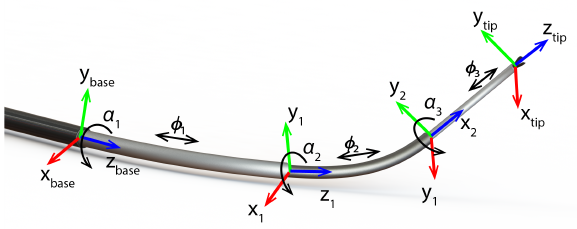


Figure 2.1 Kinematic parameters of a 6 DoF concentric tube robot, composed of 3 curved tubes [1].

Since the tubes have different stiffness, the preceding tube gives the initial shape of the examined tube. Therefore, when the tubes are moving w.r.t each other, the curvature and orientation of each tube changes, thus the tip can reach to each point within the workspace in multiple configuration. It is ideal for minimally invasive surgery (MIS), where careful navigation is required around fragile tissues.

Despite of the simple translation and rotation DoFs, due to the elastic interactions of the tubes and lack of rigid joints, the kinematic characterization of the robot results in a highly non-linear behavior [3, 4]. In addition, due to their flexibility, shape deformation caused by interaction forces when colliding with the anatomy inside the body makes the kinematic modelling even more challenging, as the sensing of these intraoperative forces is not always possible [1].

Vision-based shape sensing can lead to a safer intraoperative navigation and to more accurate estimation of the 3D alignment of the robot within the patient.

## Vision

Using a pinhole camera model, image acquisition is done by using perspective transformation to project 3D points on the image plane. On Figure 2.2, point P (x, y, z) is projected on the image plane as p (u, v), where the camera/world coordinate frame is Fc. The principal point of the image plane is located at (cx, cy), usually at the center of the image.

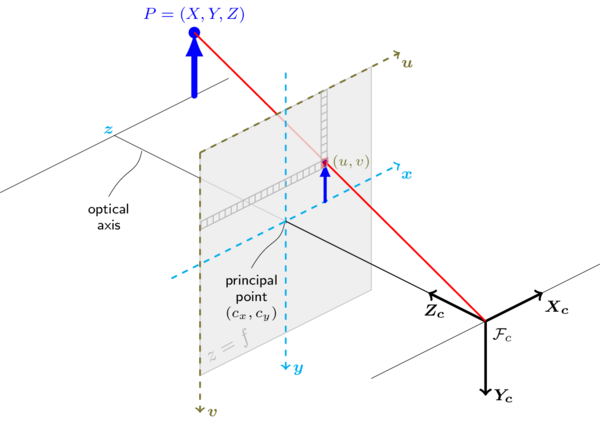


Figure 2.2 Pinhole camera model [5].

The projection of point P to the image plane:

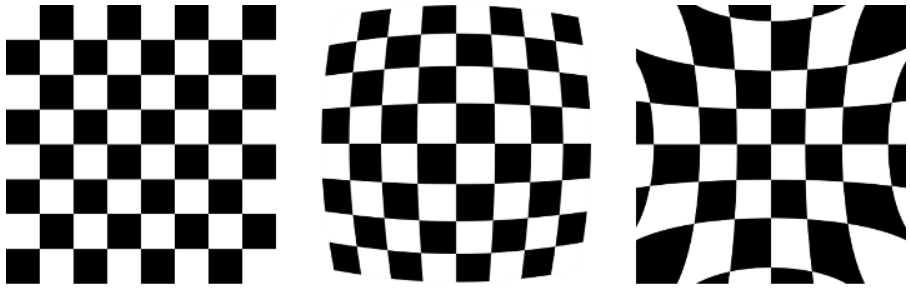
Where:

* C – camera matrix
* – Rotation-translation matrix

In matrix form:

The camera matric C contain the intrinsic parameters, focal lengths and image principal points. These parameters do not depend on the scene, they are properties of the camera, once estimated, can be re-used. The rotation translation matrix contains the extrinsic parameters, describes the relative static motion of the object viewed, they relate world points to image points.

In reality the camera model is not pinhole, as the lenses of the camera are not perfectly manufactured, therefore they have distortion. There are two types of distortion, tangential and radial [5, 6]. Radial distortion can be negative or positive as shown on Figure 2.3, and it makes straight lines curved. By finding the distortion coefficients of the camera, the images can be undistorted.



(a) (b) (c)

Figure 2.3 Radial distortion: Undistorted (a), positive (b) and negative radial distortion (c) [5].

The distortion coefficients:

Where:

* – radial,
* – tangential distortion coefficients

Both the camera matrix and distortion coefficients are obtained by camera calibration, and once found, they can be re-used, as these intrinsic parameters do not depend on the scene. Estimation of intrinsic and extrinsic parameters from different views of a calibration pattern with known dimension.

The next step is to estimate the relative position and orientation of the two cameras for stereo vision and to use epipolar geometry to know the 3D coordinates of an object.

## Stereovision

During this project, stereovision was used for the shape sensing of a CTR, therefore in this chapter an overview of stereovision will be conducted followed by a review of other approaches in the next chapter.

Stereovision is used to extract 3D information from image pairs that are obtained from the scene from 2 different angles, similarly to our eyes. With stereovision it is possible to generate depth maps and make 3D reconstruction of a scene that can be used for object tracking or navigation. A general layout is shown on Figure 2.4.

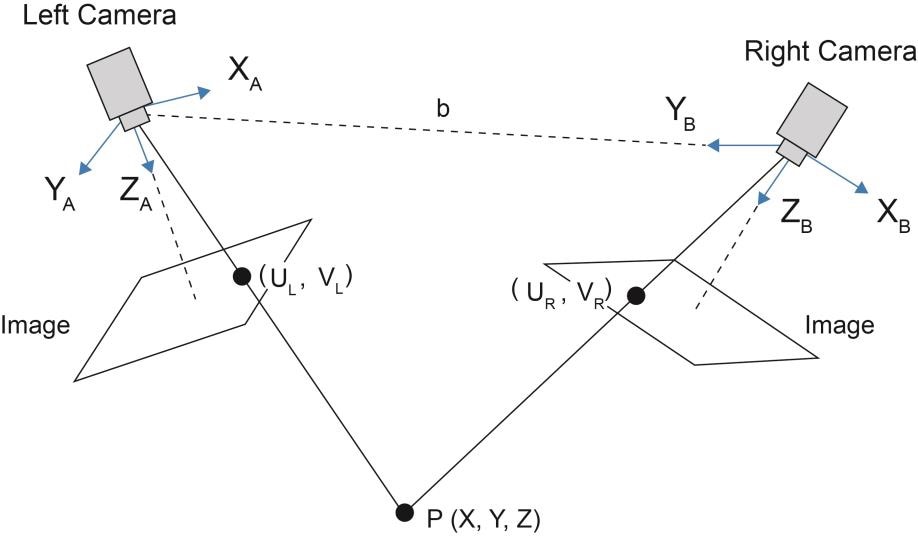


Figure 2.4 General layout of stereovision [7].

Point P is observed by the left and the right camera and projected to their image planes with coordinates , while b, the distance between the two cameras is called the baseline.

### Epipolar geometry

To be able to calculate the 3D coordinates of an object, or to generate a depth map epipolar geometry can be used, a geometric approach, also known as triangulation.

The optical centers of the two cameras are O and O’, the images of the point P are p and p’. The epipolar plane is defined by the rays of OP and O’P, where p lies on l and p’ on l’ on the two image planes. The two lines, l and l’ are called epipolar lines associated with point P. The epipoles of the cameras are e and e’, where the baseline intersects the image planes. These are the virtual image of the other camera (Figure 2.5) [6].

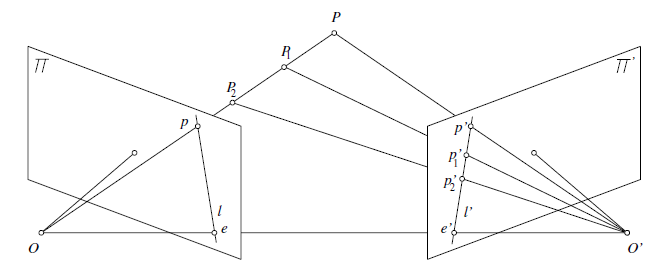


Figure 2.5 Epipolar geometry. The epipolar plane OO’P. The images p and p’ of the point P, epipolar lines l and l’ and the epipoles e and e’. The center of the cameras are O and O’ [6].

Different points on the OP line projects to different points on the right plane, however, the projections are all along the epipolar line l’. It means that from 2 views of the same scene, 3D coordinates can be calculated. With stereo calibration, the position of the two cameras w.r.t. each other can be determined.

Using an object with known dimension, such as a checkerboard we can acquire a set of image points with known global dimensions. By a given set of matched image points, one can calculate the essential matrix E that provides information about the translation and rotation transform of the right camera w.r.t. the left and the fundamental matrix F that carries additional information of camera intrinsic parameters. While E relates the two cameras in global coordinates, F relates the two cameras in pixel coordinates. These two matrixes are the result of stereo calibration.

### Rectification

To simplify the calculations and reduce it to a 2D case, a new, common image plane is created and the two image points are projected to this plane. Since it is parallel to the baseline as well as the epipolar lines that are now horizontal. The corresponding points lay on the same horizontal line, therefore v coordinates are the same.

From the essential matrix a rotation vector R can be obtained

### Disparity

Point P on the rectified image plane is represented by p (u,v) and p’ (u’,v). The closer an object is to the two cameras the bigger the difference between the u coordinate is.

Where d is the disparity.

From similar triangles, the distance of a point from the rectified image plane can be calculated by knowing b, the baseline and f, the focus of the camera as it is shown on Figure 2.6.

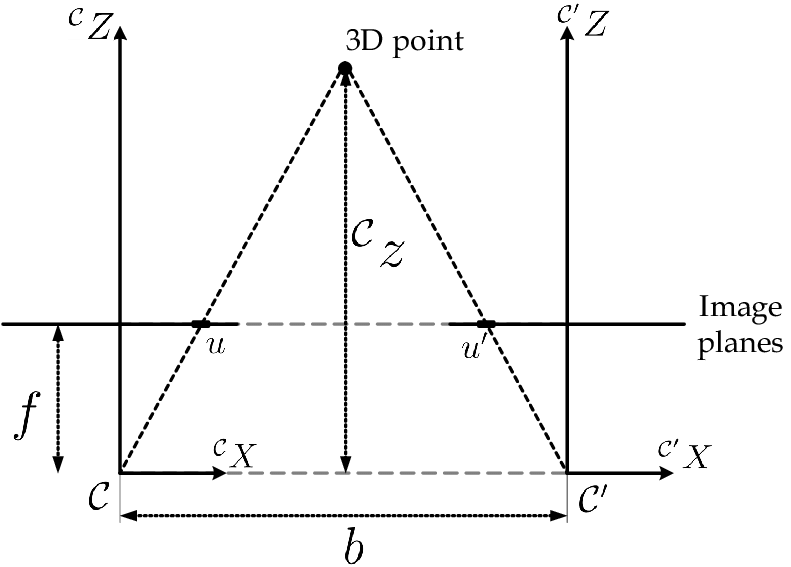


Figure 2.6 Stereo triangulation for rectified images [8].

Mathematically:

Where:

* u, u’ – points of object on rectified image plane
* z - object distance
* b – baseline
* f – focus of camera

Therefore:

And the z coordinate:

The x and y coordinates are similarly:

Z is inverse proportional with the disparity as it can also be seen on Figure 2.7, where z is called distance.

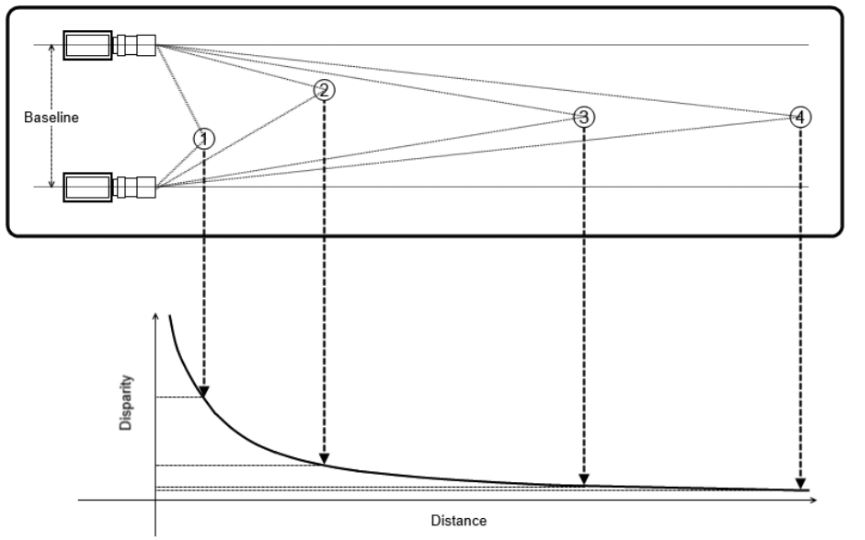


Figure 2.7 The inverse proportional relation between disparity and distance (z). The closer an object, the bigger the disparity.

Using stereovision, the 3D coordinates can be calculated, by calibrating the cameras to know their global positions and rectifying the images to a common image plane to be able to calculate the disparity. There is an inverse proportional relation to the z coordinate that can be calculated from triangulation.

First, I tried a general method to build a depth map of the CTR, by using the built-in openCV functions, such as semi global block matching (SGBM), however, the properties of the robot does not allow to use it. The robot is featureless, consisting of thin tubes with a small diameter difference that is difficult to find on the two images, thus the SGBM method failed, could not find the robot at all, even after applying a WLS filter on the images.

After analyzing similar approaches [1, 9, 10, 11, 12, 13], a new, mathematical method was developed for CTR tracking. Dalvand *et al.* [10] proposed a real-time catheter tracking method with stereovision that was very similar to this project’s use case.

# Developed algorithm

The algorithm was developed with the use of Python 3.7 and openCV 4.1.1, the cameras used were Logitech C930e (1080p, 20fps). The concentric tube robot used consists of three tubes. The challenge was that the robot does not have many features and significant diameter reduction at the joint points. The robot on the two images is captured with different number of pixels, therefore each point cannot be matched.

The goal was to find the centerline of the robot and to locate its joint points together with the tip position. After locating these main points, each tube is identified with 3 points – 2 joint and 1 centerline point in the middle. These 7 points on both images then matched and disparity is calculated to get the z coordinates. From these 7 points a 3D point cloud is developed. It is possible to identify more centerline points by locating the mid points between the neighboring points using the same function as before, therefore the accuracy of the shape sensing can be improved. The resulting method can be used for real-time tracking of the robot.

The algorithm is divided into 3 main parts:

* Pre-processing
* Image processing
* 3D reconstruction

These steps are conducted after camera and stereocalibration, since the camera matrix, the distortion coefficients and the fundamental matrix are needed.

The block diagram of the method is shown on Figure 3.1.

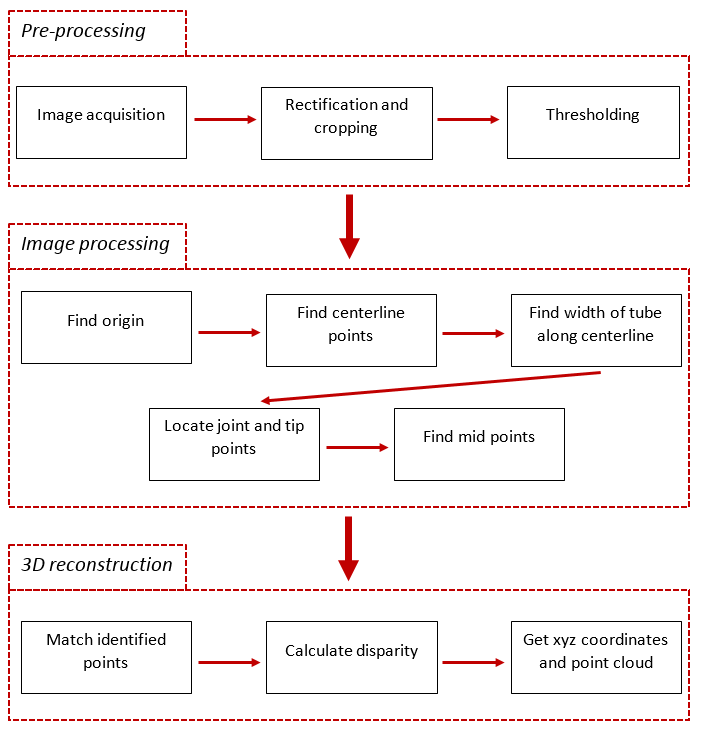


Figure 3.1 The block diagram of the algorithm for CTR 3D reconstruction.

## Pre-processing

During the pre-processing the captured images are undistorted and transformed to grayscale. The undistorted and rectified images are threshold to acquire binary images, with the robot points being represented with black pixels.

Cropping is needed to eliminate the non-investigated part of the robot, such as the frame and actuators, as these would give error in the thresholding. This step will be eliminated with the use of deep-learning to segment out the CTR from the image and make the algorithm more robust as a future step. This approach is also needed to improve the width calculation and to eliminate the effect of reflection on the surface of the tubes.

An example of the result of the pre-processing is shown on Figure 3.2.

(a) (b)

(c) (d)

Figure 3.2 Result of pre-processing: The captured left (a) and right (b) image and the resulted rectified, cropped, threshold left (c) and right (d) images.

These rectified and threshold images are used for further processing. The advantage of using threshold images is that any camera images can be used as an input for the algorithm.

## Image processing

The different steps in this section, shown on the block diagram, are built up in functions that will be discussed. Each subchapter contains the function with its arguments as well.

### Find origin

*findOrigin(imgThresh, imgColor)*

The origin is located at the center point of the robot where it enters the image, in this case it is the bottom row. The function uses the threshold image (imgThresh) and finds all the black pixels with their coordinates. Then the last black pixel is selected, which is in this case the most bottom right black pixel, together with the most left black pixel of the same row. The average of the coordinates then calculated to find the middle point, thus the origin coordinates. The output of the function are the p0(x,y) coordinates of the origin.

Where:

* P0 – origin (x,y) coordinates
* - first and last black pixel (x,y) coordinates on bottom row

The second argument, imgColor is the cropped, rectified colored image used to mark the found point with a red circle, as shown on Figure 3.3.

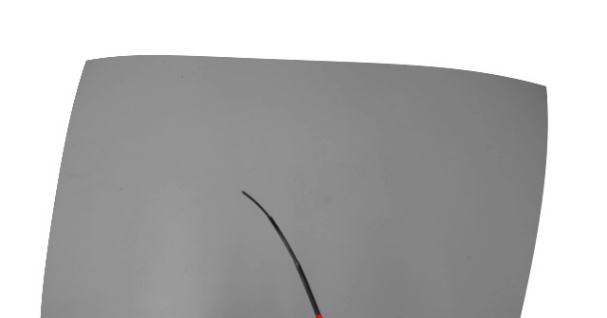
 

Figure 3.3 The origin of the robot marked by red circle on both the left and right image.

In order to make the algorithm more robust, the function first detects where the robot enters the image and finds the origin based on the orientation. It is done by checking on which edge black pixels can be found, and then finding the first and last black pixels on that row or column.

### Find centerline points

*findCenterPolar(origin, imgThresh, imgcolor)*

The next stage of image processing is identifying all the centerline points of the CTR on both images. The process of finding centerline points is described below and shown on Figure 3.6.

An initial step is finding all the black pixels with their coordinates on the threshold image (imgThresh), then these Cartesian coordinates are transformed to polar coordinates with the origin at the previously determined p0, origin of the robot.

Transforming the origin to p0:

Where:

* p0 – (x,y) coordinates of the robot origin
* p – (x,y) coordinates of all the robot pixels
* p’ – (x,y) coordinates of all the robot pixels with the new origin

Then all the new Cartesian coordinates are transformed to polar coordinate system as shown on Figure 3.4.

Where:

* - Cartesian (x,y) coordinates of the pixels
* - polar (r,) coordinates of the pixels

Polar transformation is required to find all robot pixels at each radius , thus all the points that belongs to the same cross section of the tubes. After finding the coordinates at , the mean of the pixels’ coordinate is calculated, which gives the coordinates of the centerline point of the cross section at .



Figure 3.4 Transforming Cartesian coordinates to polar coordinates.

Where:

* – coordinate of center point at
* – coordinates of each robot pixel at
* n – number of pixels at

After obtaining all the centerline coordinates, the polar coordinate system is transformed to Cartesian coordinate system, as well as the origin to the image origin.

Where:

* - x and y coordinates of the centerline points
* - x and y coordinate of the robot origin

The output of the function is an array containing all the (x,y) coordinates of each centerline point. The colored image is used to draw circles showing the identified centerline points showed on Figure 3.5.

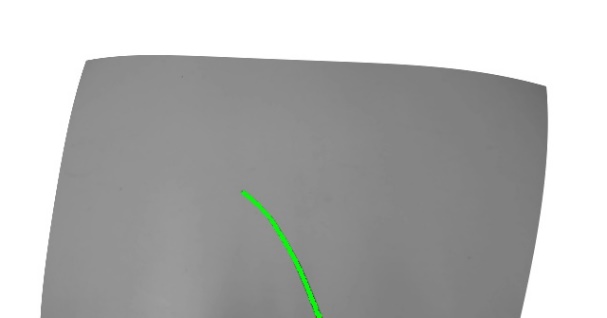
 

Figure 3.5 The identified centerline points of the CTR on both images marked by green circles.

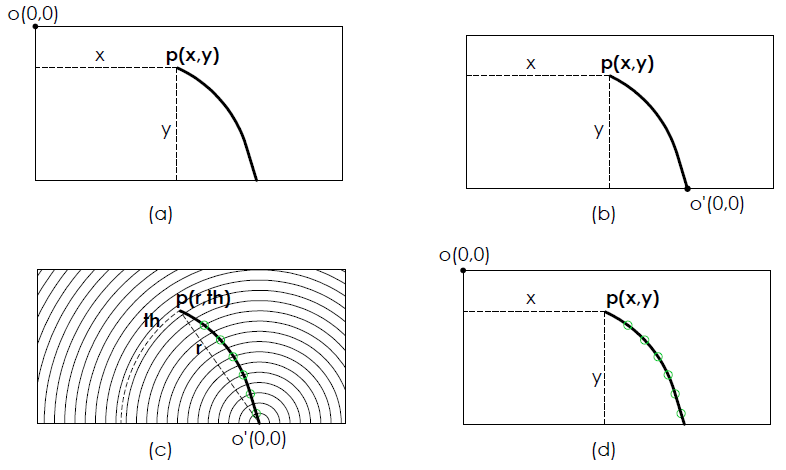


Figure 3.6 Finding centerline points. Original coordinate system with origin O (a); move origin to the robot origin O’ (b); transform Cartesian to polar coordinate system and locate centerline points (r,th) marked by green (c); reverse transform to Cartesian coordinate system with origin O and centerline points with (x,y) coordinates (d).

However all the centerline points are identified, which is ideal to build a dense 3D point cloud of the CTR, but the number of points on the images are not uniform, thus cannot be matched to calculate disparity. Therefore, further steps are required to locate specific points, like tip and joint points that can be matched and with which the different segments can be separated.

### Find width of tube along centerline

*findWidthVect(pC1, imgThresh, imgColor)*

The identification of all the centerline points results in an nx2 array containing each coordinate. This step calculates the width of the tube by creating lines perpendicular to the centerline at each point. It requires 3 points for each line: the centerline point being examined , the previous and next point.

First, a vector pointing from the previous to the next point is calculated and normalized Figure 3.7 (a).

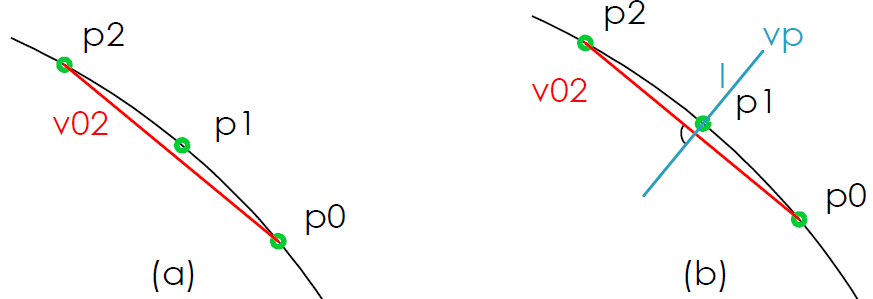
Where:

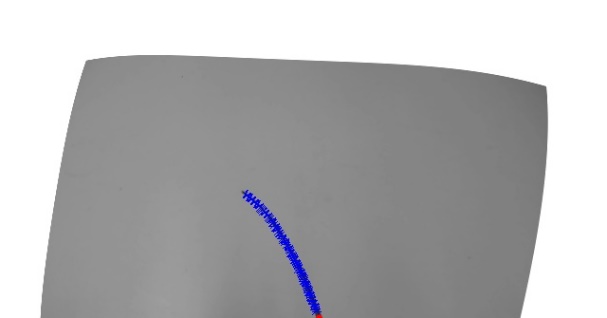
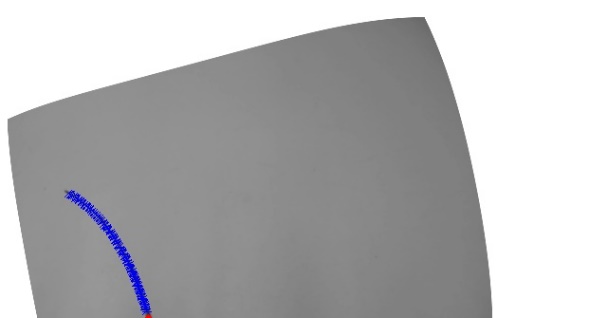
* – vector pointing from previous point to next point (nx2)
* – normalized vector (nx2)

Then the normalized vector is rotated by 90° by switching the columns and changing y coordinates to –y resulting in . The length of the line that goes through along is called the search range, since we are looking at the robot points along this line as shown on Figure 3.7.

Where:

* sr – search range vector (nx2xm)
* l – length of the line along (1x1xm)



(c) (d)

Figure 3.7 Applying search range on the centerline of the robot: Determining vector pointing from previous to next point (a) and rotating the vector by 90° going through point with length l (b). The drawn search lines at each centerline point on the left (c) and right (d) image.

The search range is then inserted on the examined center point and the robot pixels are found along the line.

Where:

* sp – search points being examined (nx2xm)

From sp, the number of robot points (non-zero elements) are determined. The width is then calculated by subtracting the number of robot points from the length of the search range.

Where:

* w – width of the tube at given point (nx1)
* - number of robot points along the search line l.

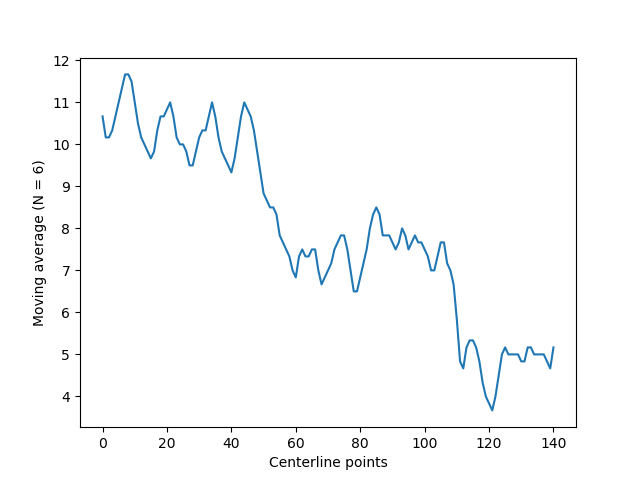
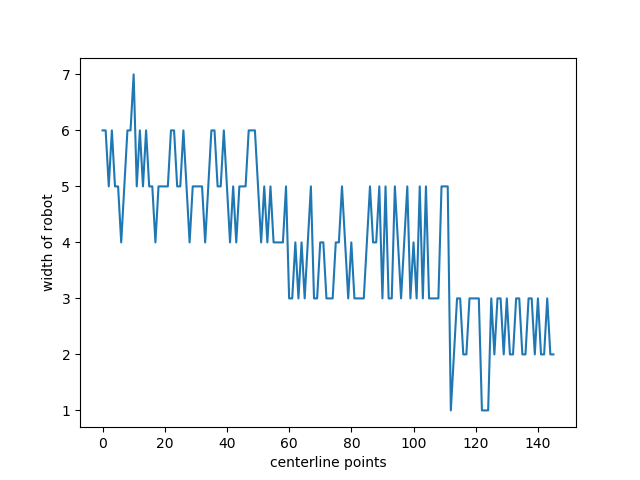
Since the width values calculated from the images are not constant at each tube, thus fluctuating by 1-2 values, the output of the function is an array containing the sum of the current and next width of the tube. It results in a more robust way to determine where a diameter reduction is, therefore a joint point.

Since we have all the points in one array, this can be done in one step, as it can be vectorized by rolling the array by 1 to get the previous point and by -1 to get the next point in the same row as our examined point. The same can be done with the width values.

### Find the threshold values for joint and tip allocation

*findThresholdValues(widthData, N); findJointPoints(movingAvg, centerPointX, centerPointY, th1, th2, imgColor)*

As the robot moves in 3D space the width values are slightly changing and can be different on the two images, so the threshold values that determine the diameter reduction based on the width values may vary. Therefore, adaptive thresholding is required, that calculates these values for each frame. An example of the change of the width along the robot is shown on Figure 3.8.



Thresh1

Thresh2

1. (b)

Figure 3.8 The change of width along the robot (a) and the moving average (b) that is used for adaptive thresholding marked by the red lines.

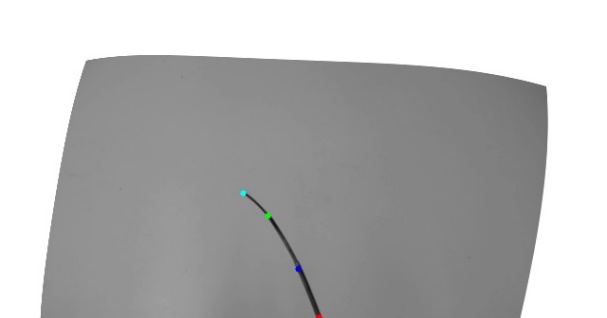
To find the width values where there is a step on the plot, first, moving average is calculated for each N number of points to filter out the signal for a better estimation. Then the mean value m of all the widths is calculated

The first threshold value that determines joint 1, where the middle tube enters the outer tube is calculated by the average of the max width and the mean.

While the second threshold value locating the second joint point, where the inner tube enters the middle tube is calculated by the average of the min width and the mean.

The -1 is added to the equation through practical experience, as the segmentation is not perfect, there can be significant jumps in the width values, where the diameter would shrink significantly, activating the thresholding, ending in an incorrect joint location. It shows that this function can be further improved, either by another approach or a better segmentation of the robot, where the width values are less fluctuating.

The first point where the width value matches with the threshold values locates the joint points. Tip position is simply located by the last centerline point along the robot. Each point are shown with different color on Figure 3.9.

(a) (b)

Figure 3.9 The identified joint and tip points on the left (a) and right (b) images. The first joint is colored with blue, the second with green and the tip is marked by light blue color.

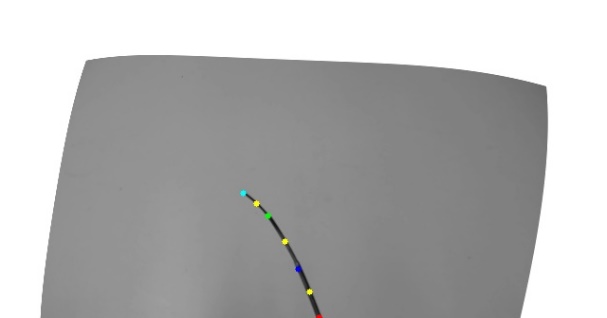
The output of *findThresholdValues* are the two threshold values and the moving average, while the output of *findJointPoints* are the two joint and the tip coordinates.

### Find mid points

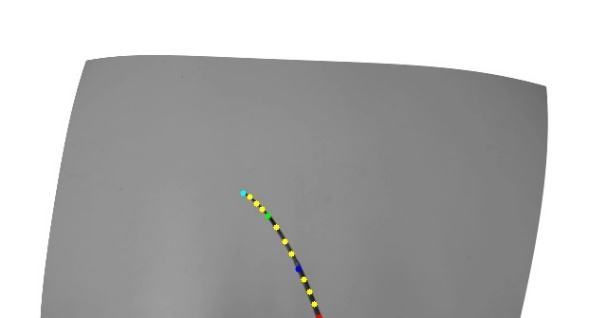
*findMidPoints(imgThresh, joint1, joint2, imgColor)*

The last step of the image processing is finding 3 additional points of the robot, so each tube is identified by 3 points. It results in altogether 7 points on each image: the origin, the 2 joint, the tip and 3 mid points. The number of points can be increased by finding further mid points between the already located robot points. Currently the algorithm can find 13 points.

The function takes 2 points as an input, e.g. the two joint points. Similarly to the width identification, a search line is generated in between the two points, perpendicular to the robot. Then the robot points are found along this search line and the mean coordinate values gives the middle point (x,y) coordinates. The working principle is the same as in subchapter 3.2.3. The found midpoints can be seen on Figure 3.10 marked by yellow points.

(a) (b)

(c) (d)

Figure 3.10 The determined mid points of each tube segment on the left (a) and right (b) view and the additional mid points resulting in 13 points per image on (c-d).

The output of the function is the midpoint coordinate. Since the function just requires 2 points as input, it can be used as many times as needed.

The image processing results in a number of specific robot points that can be matched on the two images and disparity can be calculated to get the z coordinate of each point. The reason why these points are matched and not all the centerline points is that the robot does not have the same number of pixels on both images, so matching all the points would cause error, as in this case the error can be reduced as specific points, like joint and tip are matched. These are enough to roughly estimate the orientation of the robot in 3D space.

## 3D reconstruction

*find3dCoords(xyL, xyR)*

The goal of the project is to get the 3D coordinates of the CTR, so that it can be used for control or shape estimation. Epipolar geometry (triangulation) is used to calculate these points as shown on Figure 3.11. The function has 2 inputs, the (x,y) coordinates of the points from both the left and the right rectified image.

Rectification is required in order to have a common image plane for the two cameras. On the rectified image the y coordinates of the corresponding points are the same, therefore the z coordinate can be calculated with triangulation on a 2D plane.

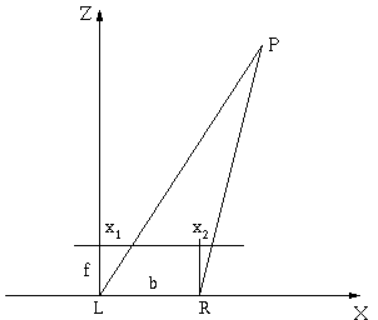


Figure 3.11 Triangulation to calculate z coordinate

Where:

* z – 3rd coordinate of the point in world coordinate system
* f – focal length of the camera [pixel]
* b – baseline, distance between the two cameras [mm]
* - disparity, the x coordinates of the point on the left and right rectified image [pixel]

Since the (x,y) coordinates are in the rectified image planes, transformation is done to obtain the (x,y) coordinates in the world coordinate system in [mm].

Where:

x, y – coordinates of point in world coordinate system [mm]

- coordinates in rectified image coordinate system [pixel]

z – z coordinate in world coordinate system [mm]

f – focal length of the camera [pixel]

Since the 3D coordinates of each point are obtained, a 3D scatter plot shows the reconstructed shape of the robot on Figure 3.12.

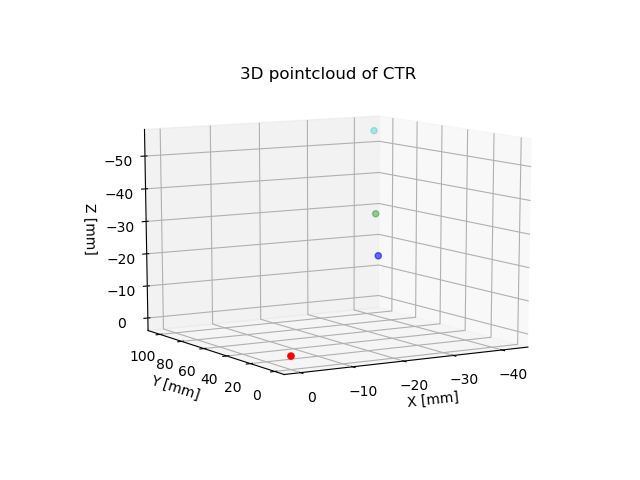


Figure 3.12 The reconstructed 3D points of the CTR joints

3D reconstruction results in the coordinates of each examined point (x,y,z) in the world reference coordinate system in [mm], that is attached to the right camera origin. The resulting 3D coordinates are then validated, by comparing them to electromagnetic (EM) sensor data.

## Graphical user interface

A graphical user interface (GUI) was developed for a mor euser-friendly operation. It is used for giving input, and locating the estimated calibration data. The GUI is shown on Figure 3.13.

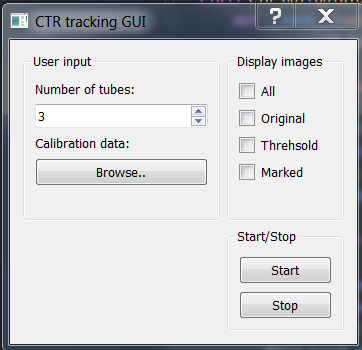


Figure 3.13 The graphical user interface.

In the user iput block, the number of concentric tubes can be given and the folder where the camera calibration data is stored can be chosen, by clicking on browse.

The display images block allows the user to select which views he/she wants to visualize: original is the camera views, threshold is the threshold images after pre-processing, marked is the end result, where the joints and other found points are marked on the undistorted rectified images.

By pressing start, the algorithm runs, and with the stop button it can be stopped.

## Camera holder design

To have a more convenient setup, a stereo camera holder was designed that fixes the two cameras in the same orientation and transformation w.r.t each other, therefore the calibration is not required each time. The camera holder is shown on Figure 3.14.

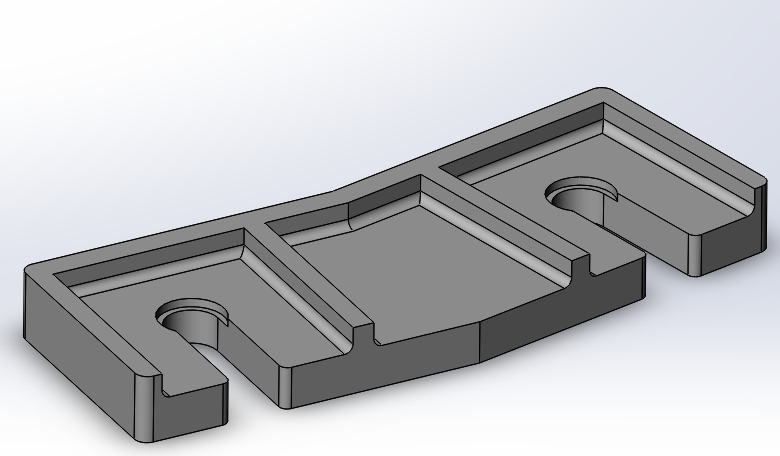


Figure 3.14 Stereo camera holder.

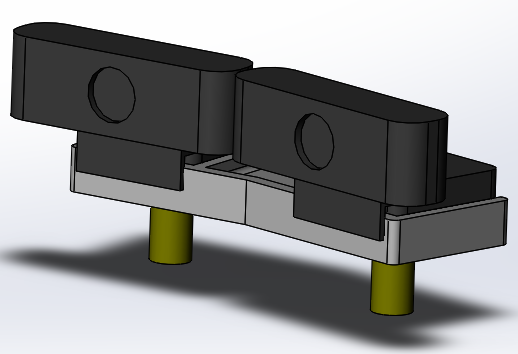


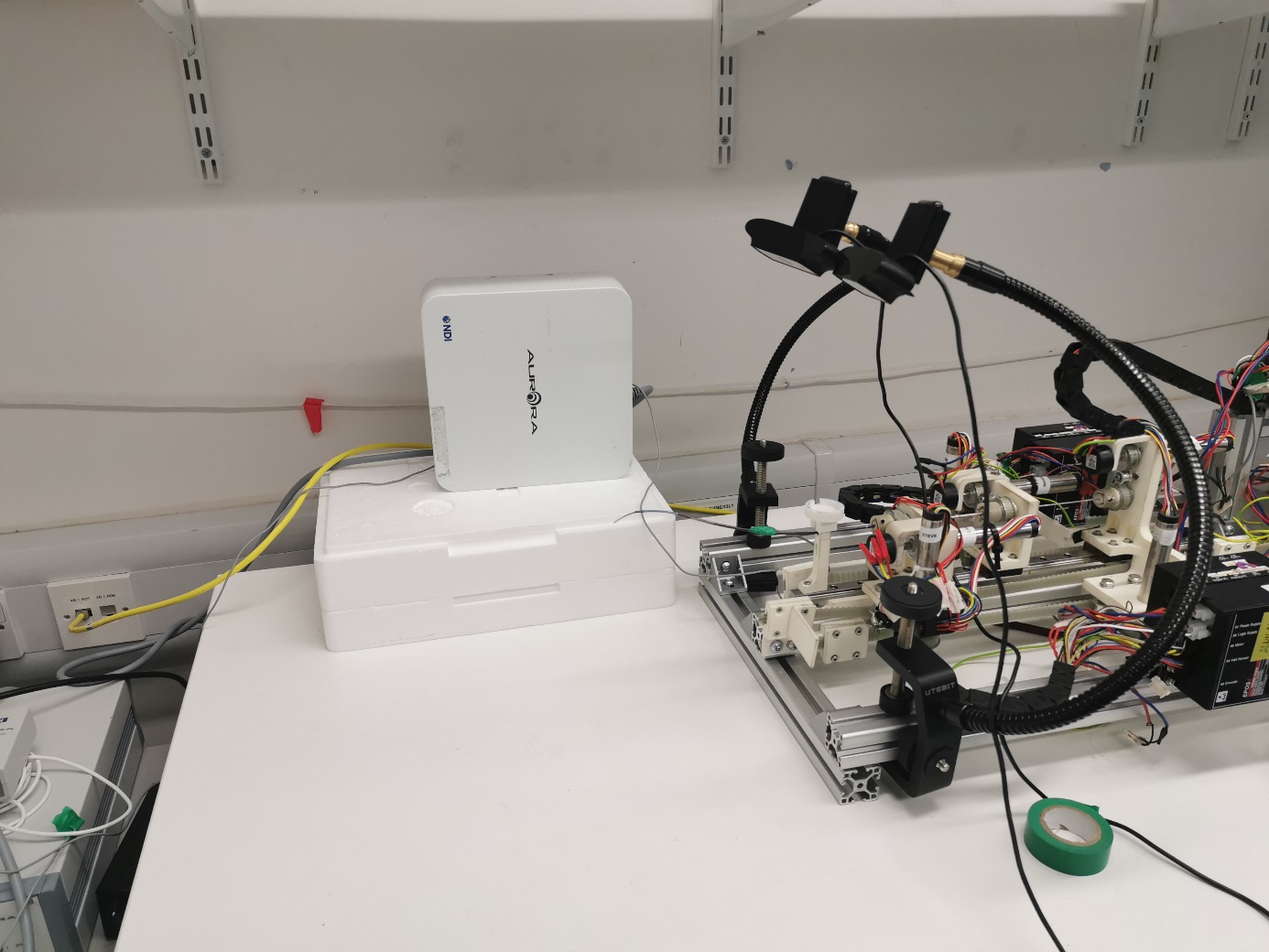
Figure 3.15 The camera holder with cameras and arms.

Cameras can be slid into it, while they are fixed to the flexible arms. Cameras has to face the curved edge, on which their clip can be closed. The model of the holder is also uploaded to github.

# Validation

Experiments were carried out to validate the obtained 3D coordinates of the robot. Ground truth was collected with the use of an electromagnetic sensor, and the two coordinates were compared. A further comparison will be carried out with the mathematical model of the CTR.

The sensor that was used is an NDI Aurora sensor that consists of a field generator and a cable that has 3 coils at the tip. The position and orientation of these coils in the electromagnetic field provides 6 DoF measurement. During these experiment only the x, y and z coordinates were taken into account, the orientation was neglected. The layout of the experiment can be seen on Figure 4.1.



EM sensor

Camera L

Camera R

CTR

Field generator

Figure 4.1 The experiment layout.

Static orientations were examined by first taking stereo image of the robot, calculating the coordinates of 4 points: chosen origin, 2 joints and the tip. The sequence of the experiment was the following:

1. Move robot to a certain orientation
2. Acquire stereo image pairs of the robot and obtain the 3D coordinates of the points
3. Put on the EM sensor and measure the 3D coordinates of the same points
4. Move robot to a new orientation

Where:

* – distance from origin to the i th point [mm]
* – coordinates of joint1, joint2 and tip

The data was collected in excel for both aurora and algorithm coordinates. An example of the collected coordinate data can be found in Table 4.1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Aurora coordinates** | | | | **Algorithm coordinates (CV)** | | | |
| **Origin** | **Joint 1** | **Joint 2** | **Tip** | **Origin** | **Joint 1** | **Joint 2** | **Tip** |
| **X** | 106,24 | 45,03 | 28,47 | 10,36 | -112,63 | -135,3 | -138,57 | -122,47 |
| **Y** | 49,25 | 25,13 | 14,35 | -17,91 | -135,26 | -80 | -61,05 | -17,145 |
| **Z** | -209,24 | -206,58 | -203,9 | -213,14 | -238,2 | -254,35 | -251,51 | -232,06 |

Table 4.1 Obtained 3D coordinates for a certain CTR orientation from Aurora sensor and from the CV algorithm for the origin, joint 1, joint 2 and tip.

Since the reference coordinate system for the vision based and the Aurora sensing is different, the coordinates cannot be compared directly, therefore distances between the origin and the 3 other points were calculated and compared as shown in Table 4.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Aurora**  **[mm]** | **CV**  **[mm]** | **CV/Aurora** | **Mean CV/Aurora** |
| **0 - j1** | 65,84 | 61,87 | **0,94** | **0,91** |
| **0 - j2** | 85,41 | 79,73 | **0,93** | **0,97** |
| **0 - tip** | 117,13 | 118,68 | **1,01** | **1,04** |

Table 4.2 The calculated distances with the Aurora and CV coordinates together with the ratio between the results of the two methods. The last column shows the mean CV/Aurora values based on the 10 measurements.

The distances determined by the algorithm were within a 10 % range of the sensory data for each 10 measurements.

To validate the distance calculations, reference measurements were done with the calibration board, where the length of the rectangle edges are known to be 15 mm. Four corner points of one of the rectangles were measured and horizontal, vertical edges together with a diagonal were calculated. Similarly, 4 measurements were taken by both the algorithm and the Aurora.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Aurora**  **[mm]** | **CV**  **[mm]** | **Ref/Aurora** | **Ref/CV** |
| **Horizontal** | 15,21 | 14,24 | 0,99 | 1,05 |
| **Vertical** | 14,31 | 16,12 | 1,05 | 0,93 |
| **Diagonal** | 21,08 | 19,46 | 1 | 1,08 |

Table 4.3 The length of the dges of the rectangle measured by the Aurora and by the CV algorithm. The reference is 15 mm for the horizontal and vertical edges and ~21 mm for the diagonal.

# Discussion

The code can be found on github: <https://github.com/viktorvoros/CTR3dReconstruction>. I tried to put as many comments as needed to make it easy to understand.

An algorithm was developed for tracking and shape reconstruction of a concentric tube robot by using stereo vision and it was written in python, using openCV. It is able to segment out the robot of the scene and localize the centerline points of the tubes. Using these points, the joint and tip coordinates can be determined by finding local diameter reductions along the robot. After matching the localized points of both images and the use of triangulation, the 3D coordinates can be calculated in the world coordinate system attached to the right camera.

The developed algorithm was validated through experiments, where the obtained coordinates were compared to an electromagnetic sensor coordinates. The distances were calculated between the points and compared resulting in a 10 % of difference w.r.t the sensor data.

The distances are obtained in [mm], and the distance measurement was validated by measuring rectangles on a calibration paper with known side lengths. The algorithm was within 8 % from the reference values, which is a good accuracy for the first version. As the segmentation of the robot will be further improved, the accuracy will improve as well.

The algorithm works in real time with the used webcams that are capable of 20 fps. There is also a version as a ROS node that publishes the distance [mm] of the tip from the origin called CTR3dVideoROS.

# Future work

Recommendations for future work:

* Use of a green background for a color based segmentation instead of using grayscale image.
* Real time tracking of moving robot, comparison to the mathematical model.
* Deep learning for the segmentation of the robot for a more robust algorithm.
* Improve GUI
* GUI number of tubes implementation, adapt code in find joint points, to be able to identify more threshold values than 2.
* Determine transformation matrix between world coordinates and CTR coordinates

# Scripts and files

The overview of the scripts together with their function and all the other files. I tried to put as many comments as needed in the scripts to help understand.

**CameraCalStereoCal.py**

To calibrate cameras separately and stereocalibration

1. Window pops up to take images for first camera calibration
2. Press SPACE to take images (as many times as needed)
3. Images are saved with calibrationcam1\_\*.jpg in the same folder as script
4. Press ESC to finish taking images
5. Calibration takes place automatically
6. New window pops up for second camera calibration
7. Repeat previous steps – images calibrationcam2\_\*.jpg
8. After completing 2 camera calibration separately, stereocalibration is done
9. 2 windows pop up with the 2 camera views
10. Similarly, press SPACE to take images: StereoCalCam1\_\*.jpg and StereoCalCam2\_\*.jpg
11. Press ESC to finish
12. Stereocalibration is done

* All the matrixes are saved in \*.npy format in the same folder as the script
* Camera calibration is only needed once, set isCalibrated = True in line 24 to skip that step.
* Set the calibration board parameters in line 28 and 29 – chessboardSize and rectangleSize

**datacheck.py**

To check calibration data

* Prints out the estimated calibration data, to check whether they are realistic or not
* I usually checked T – translation (x,y,z) distances between the 2 cameras

**steroSnap.py**

To take stereo images

* Same principle as calibration – Press SPACE to take images, when finished, press ESC
* Images saved Cam1\_\*.jpg and Cam2\_\*.jpg in same folder
* !! If you run it again, the images can be overwritten as the numbering starts again !!

**stereoVideoCapture.py**

To take stereo videos

* Run to start recording
* Press Q to finish recording
* Videos saved as OutCam1\_\*.avi and OutCam2\_\*.avi

**saveFrames.py**

To save specific frames of recorded videos

* Same principle as before, video starts playing, press SPACE to save the current frame

**saveAllFrames.py**

To save each frame of video

* Run – saves each frame of video
* !! cameras record with 20 fps – it can result in 1000s of images !!

**CTR3dReconstruction.py**

To make 3D reconstruction from images – The documented algorithm

* Input stereo images and make 3D reconstruction
* Obtain 3D coordinates

**CTR3dVideo.py**

To make 3D reconstruction from videos – previous, but video

* Cameras can be connected directly and it can operate real-time
* Line 10 and 11 – give input
* 0 or 1 is cameras directly,
* or write \*.avi, then it reconstructs from recorded video

**CTR3dVideoROS.py**

ROS node of the previous script

* Subscribes to camera images
* Publishes 3D coordinates and orientation of the points
* RVIZ monitoring

**CTR\_GUI.py**

GUI explained in Chapter 6.

* CTR3dVideoROS.py can be run from here, by pressing start

**camera1ROS.py and camera2ROS.py**

ROS node of cameras

* Publishes images

Images together with the used stereocalibration parameters saved as \*.npy are in folders with names with the dates they were taken, eg. 26112019 – taken on on 26th November 2019. Each folder contains the images used for stereocalibration (stereoCalCam1\_.jpg), as well as the calibration matrixes (\*.npy). Cam1 – left camera, Cam2 – right camera images together with numbering.

Find all these: <https://github.com/viktorvoros/CTR3dReconstruction>

# References

|  |  |
| --- | --- |
| [1] | A. Vandini, C. Bergeles, B.Glocker, P. Giataganad and G.-Z. Yang, "Unified tracking and shape estimation for concentric tube robots," *IEEE Transactions on Robotics.* |
| [2] | Z. Mitros, M. Khadem, C.Seneci, L. DaCruz and C. Bergeles, "Mechanics modelling of eccentrically arranged concentric tubes," *Hamlyn Symposium on Medical Robotics,* pp. 23 - 24, 2018. |
| [3] | R. Grassmann, V. Modes and J. Burgner-Kahrs, "Learning the forward and inverse kinematics of a 6-DOF concentric tube continuum robot in SE(3)," *RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 5125 - 5132, 2018. |
| [4] | R. M. Grassmann and J. Burgner-Kahrs, "On the merit of joint space and orientation representations in learning the forward kinematics in SE(3)," *Robotics: Science and Systems,* 2019. |
| [5] | "openCV documentation," [Online]. Available: https://docs.opencv.org/2.4.13.3/modules/calib3d/doc/camera\_calibration\_and\_3d\_reconstruction.html. [Accessed 09. 12. 2019.]. |
| [6] | D. A. Forsyth and J. Ponce, Computer vision: A modern approach, Pearson, 2003. |
| [7] | [Online]. Available: https://www.azom.com/article.aspx?ArticleID=16003. [Accessed 19 11 2019]. |
| [8] | S. A. R. Flórez, "Contribution by vision systems to multi-sensor object localization and tracking for intelligent vehicles," *PhD thesis,* 2010. |
| [9] | D. B. Camarillo, C. R. Carlson and J. K. Salisbury, "Configuration tracking for continuum manipulators with coupled tendon drive," *IEEE Transactions on Robotics,* no. 25, pp. 798-808, 2009. |
| [10] | M. M. Dalvand, S. Nahavandi and R. D. Howe, "Fast vision-based catheter 3D reconstruction," *Phys. Med. Biol.,* no. 61, pp. 5128-5148, 2016. |
| [11] | A. Vandini, C. Bergeles, F.-Y. Lin és G.-Z. Yang, „Vision-based intraoperative shape sensing of concentric tube robots,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 2603-2610, 2015. |
| [12] | D. Camarillo, K. E. Loewke, C. R. Carlson és J. K. Salisbury, „Vision based 3D shape sensing of flexible manipulators,” *IEEE International Conference on Robotics and Automation,* pp. 2940-2947, 2008. |
| [13] | J. M. Croom, D. C. Rucker, J. M. Romano és R. J. W. III, „Visual sensing of continuum robot shape using sel-roganizing maps,” *IEEE International conference on Robotics and Automation,* pp. 4591-4596, 2010. |