Faculty of Natural and Mathematical Sciences
Department of Engineering

Bush House, King's College
London, Strand Campus, 30
Aldwych, London WC2B 4BG
Telephone 020 7848 2145
Fax 020 7848 2851

7CCSMPRJ

Individual Project Submission 2019/2020

Name:	Mingcong Chen
Student Number:	19007740
Degree Programme:	MSc Robotics
Project Title:	Visual Servoing Control and Modelling for Flexible Endoscopic Robots
Supervisor:	Dr Hongbin Liu
Word count:	10089

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☒ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature:

Date: 21/08/2020

KING'S COLLEGE LONDON

MASTER THESIS

Visual Servoing Control and Modelling for Flexible Endoscopic Robots

---

Author:

Mingcong CHEN

Supervisor:

Dr Hongbin LIU

A thesis submitted in fulfillment of the requirements

for the degree of Master of Science in Robotics

in the

Department of Engineering

Faculty of Natural and Mathematical Sciences

August 21, 2020

Acknowledgments

I would like to express my gratitude to all those who helped me during the writing of this thesis.

My deepest gratitude goes first and foremost to Dr Hongbin Liu, my supervisor, for his constant encouragement and guidance. He has walked me through all the stages of my final project and my MSc study. Without his consistent and illuminating instruction, this thesis could not have reached its present form. Furthermore, I would like to thank the other members of Dr Liu's workgroup, Dr Junghwan Back and Dr Guokai Zhang in particular. Dr Back was beneficial throughout my entire work and patiently supported me. Dr Zhang arranged my project's schedule and provided many valuable suggestions.

I would like to thank my parents Shangyun and Mei for giving me this great opportunity to pursue my Master's degree and supporting me all of my life. Many thanks go as well to my girlfriend Miaowen for motivating and supporting me every single day.

Last but not the least, thank NHS staffs, China anti-epidemic staffs and the medical staff all over the world for the great working against coronavirus.

Abstract

The flexible endoscopic robot has shown great potential in minimally invasive surgery. Compared with conventional open surgery, The endoscopic robot will cause less pain and lower bleeding rates which can help patients recover faster. However, the current endoscopic robot system relies on the surgeon too much that will lead to the lack of surgeon resources and cause danger when the surgeon is getting tired after long-duration operations. Moreover, with the development of 5G, the discussion about the benefit of remote surgery has been revealed to the public. Especially in today's medical situation, due to the coronavirus, the remote surgical system becomes more and more important.

At present, there are already several soft manipulators based on cable drive that has been developed, which allows an accurate movement and force control resulting in safety risk for the patient. To give a more straightforward model for the robot, a helix mesh design of the robot's tip is investigated in this project. A cable-driven actuation system is also developed with a Wi-Fi-based remote controller system. To realize accurate control, the soft robot in this project is modeled by constant curvature to set up the relationship between working space, joint space, and actuator space. Moreover, an inverse kinematics model can solve the visual servoing task is demonstrated based on the differential Jacobian matrix and the estimated Jacobian matrix. All the algorithms are tested in a simulated experiment, and a physical robot experiment for target guided controlling, which shows the desired stable movement. Furthermore, the visual servoing of the soft robot is validated using a painted target showing adequately accurate movement and linear behavior. As an inspiration result, an external contact location estimation algorithm is discussed by an experiment based on visual servoing.

The expected work in the future is the combination with a haptic sensor on the tip of the robot to give a duel-feedback control and the hyper-remote control based on a 5G modular.

Contents

List of Figures

List of tables

Nomenclature

Roman Symbols

l	Length of catheter
k	Radius of curvature
J	Jacobian matrix
e	Error between catheter and target
p	Catheter position

Greek Symbols

α	Rotation angle
β	Bending angle
λ	Damping constant
κ	Step size of iteration

Acronyms / Abbreviations

PWM	Pulse width modulation
UDP	User datagram protocol
TCP	Transmission control protocol
OSI	Open system interconnection
IP	Internet protocol
LAN	Local area network
FPS	Frames per second
TX	Transport
RX	Receive

RGB	Red, Green, Blue
MBR	Minimum bounding rectangle
PC	Personal computer

Chapter I

Introduction

1.1 Motivation

With the development of minimally invasive surgery (MIS), there are more and more flexible endoscopes used to monitor the interior of the human body and help the surgeon do an operation. Compared with the conventional open surgery, the endoscopic robot will cause less pain and less bleeding rate so that complications can be less, and patients can recover faster. For instance, Natural Orifice Transluminal Endoscopic Surgery (NOTES) involves the intentional puncture of the viscera of the stomach, rectum, or vagina with an endoscope to access the abdominal cavity to perform an intra-abdominal operation [1]. In the abdominal cavity, the endoscope will locate the targeted tissue, and the surgery will excise it. After the treatment, the endoscope will be used to send back the incision on the organ before leaving the patients' bodies. However, the surgeon's manual control is not always precise and efficient [48]. Currently, the development of NOTES robot becomes a significant aspect in the field of medical robots. For this project, the focus is the autonomous drive of the flexible endoscopic robot based on visual servoing feedback control. The project gives a structural design of a flexible robot manipulator that is actuated by a cable-driven system with stepper motors, which can give the flexible robot better bending ability and more accurate control. A Wi-Fi-based control system will be introduced to provide the flexible robot a remote operation function. Moreover, to realize the visual servoing control, the soft robot manipulator will be modeled by a forward kinematics algorithm to set up relationships between end-effector working space, joint space and actuator space. Two different Jacobian matrix update algorithm for the inverse kinematics will be introduced to enable the visual servoing function. Then a contact location estimation method is inspired by the visual servoing algorithm, through which can know the obstacle position related to the flexible manipulator. Hence, the image servoing adapted flexible endoscopic robot system can recognize the target area and move by following the target trajectory

automatically so as to increase the efficiency of endoscopic surgery and reduce the risk of medical errors due to the doctors' fatigue.

1.2 Objectives

micro Dexterous Assistive Robotic Therapy (microDART) is a flexible endoscopic robot developed by HaMMeR Lab, King's College London. This project aims to improve the design of the microDART robot system and implement it with image-guided visual servoing. The system was designed to improve the traditional manual surgery during the flexible endoscope minimally invasive surgery. A new design of the flexible robot tip will be developed to improve the robot bending ability, which can make the soft robot control easier and more accurate. Furthermore, a remote control system will be demonstrated to set up a teleoperation system. The image servoing control will be discussed to give an automatic surgery in an endoscopic operation. For example, in endoscopic submucosal dissection (ESD) surgery, image recognition can guide the robot to reach the target tissue and control the flexible instrument to finish the dissection.

1.3 Background

The research of modern robots began in the middle of the 20th century and its technical background in the development of computers and automation. Since the first digital electronic computer came out in 1946, the computer has made fantastic progress and has developed in the direction of high speed, large capacity, and low price. The urgent need for mass production promotes the development of automation technology. One of the results is the birth of an Industrial rigid-body robot. With the rapid development of technology, the intelligent robot has a variety of sensors, which can fuse the information from sensors and adapt to the changing environment effectively. Therefore, it has a robust adaptive ability, learning ability, and autonomous function. According to a review paper [2], the robot manipulators can be classified in there types: Rigid-link, Discrete hyper-redundant, Soft. Rigid-link robots are usually made of metal or plastic, and each joint has an actuator, usually an electric motor. The rigid robot is

generally easier to control, and its precision is higher than the other two types. Although the hyper-redundant is still composed of rigid links, it achieves excellent flexibility through a large number of partial redundant joints. It is a kind of robot that contains more degrees of freedom than the minimum degree of freedom required to complete a specific task. Although the robot, as mentioned above, the system contains a finite number of degrees of freedom, the soft robot is composed of flexible structures, so it has very high flexibility and theoretically unlimited degrees of freedom.

The bionic flexible robot has good bending performance and can change its shape flexibly and smoothly. Its excellent bending performance can even be connected with the antenna of the snake body, nose, octopus, and other biological organs [3] [4]. Because of its good performance, the application prospect of the continuous bionic flexible robot is comprehensive. It can be used to enter the site of bend pipe and earthquake disaster, providing people with people under pressure in gravel, repairing the interior of plants, and diagnosing digestive tract diseases of the human body. At present, researchers all over the world have carried out a series of research on wire driven flexible robots, and have achieved some research results. Simaan has developed a wire driven snake-like robot with flexible support, whose robot is mainly used for minimally invasive surgery of human and animal throat [5]. Ian D. Walker and Michael W. Kerri Hannan have developed a line driven bionic trunk robot, which uses rope and pneumatic drive [6] [7] [8]. Chen developed a pneumatic driven continuous colonoscopy instrument [9] [10].

Robot-assisted surgery was first used in 1985 when the PUMA560 robotic arm was used for fine neurosurgical biopsies, a non-laparoscopic procedure [11]. Since 2004 Kallo published experience of trans gastric peritoneoscopy in a porcine model, the discussion about the benefit of NOTE has been revealed to the public [12]. Yeung and Gourlay, in their review paper, classified flexible multi-tasking platforms as mechanical, which are directly driven by hand and robotic systems with actuators. The differences between the two systems are that for the direct-driven robots, the actuating force comes from the endoscopist. At the same time, actuators drive the actuator-driven robots based on information from a master platform. Direct-driven robots are less bulky and are considerably cheaper because there is no requirement for

any actuators. The control and actuation are in the same device, which makes it less intuitive for the surgeon to control the mechanical system. For actuator-driven robots, due to the use of master and slave systems, the controls for the endoscopist are separated from the actuating means, making their design more straightforward and more ergonomic. A robotic manipulator called Master, And Slave Transluminal Endoscopic Robot (MASTER) from Nanyang Technological University can be used in tandem with a conventional flexible endoscope. The two-armed prototype provids nine degrees of freedoms (DOF) in which the surgeon can get the control of the slave manipulator with seven motorized DOFs, and two non-motorized DOFs can be operated by endoscopist [13]. The MASTER has been tested and demonstrated its ability to perform endoscopic submucosal dissection (ESD) [14]. Another electronically controlled master-slave robot called ViaCath is a long-shafted, flexible, narrow bore instruments which have fixed end-effectors [15]. It has also been used to perform endoscopic mucosal resection in a live porcine model. [16]

The kinematics and dynamics of the flexible robot system are different from the traditional rigid robot. When the robot composed of a series of driving elements, the behavior of these robots tends to be continuous. In theory, the final shape of the robot can be described by a continuous function, and a continuous mathematical model is needed to model this behavior. Since soft robots are different from traditional rigid link-based systems, researchers have developed new static, dynamic, and kinematic models to capture their bending and bending capabilities [17]. Modeling of soft robots is still a significant challenge in current research [18]. Bryan A. Jones gave a differential method and a modified D-H method, which can enable the robot to execute real-time task and shape control by mapping workspace coordinates to actuator inputs, such as tendon lengths and pneumatic pressures, via robot shape [19] [20].

Moreover, the designers also often model the soft robots' kinematics by using a simplified assumption that leads to the constant curvature model [18]. Based on the constant curvature model, the researchers developed a method to map the drive space to the configuration space. These methods are unique to robots because they combine the characteristics of robot morphology and the driving system. However, a constant curvature model cannot cover all

aspects of a soft robot. In order to increase the envelope of the model, a non-curvature model has been developed [21].

The inverse-kinematics problem of the soft robot is more challenging compared with forward kinematics. For a continuum robot with constant curvature, the inverse mapping from task space to configuration space is usually calculated in the robot independent inverse mapping situations [22]. One prominent limitations of the existing methods for solving inverse kinematics problems of linear flexible bodies is that neither the whole body nor the posture of the end effector is considered in the solution, which may be essential for manipulation, sensing, etc. One approach from Neppalli provides simple access to all the solution space of the rigid-link robot, which can be easily applied to an n-link robot [23]. Nevertheless, this method does not account for physical actuation limits like limited actuator lengths. Jacobian matrix provides another method for independent inverse kinematics of the robot. It finds a single solution to the problem from the initial guess through the virtual copy of the servo robot [24][25].

In order to control the motion of a robot, many sensors are involved. Among them, the vision system is an effective way to realize precise motion control of a robot. The visual servo system is an organic combination of robot vision and robot control [26]. It is a nonlinear and robust coupling complex system involving image processing, robot kinematics and dynamics, control theory, and other fields[27]. The existing methods can be divided into two categories: (1) position-based visual servoing (PBVS), in which feedback is defined according to the 3D Cartesian information from the image; (2) image-based visual servoing (IBVS), where feedback is defined directly in the image according to the image features [28] [29]. Visual servoing has been widely studied in the past decades, and many methods, such as the homography method, have been proposed [30]. An approach based on the concept of a depth-independent interaction matrix was developed by Y. H. Liu and H. Wang [31][32][33]. With the development of a soft robot, it is worth developing a visual servoing algorithm for soft robots. H. Wang also proposed a visual servo control method for a cable driven flexible robot in 2013[34], which shows the enormous potential of applicability in medical surgery and complex environments exploration.

1.4 Project Outline

In chapter 2, the hardware and algorithm design for the microDART robot are demonstrated. In the hardware design part, a tendon driven robot end-effector design is demonstrated. Due to the tendon driven structure, a cable-driven actuation system is developed. Stepper motors control the cable-driven system as an open-loop control based on a mapping relationship between the cable screw platform position and step numbers of the stepper motor. To provide a wireless control function, a Wi-Fi modular is set up with a microcontroller to receive the command from the PC and control the robot. Moreover, for a visual servoing system, the robot end-effector and target tracking system are developed due to image recognition. To model the movement of microDART, a constant curvature kinematics model is introduced to give the relationship among end-effector working space, joint space, and actuator space. The visual servoing algorithm is converted to be the error in damped least squares inverse kinematics model whose results are calculated from the target and end-effector position difference and Jacobian matrix. Moreover, to avoid the irregular shape change during servoing, an estimated Jacobian matrix is demonstrated to compare with the differential Jacobian matrix method.

In chapter 3, the physical experiment is done to test the constant curvature model which leads the catheter to move to an expected position as the model's result. To test the inverse kinematics, a rectangular trajectory is designed to give a target for the system. Both the differential Jacobian matrix and estimated Jacobian matrix methods are tested on a MATLAB simulation environment and a physical robot. For the visual servoing task, the robot is guided by the inverse kinematics model and move to a target drawn on a paper. The catheter position is tracked by an image recognition algorithm to give numerical data for analyzing process. A contact location estimation will be discussed as an inspiration form the visual servoing algorithm.

In chapter 4, the outcome of this project is discussed. The design and behavior of the modified microDART system is discussed due to its performance during the project experiment. The visual servoing system under the forward and inverse kinematics algorithms are discussed with

the state-of-the-art endoscopic navigation system. In future work, the 5G hyper-remote control and haptic feedback control are worth discussing.

Chapter II

Design

2.1 System Design

MicroDART is a multi-DOF flexible endoscopic robot. The robot's hardware contains six main parts, including catheter, camera, multi-channel tube, tendons, actuators, and control box. The catheter is a high-performance nylon 3-D printed structure that can provide the robot bending ability. Tendon is a steel wireline used to pass through the multi-channel tube which connected with the catheter to control the bending angle. Five stepper motors linked to screw platforms act as actuators for the system. Motor drivers and controllers with Wi-Fi modular are located in the control box. Fig.2.1 and Fig.2.2 show the whole microDART robot system.

The control algorithm of the robot is image-guided feedback control. At first, to control the robot, a forward kinematics model called constant curvature is introduced, which can set up the relationship among end-effector working space, joint space, and driven space. PC can recognize the catheter tip position and target position by real-time video stream, in which the position error in the image can be calculated. As well as the error from image inputs into an inverse kinematics method based on the Jacobian matrix and target error to get the joints to pose. During the movement, the Jacobian matrix will keep updating until the robot reaches the target and stop moving. The whole logic of the microDART visual servoing system is shown as Fig.2.3.

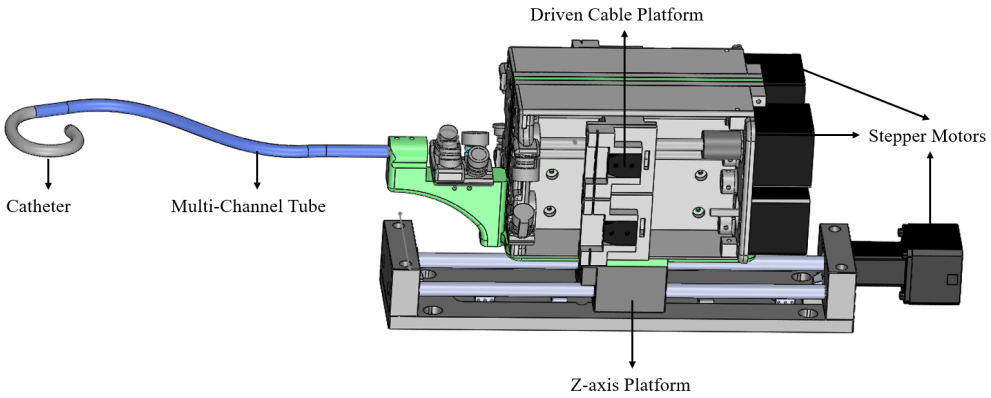


Fig.2.1 Drawing of microDART robot system

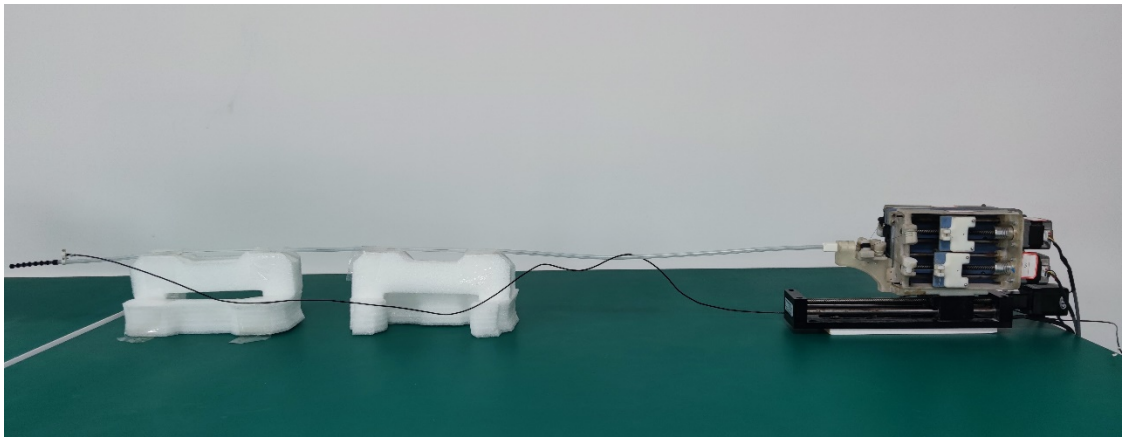


Fig.2.2 Assembled microDART robot system

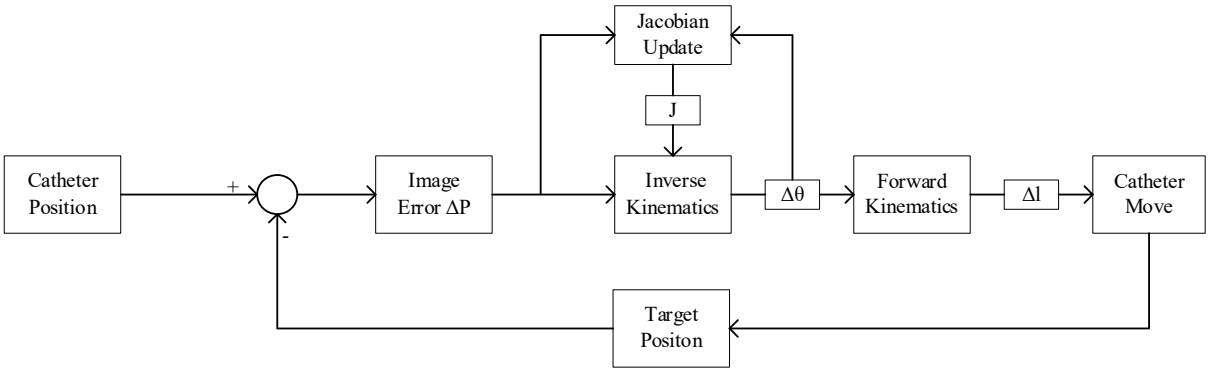


Fig.2.3 Algorithm overview

2.1.1 Flexible Manipulator Design

The manipulator of microDART is called catheter, which can control the robot direction by tension tendons. As Fig.2.4 shown, the catheter gets tension from the driven tendons then the structure of the catheter will get some deformation, which performs as bending ability of the plastic part (the core part of catheter design). To protect the catheter under high tension, the material of the catheter is high-performance nylon which can provide $48MPa/m^2$ tensile strength.

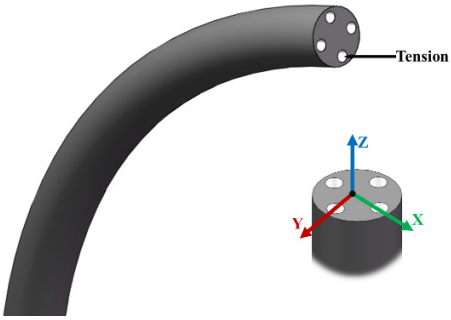


Fig.2.4 microDART Manipulator bending

(a) Disk Cutting original design (b) Compression when applied tension

Fig.2.5 Disk Cutting Design

Fig.2.6 Helix Design

As Fig.2.5(a) shown, the design of the catheter is called Disk Cutting. The main cylinder body of the catheter is cut by disk-like to give space for deformation. The expected situation is that the disk will move to the space under it when tension applies to it. However, the tension will pass through the connected wall between disks, which will cause the deformation of the connection wall to make an irregular shape of the catheter.

The final design of the catheter is shown in Fig.2.6. It is composed of seven units, and each unit is composed of one disk and one helix bar. The disk on the top is called base disk, and the bottom one is the terminal disk. Each joint has eight disk holes whose usage is for driven cables and optical fibers. Each unit together to create a continuous soft mechanism under the driven of disks.

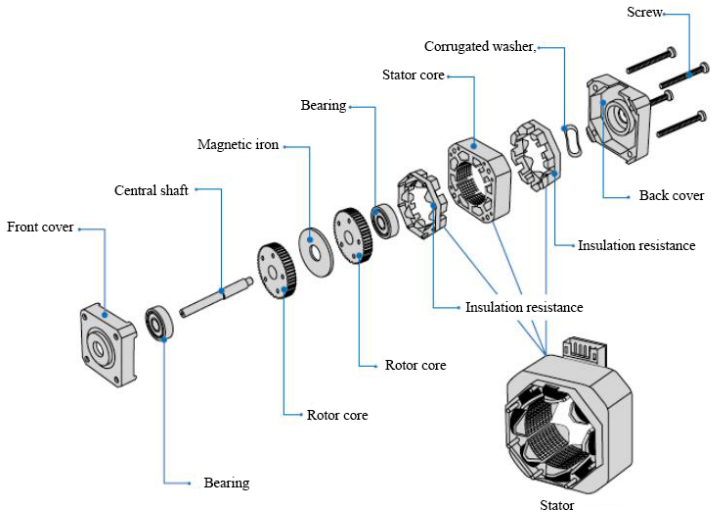


Fig.2.7 Stepper motor structure

2.1.2 Actuation

Stepper motor is a kind of motor which converts electric pulse signal into corresponding angular displacement or linear displacement. When a pulse signal is an input, the rotor will rotate an angle or advance one step. The angular displacement or linear displacement of the rotor output is directly proportional to the number of input pulses, and the rotational speed is proportional to the pulse frequency. Stepper motor is generally composed of front and end covers, bearings, central shaft, rotor iron core, stator core, stator components, corrugated washer, screw, and other parts. As Fig.2.7 shown, the stepper motor uses the electromagnetic principle to convert electrical energy into mechanical energy, which is driven by the coil wound on the stator slot of the motor. Normally, the coil of wire is called a solenoid, while in a motor, the wire wound in the stator slot is called a winding, coil, or phase.

To generate a strong enough magnetic field to control the stepper motors. An external motor driver, 2M542-N produced by SainSmart is selected. The driver provided three different digital inputs for function control, which are PLC+/-, DIR+/- and ENA+/-. PLC+/- is for clocked square-wave pulse signal whose frequency controlled the angular velocity of the motor. DIR+/- defined the rotation direction of the motor while Low-level input counter-clockwise and High-level input for clockwise motion and ENA+/- is the enabling for the motor.

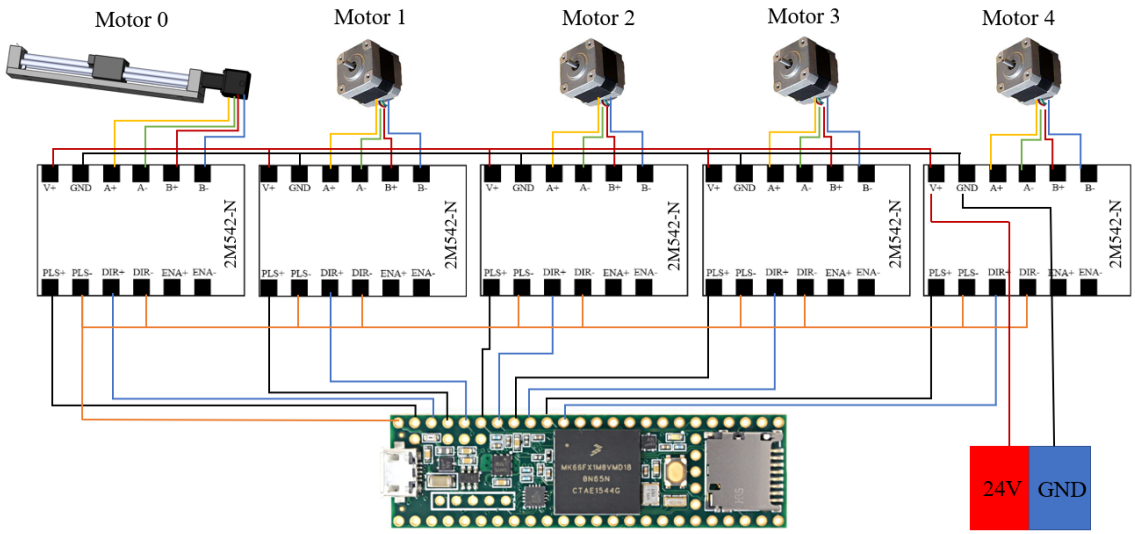


Fig.2.8 Stepper motors connection

The microcontroller for the actuation system is Teensy 3.6, whose processor is a 32 bit 180 MHz ARM Cortex-M4. As shown in Fig.2.8, the I/O port PIN0-PIN9 of Teensy 3.6 is connected to the stepper motor drivers, in which PIN0, PIN2, PIN4, PIN6, PIN8 generate pulse width modulation (PWM) signal to control the velocity with connecting with PLC+. PIN1 PIN3, PIN5, PIN7, PIN9 connect to DIR+ to control the rotation direction. The PLS- and DIR- ports of stepper motor drivers are wired to the ground (GND) of Teensy. The MicroDART robot placed on the platform can be moved by motor 0 to provide the Z-axis displacement. Motor 1-4 can control the driven cable, which can give bending ability of the catheter so that the tip of the catheter can change the position in the X-Y plane.

To give the strain to the catheter tip, four steel cable was connected to the tip with a platform that can be moved on stepper motors on the other side. To give an accuracy control to the cable, a converted model from the stepper motorcycle and platform displacement is set in the microcontroller code.

2.1.3 Communication

Internet protocol set supports a connectionless transport protocol, which is called user datagram protocol (UDP). UDP provides a way for applications to send encapsulated IP packets without establishing a connection. UDP protocol is used to process packets as TCP protocol. In the OSI model, both of them are located in the transport layer, which is on the upper layer of the IP

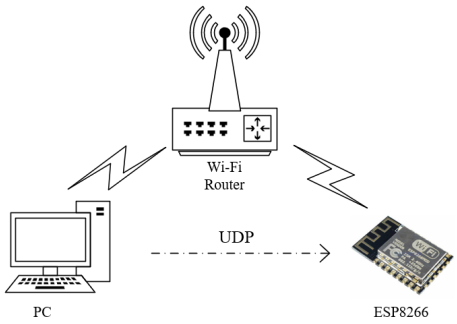


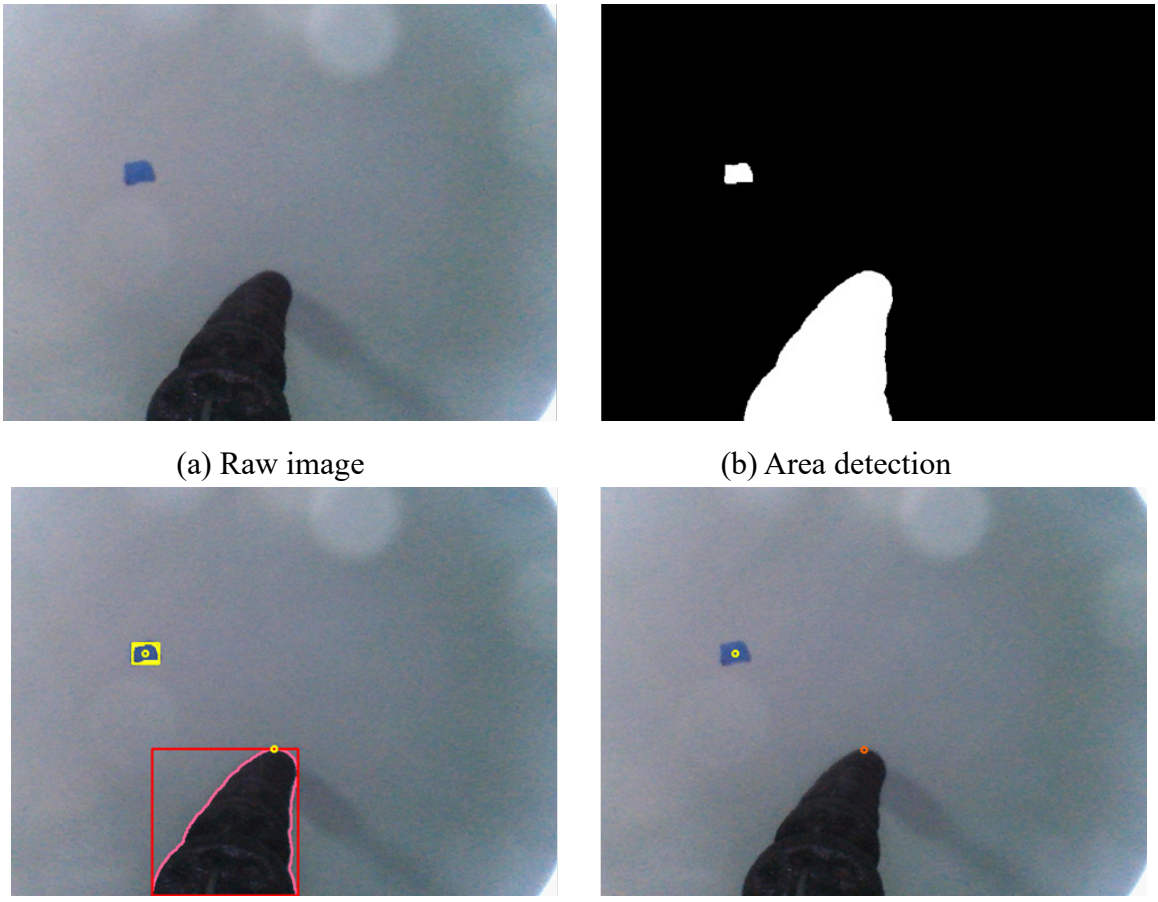
Fig.2.9 Communication logic of microDART

protocol. UDP has the disadvantage of not providing packet Grouping, assembling, and sorting packets, which means when a message is sent, and it is impossible to know whether it arrives safely and completely. UDP message has no reliability guarantee, sequence guarantee, and flow control field, so its reliability is poor. However, due to the fewer control options of the UDP protocol, the delay is small, and the data transmission efficiency is high. For the MicroDART wireless control, this project, the robot, and the PC both connected to the Wi-Fi router, which is an unstable connection. For Transmission Control Protocol (TCP), the connection occurs with a packet handshake. If the connection is lost due to the Wi-Fi signal, the robot will stop moving until the connection rebuilt up. Hence, as Fig.2.9 shows, the communication system is designed as the robot, and PC connect to the same router who can provide UDP communication on local area networks (LAN).

To give MicroDART the ability of connection with Wi-Fi, an external control board NodeMcu, which is developed based on an IoT Wi-Fi modular ESP8266, was connected to MicroDART central controller Teensy. The serial port TX/RX of NodeMcu is wired to Teensy's RX5/TX5 port. A byte command package, which includes 22 bytes, is created for sending data in UDP. As Table 2.1 shows, the package contains one-byte header as the message start flag, and 20 bytes hold the motor movement command and one-byte terminal flag.

	Header	Motor 1	Motor 2	Motor 3	Motor 4	Motor 0	Terminal
Encoded	0x24	Byte[4]	Byte[4]	Byte[4]	Byte[4]	Byte[4]	0x26
Decoded	$	Float type	Float type	Float type	Float type	Float type	&

Table 2.1 Communication protocol



(a) Raw image (b) Area detection

(c) MRB detection (d) Result image

Fig.2.10 Catheter and target detection

2.2 Image Sensing

The simulation task for microDART in this project is detecting a blue target on a white paper. The camera is located at the end of the multi-channel tube. This configuration allows the camera to gain the view of the whole catheter and blue target. The algorithm can be separated into two parts. Part one consists of the pre-processing and area extraction, and part two recognize the target and catheter position by the area extracted from part one.

For part one, the raw camera image is split into RGB channels and convert to a binary image after filtering by a threshold. The raw blue target area is the subtraction of blue and red channels—the inverse of summed up RGB channel intensity set as the catheter area. The two images are then processed by Gaussian blur, followed by the open morphological operation to reduce noise and give continuous area results as Fig.2.10 (a) and (b) shows.

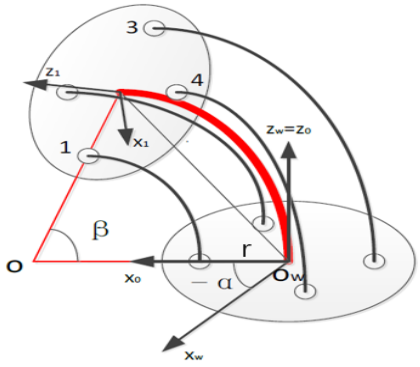


Fig.2.11 Constant curvature modeled catheter

For both target and catheter, using the Suzuki border following algorithm to extract contours from the result area in part one[35]. Furthermore, get the minimum bounding rectangle (MBR) of the areas by rotating calipers [36]. Both contours detection and MBR algorithm are implemented in the OpenCV library. For the blue target, the center of the MBR result rectangle is the target point. For the catheter, the functions of two shorter sides of the rectangle can be calculated as $y = ax + b$. The line with a higher bias value b will be selected then the average of contacted points in the area with the selected line is the catheter tip point.

2.3 Forward Kinematics

The traditional rigid-body robot can be modeled and analyzed by Denavit- Hartenberg matrix. Nevertheless, for a flexible structure robot like MicroDART in this project does not have rotation and translation joints. For this project, the kinematics analysis algorithm is the Constant Curvature Model. For constant curvature model, (1) assume that the catheter is equal to the smooth curvature continuous curve; (2) the weight of the whole catheter and driven cable is negligible; (3) assume that during the bending process the driven cable is equal curvature of the curve [37].

The original of the coordinator system is in the center of the base disk, and the Z-axis is perpendicular to the support disk and points to the longitudinal extension. The X-axis points to the first driven cable hole of the disk. The Y-axis of the coordinate system is set due to the right-hand rule. The geometric model of a single unit of the catheter is shown in Fig.2.11, where α is the rotation angle, and β is the bending angle.

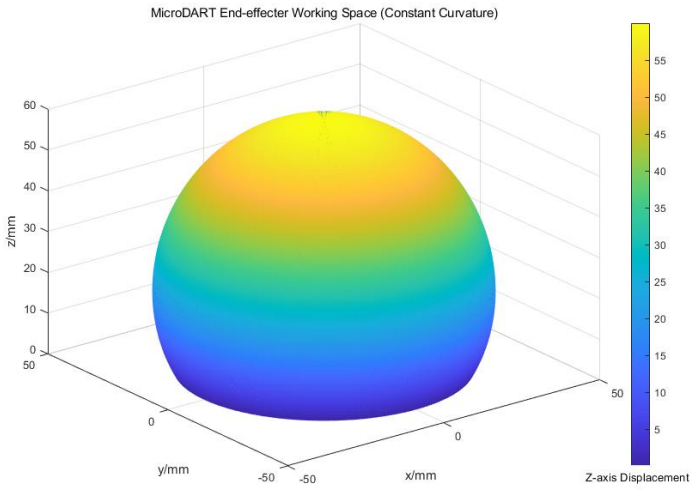


Fig.2.12 microDART end-effector working space

The driven cable is connected to four stepper motors, so the kinematics analysis should not only include the relationship between joint space and operating space but also includes the transformation from driven system to joint space. Hence, the kinematics problem of MicroDART can be divided into two parts: First, find the relationship between the end-effector position and the angle α and β. Second, deduce the transformation from the driven cable length change Δl and angle α and β.

The transforming part of kinematics is from the translation of the coordinate system from base to termina, rotate Z-axis, then rotate Y axis followed by rotating Z axis again. The transformation matrix is shown below:

$$T = Trans\left(\frac{l}{\beta}cos\alpha(1-cos\beta), \frac{l}{\beta}sin\alpha(1-cos\beta), \frac{l}{\beta}sin\,\beta\right) Rot(z,\alpha)Rot(y,\beta)Ros(z,-\alpha)$$

$$= \begin{bmatrix} cos^2\alpha + sin^2\beta & cos\alpha\, sin\alpha\, cos\beta - cos\alpha\, sin\alpha & cos\alpha sin\beta & \frac{l}{\beta}cos\alpha(1-cos\beta) \\ sin\alpha cos\beta - cos\alpha\, sin\alpha & sin^2\alpha cos\beta + cos^2\alpha & sin\alpha sin\beta & \frac{l}{\beta}sin\alpha(1-cos\beta) \\ -cos\alpha\, sin\beta & -sin\alpha cos\beta & cos\beta & \frac{l}{\beta}sin\beta \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

According to the formula, the equation of rotation angle and bending angle can be concluded as:

$$\alpha = arctan\left(\frac{p_y}{p_x}\right) \tag{2}$$

$$\beta = arccos(\alpha) \tag{3}$$

The range of bending angle β is $(0, \pi]$. Hence, we can calculate the range if α is $[0,2\pi)$.

The length of the catheter of the microDART design is 60 mm. So the working space of microDART end-effector can be analyzed by formula (1), the parameters in the formula are: $l = 60mm, \alpha \in [0,2\pi), \beta \in (0, \pi]$. Draw the working space in MATLAB, as shown in Fig.2.12.

The result of MicroDART, working space simulation, it looked like a three fourth ball. However, during the calculation of the simulated shape, there is no consideration about the z-axis displacement. In that case, if the MicroDART can move in the z-axis direction, the working space should be a cylinder-like shape.

To simplify the calculation, the bending curve of a single element is set to be the equivalent curve. Because of the offset between the driven cable hole and the center of the disk, the curvature radius is different even their bending angles are the same. The first drive wire length can be concluded by formula (4) as:

$$\Delta l_1 = l - l_1 = \left(\frac{1}{k} - \frac{1}{k_1}\right)\beta = \frac{k_1 - k}{kk_1}\beta = r_1\beta \tag{4}$$

Where, k, k_1 is the curvature of the catheter and r_1 represents the distance between the driven cable and the center of the disk. Therefore, when the catheter's bending angle is β, and the rotation angle is α, we can get the four driven cable variation by the formulas (5) as below:

$$\begin{cases} \Delta l_1 = r_1\beta cos\alpha \\ \Delta l_2 = r_2\beta cos\left(\alpha - \frac{\pi}{2}\right) \\ \Delta l_3 = r_3\beta cos(\alpha - \pi) \\ \Delta l_4 = r_4\beta cos\left(\alpha - \frac{3\pi}{2}\right) \end{cases} \tag{5}$$

Now we have the mapping relationship between joint working space and the driver space. It is

also easy to map the formula from drive space to joint space as:

$$\alpha = arctan\left(-\frac{\Delta l_2}{\Delta l_3}\right) \tag{6}$$

$$\beta = \frac{\Delta l_1}{r_1 cos\alpha} \tag{7}$$

The range of α is $[0,2\pi]$, so we can obtain the two solution difference of formula (6) is π. However, form formula (5) we can conclude that if $\Delta l_1 < 0$, $\alpha \in [\frac{\pi}{2}, \frac{3\pi}{2}]$, and if $\Delta l_1 > 0$, then $\alpha \in [0, \frac{\pi}{2}] \cap [\frac{3\pi}{2}, 2\pi)$. Now we can conclude the only solution of rotation angle α. Hence, we can calculate the value of β by the only solution of α.

2.4 Inverse kinematics

To control the movement of a robot, the standard way is using inverse kinematics. The inverse kinematics algorithm of MicroDART in this project is the damped least squares method [38][39]. One way to solve the problem of whether there are component solutions related to small singular values is to balance the cost of the large solution and the cost of sizeable residual error by minimizing their sum[40].

$$||J\Delta\theta - \vec{e}||^2 + \lambda^2||\Delta\theta||^2 \tag{8}$$

This rewrite of the original equation is known as damped least squares [41]. This can be written as

$$||\begin{bmatrix} J \\ \lambda I \end{bmatrix}\Delta\theta - \begin{bmatrix} \vec{e} \\ 0 \end{bmatrix}||^2 \tag{9}$$

The unique minimizer angle change is shown as:

$$\Delta\theta = [\Delta\alpha, \Delta\beta] = (J^T J + \lambda^2 I)^{-1} J^T \vec{e} \tag{10}$$

Both sides of the equation are equally multiplied by $(J^T J + \lambda^2 I)$, we can find

$(J^TJ + \lambda^2 I)^{-1}J^T = J^T(JJ^T + \lambda^2 I)^{-1}$, hence formula(10) can be written as

$$\Delta\theta = [\Delta\alpha, \Delta\beta] = J^T(JJ^T + \lambda^2 I)^{-1}\vec{e} \tag{11}$$

Where $\lambda \in \mathbb{R}$ is non-zero damping constant, J is the Jacobian matrix. J^TJ and I are $n \times n$ matrix where n is the degrees of freedom. $\vec{e}$ is the difference between target and end-effector position.

The Jacobian matrix is a function of the angles θ values defined by

$$J(\theta) = \left(\frac{\partial X_i}{\partial \theta_j}\right)_{ij} \tag{12}$$

Where ∂X_i is the end-effector speed. It is easy to get the Jacobian matrix by calculating the differential of the forward kinematics formula (1). Let us give a name of this method as the "differential Jacobian matrix". In the next section, another method called the "estimated Jacobian matrix" will be introduced. The result of the differential Jacobian matrix is shown below:

$$J(\theta) = \begin{bmatrix} \frac{l}{\beta}sin\alpha(cos\beta - 1) & \frac{l}{\beta}cos\alpha sin\beta + \frac{l}{\beta^2}cos\alpha(cos\beta - 1) \\ -\frac{l}{\beta}cos\alpha(cos\beta - 1) & \frac{l}{\beta^2}sin\alpha(cos\beta - 1) + \frac{l}{\beta}sin\alpha sin\beta \\ 0 & \frac{l}{\beta}cos\beta - \frac{l}{\beta^2}sin\beta \end{bmatrix} \tag{13}$$

Hence, we can update the Jacobian matrix after each movement by the formula(13) and input the new calculated Jacobian matrix to next time inverse kinematics calculation.

2.5 Jacobian Matrix Update

In the visual servo task, the kinematic model is considered to be inaccurate because of the uncertainty of the external contact force. This section introduces an "Uncalibrated visual servoing" [42], which can help us update the Jacobian matrix for damped least squares inverse

kinematics. It enables robot manipulation without the pre-known kinematics relationship and camera parameters[43] [44] [45]. In this kind of algorithm, the Jacobian matrix is directly estimated from the image motion. We call this method the “estimated Jacobian matrix”.

Assume the joints state vector change $\Delta\theta$ of the MicroDART robot at a time t_i is known, whose state vector is the rotation α and bending angle β. We equipped a camera on the tip of the catheter of MicroDART, which can track the position of the tip. As a result, we can know the pose Δp as time t_i, the formula without any knowledge about the robot kinematics can be demonstrated as below.

$$\Delta p \cong J(\theta)\Delta\theta \tag{14}$$

By Broyden-type rank, one updating the estimated Jacobian matrix can be calculated by the following formula(15)

$$J_{i+1} = J_i + \kappa \frac{\Delta p - J_i \Delta\theta}{\Delta\theta^T \Delta\theta} \Delta\theta^T \tag{15}$$

Factor κ is the step size of iteration, and it needs to be related to the system’s sampling frequency, which is the video frame FPS. It can be shown that the Jacobian J converges when remains similar between two iteration steps. Until now, the primary visual servoing algorithm was established. The next chapter will demonstrate some simulation and real-world experiments to evaluate the methods in Chapter 3.

2.6 Visual Servoing

In the last section, an inverse kinematics solution for the manipulator was introduced, which can let the visual servoing system in this project easy to solve. For the visual servoing system, the main idea is to minimize the error between the end-effector and target through the image like the equation below.

$$e(t) = s(I(t), a) - s^* \tag{16}$$

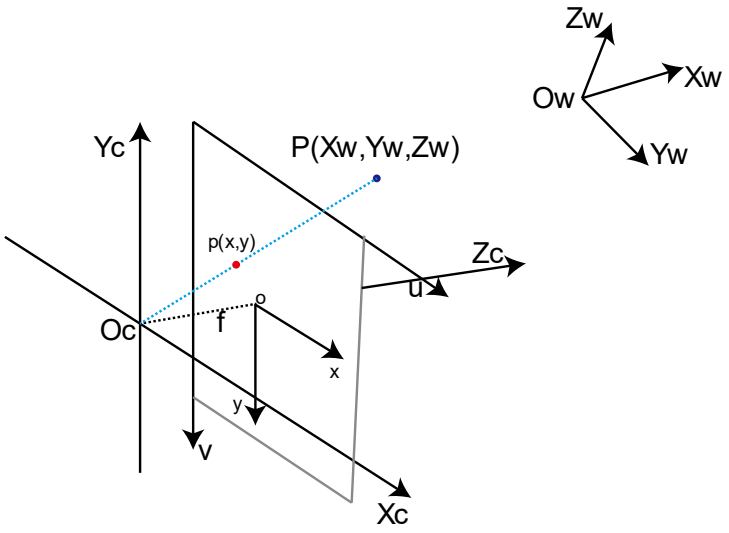


Fig.2.13 Coordinates in a camera system

In which s is the features extracted from the image $I(t)$ at time t based on system hardware information a like camera parameters and s^* is the target feature.

For the damped least squares method in section 2.4, it has already been optimized to minimize the end-effector error and target error in the working space. Hence the problem becomes how to convert the working space (or real-world coordinates) to camera pixel space. This problem should involve four different coordinates systems: world coordinate system, camera coordinate system, image coordinate system, and pixel coordinate system.

Where the world coordinate system is $O_w - X_w Y_w Z_w$ described the camera position in the real world, camera coordinate system $O_c - X_c Y_c Z_c$ whose original point is the optical center, image coordinate system $o - xy$ image coordinate system with the optical center as image center point, and pixel coordinate system uv whose original point is the top left corner. Then through the transformation of the above four coordinate systems, we can get how a point transforms from the world coordinate system to the pixel coordinate system as the formula below.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \vec{I}\vec{O} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{17}$$

Where $\vec{I}$ is camera internal parameters and $\vec{O}$ is extrinsic camera parameters.

Chapter III

Experiment and Results

3.1 Forward kinematics

There is no sensor to measure the distance between the catheter tip and the target plane. So for the experiment in this and rest sections, we only considered the end-effector's movement in the X-Y plane. The catheter tip will be manually put at the position where it can contact with the target plane. In this situation, as the length of the catheter is 60mm, the start end-effector position should be [0,0,60]. As the simulation Fig.is shown in Section 2.3, when the catheter applies a movement in X-axis and Y-axis, it will perform a backward move in Z-axis. Hence, to compensate for the Z-axis movement, which means let the catheter tip keep contacting with the working plane, the motor 0, which controls the Z-axis position will move when there is any movement applied on X-axis and Y-axis. The value of Z-axis compensation can be calculated by the formula(1) as $l - \frac{l}{\beta} sin(\beta)$ in which the length l is 56mm.

To test the forward kinematics model, a 3mm square drawn on a white paper as a reference. The first test is trying to pull the right driven cable backward 4mm. At the same time due to the formula(5), we can get if $r_1 = r_3 = r$ then $\Delta l_3 = r\beta cos(\alpha - \pi) = -r\beta cos\alpha = -\Delta l_1$. Hence, the left driven cable should be pushed forward 4mm. To avoid unlimited solutions during the calculation, the up and down driven cable was given a minimal value as 0.001mm and -0.001mm. The expected angle change calculated from constant curvature of this cable configuration is $\alpha = arctan\left(-\frac{0.001}{40}\right) = -2.5 \times 10^{-5}, \beta = \frac{\Delta l_3}{rcos\alpha} = 14.2857$. Then convert the joint change into the working space. The end-effector position should be [30.16,0.00075,34.47] mm. The movement procedure from the first view camera of microDART is shown in Fig.3.1.

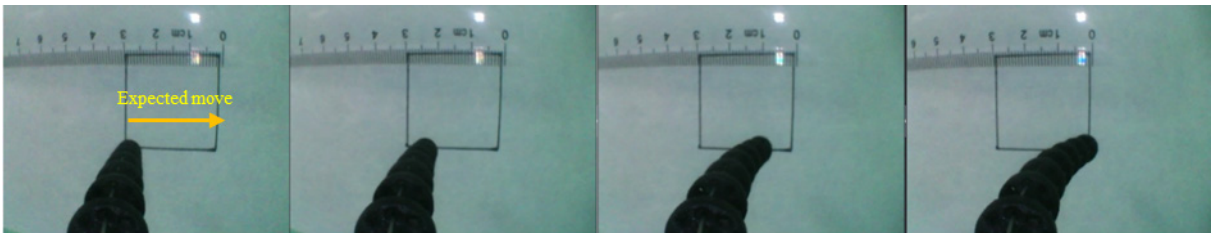


Fig.3.1 Catheter movement procedure

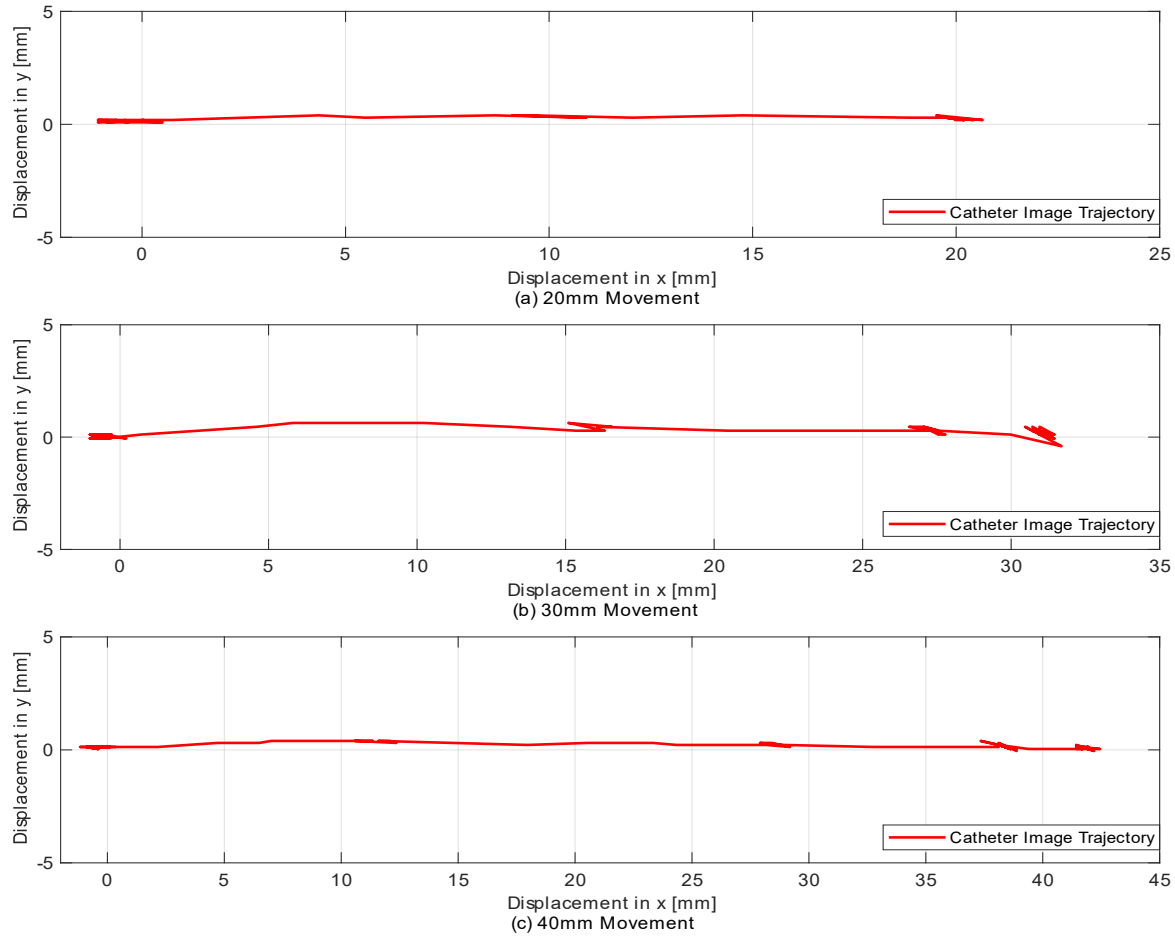


Fig.3.2 Catheter trajectory under command from forward kinematics

As the Fig.3.1 is shown, the catheter move from the left of the box to the right of it. The length of the box side is 3cm, and the base platform moved forward 21.53mm to keep contact with the target plane. The expected trajectory of the catheter is to move 30.16mm from left to right, and through the robot's first view, we can see the robot finished the task correctly. Moreover, through the photo of the top view, we can also see that the base platform succeeds in giving a Z-axis compensation, and the shape of each segment of the catheter keeps a constant curvature.

The catheter was controlled to reach several movement targets as moving to right 20mm, to

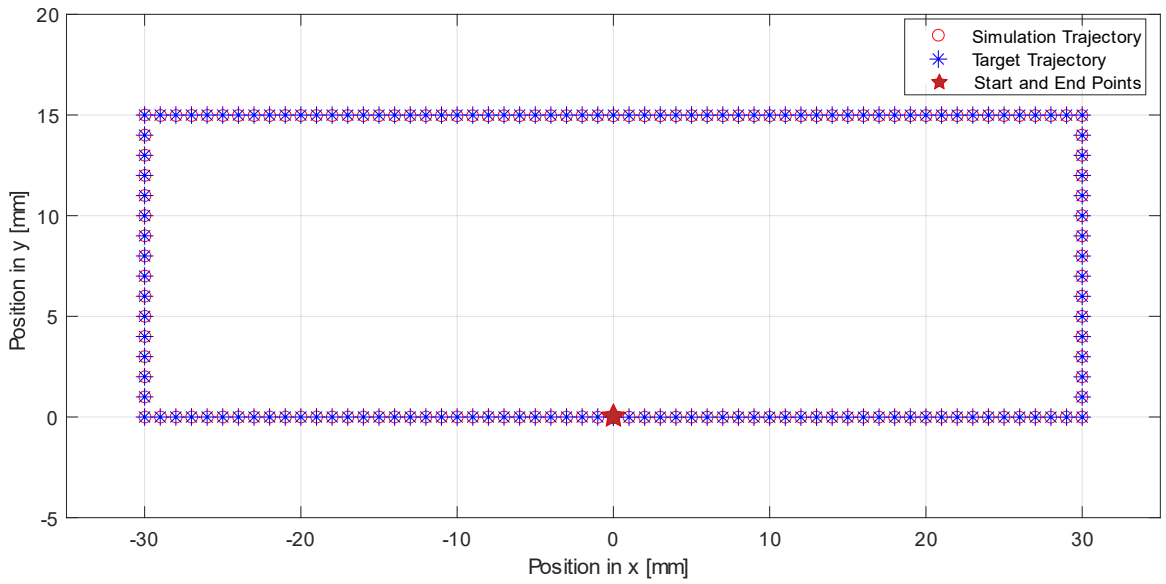


Fig.3.3 Target and simulated trajectory by differential Jacobian matrix method

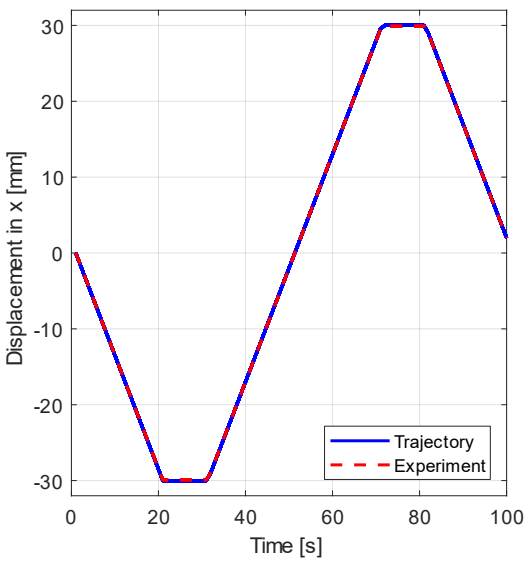
right 30mm and to right 40mm. Figure 3.2 shows the catheter tip recoded by camera with the algorithm in section 2.2. All the three commands were perfectly executed by the microDART robot. It is noticed that the error became larger with the distance increasing, which are 0.5mm in 20mm movement, 1.5mm in 30mm and 2mm in 40mm.

In this section, the constant curvature model shows an excellent performance about mapping the relationship between driven cable space, joint space, and end-effector working space. To finish the visual servoing task, a more numerical experiment is done in the next sections.

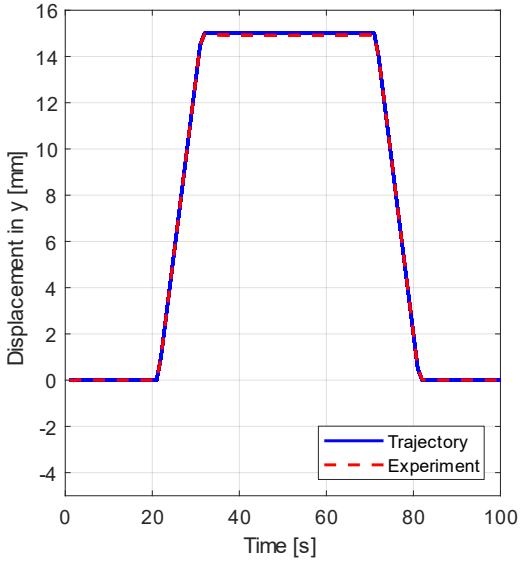
3.2 Inverse kinematics

To test the algorithm, a rectangular trajectory is defined as the end-effector target position, as Fig.2.3.1 shown. There are five parts in the movement of the trajectory:

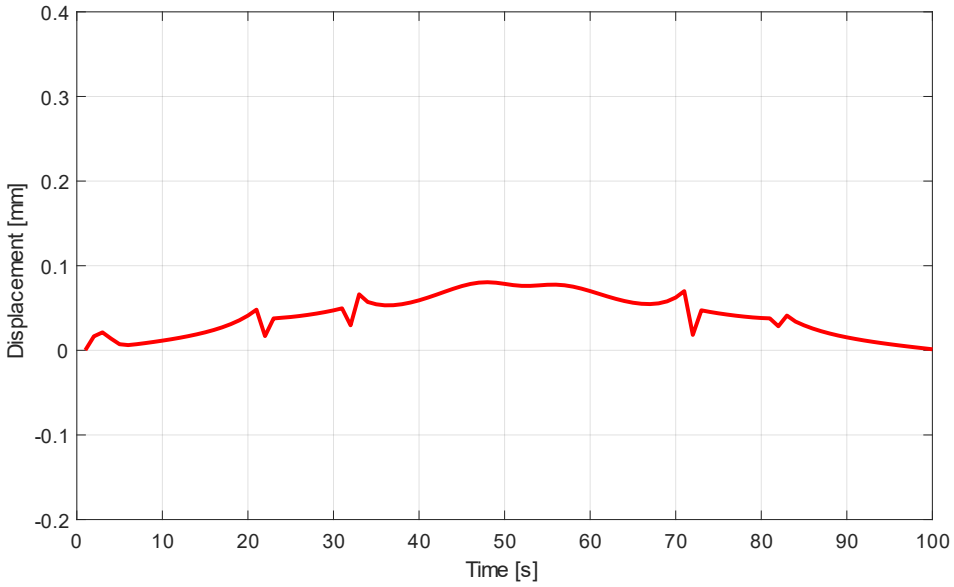
1. Start in (0,0), move alone x-axis from 0mm to -30mm.
2. Move alone y-axis from 0mm to 15mm reach the point (-30,15).
3. Move alone x-axis from -30m to 30mm get point (30,15).
4. Move alone y-axis from 15mm to 0mm.
5. Move alone x-axis from 30mm to 0mm back to the original point (0,0).



(a) Displacement in x over time



(b) Displacement in y over time



(c) Resultant error in [x,y] over time

Fig.3.4 Inverse kinematics simulation based on differential Jacobian matrix

Use the inverse kinematics we discussed before to follow the triangular traction. The comparison of the expected and temporary position in x-displacement and y-displacement is shown below. The resultant error is calculated by the square root of the errors in the X-axis and y-axis directions whose equation is $error_{total} = \sqrt{error_x^2 + error_y^2}$.

The calculated trajectory in the x-axis over time can be seen from Fig.3.4(a), and the y-axis is from Fig.3.4(b). The experiment results follow the shape of the defined trajectory. In these two Figures, the amplitudes cannot be distinguished clearly.

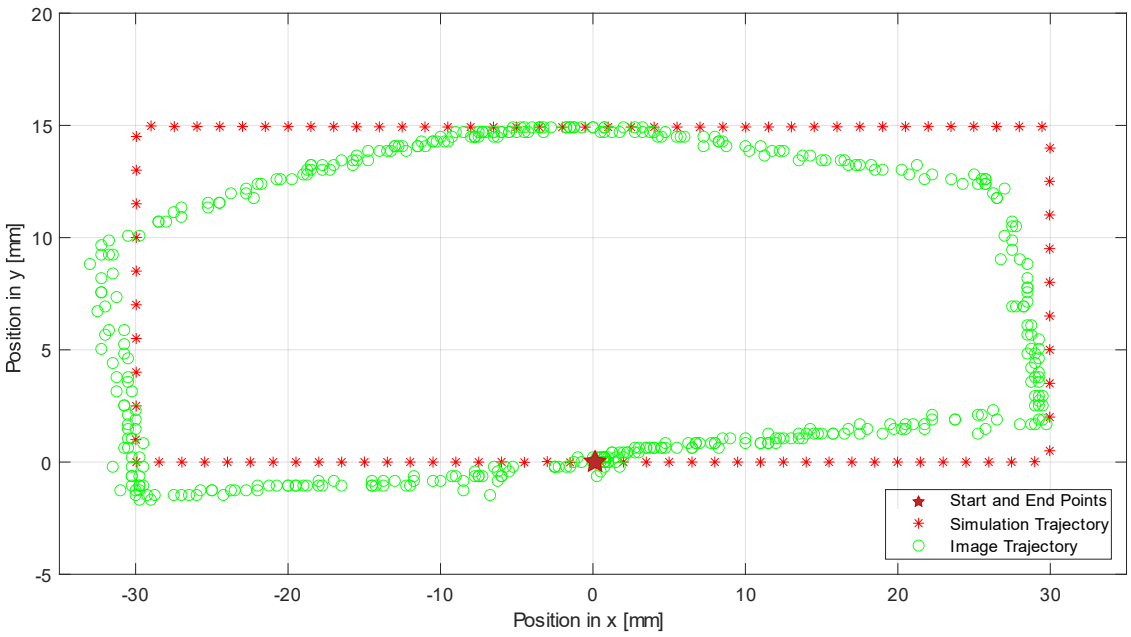
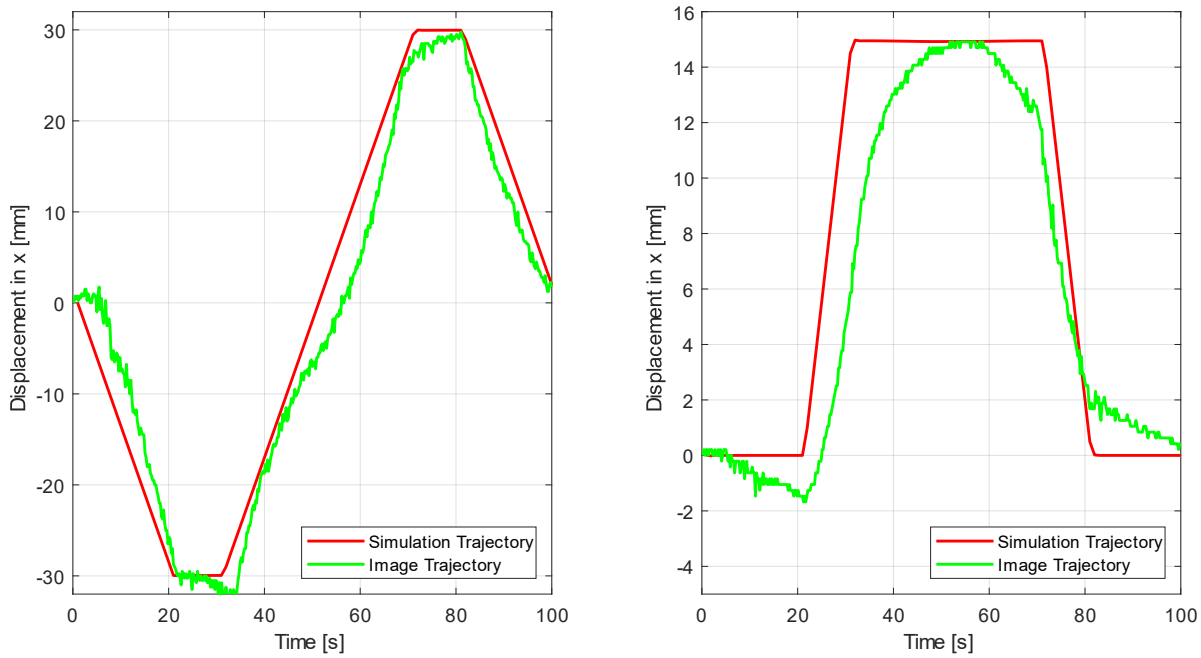


Fig.3.5 Trajectories form simulation and camera based on differential Jacobian matrix

In Fig.3.4(c), the displacement error can be demonstrated clearly. The total error of the x-axis and y-axis is less than 0.1. It is noticed that the error keeps increasing. Generally speaking, the algorithm only generates a small error that shows the damped least square can solve the inverse kinematics problem quiet well.

Assume the trajectory is in the real world and input the target points on the rectangular to the microDART robot one by one, so the expected trajectory of the robot should be rectangular. In other words, this experiment is to test the combination of the constant curvature and damped the least square with the differential Jacobian matrix. In section 4.2, the catheter tip tracking method was introduced. In this section and the next two sections, the camera will keep recognizing the catheter tip position and recoding it. Convert the image pixel position to real-world position and draw the trajectory as Fig.3.5.

The microDART moved a rectangular liked trajectory, as Fig.3.5 shown. At the start, the tip was at (0,0) and moved to the lower-left direction, but the trajectory is always under the simulation results. Then followed the command to increase the Y-axis value, but it cannot reach the top right corner. After that, the catheter moved to the right and can follow the simulation result in a range of -10mm to 5mm. However, it still cannot reach another highest position on



(a) Displacement in x-axis (b) Displacement in y-axis

Fig.3.6 Comparison between simulation result and experiment result

the right side. To show the algorithm more clearly, the differences in X-ais and Y-axis are shown as Fig.3.6.

In Fig.3.6 (a), the overall error is not very big, and the catheter in the image followed the expected simulation result most of the time. Before the 20s, the catheter keeps moving ahead of the expected result, which shown as the green line is always higher than the red one, and the error creeps down until the catheter stop moving in X-axis at 20s. From the 20s-40s catheter start to move in Y-axis, and the error in X-axis climb up, but after the 30s, when the X-axis direction starts to move again, the error drop to its lowest point at 40s and start to edge up until next time movement in X-axis.

In Fig.3.6 (b), the image trajectory whose color is green is always lower than the expected position. Especially for 10-40s and 65-70s, the difference is about more than 2mm. However, the shape and the movement trend are the same as the simulation result. In summary, with the distance from the original start point, increasing the error keep at a high level. However, for the near start point position like 0-10s, 50-60s, and last 10s, both X-axis and Y-axis differences are not very high.

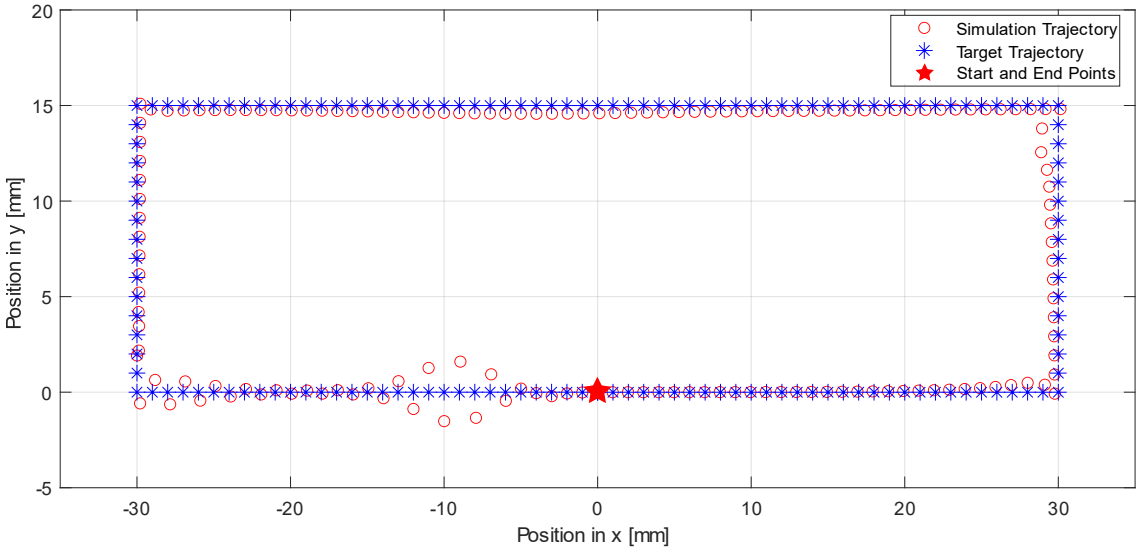


Fig.3.7 Target and simulated trajectory by estimated Jacobian matrix method

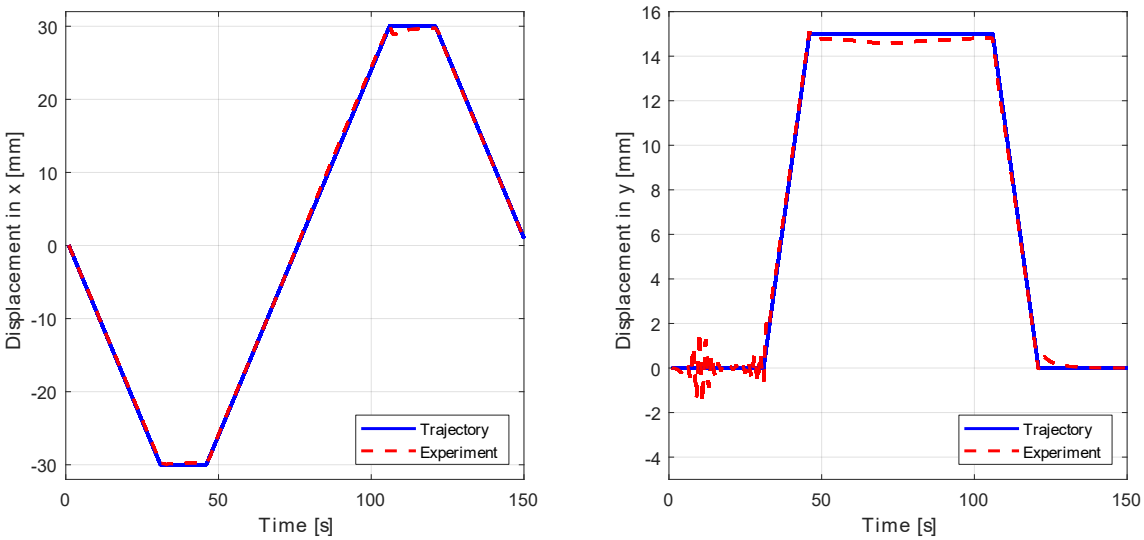
3.3 Estimated Jacobian Matrix

Use the same rectangular trajectory design, as shown in Section 2.4, to test the estimated Jacobian matrix algorithm.

1. Start in (0,0), move alone x-axis from 0mm to -30mm.
2. Move alone y-axis from 0mm to 15mm reach the point (-30,15).
3. Move alone x-axis from -30m to 30mm get point (30,15).
4. Move alone y-axis from 15mm to 0mm.
5. Move alone x-axis from 30mm to 0mm back to the original point (0,0).

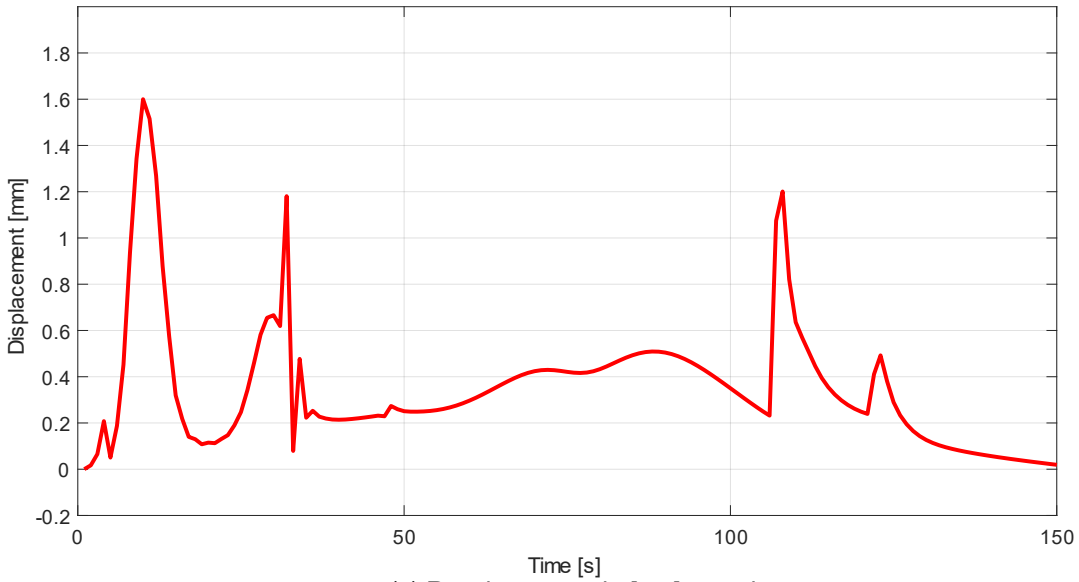
The target trajectory and end-effector simulated trajectory of the robot is shown as Fig.3.7

Same as Section 2.3, we use the comparison of the expected and experimental position in x-displacement and y-displacement and total error $\sqrt{x^2 + y^2}$ to evaluate the performance of the estimated Jacobian matrix inverse kinematics algorithm.



(a) Displacement in x over time

(b) Displacement in y over time



(c) Resultant error in [x,y] over time

Fig.3.8 Inverse kinematics simulation based on estimated Jacobian matrix

Fig.3.8 (a) demonstrates the displacement in X-axis, and there is no apparent difference between simulation result and target except the time from 105s to 110s, where is a 1mm error. However, for Y-axis displacement, there are two times oscillations at the beginning of the movement in the 0-30s. After the 30s, the error keeps a low level, which is less than 1mm. Both X-axis and Y-axis show a higher difference with the expected one while the other direction is moving instead of itself.

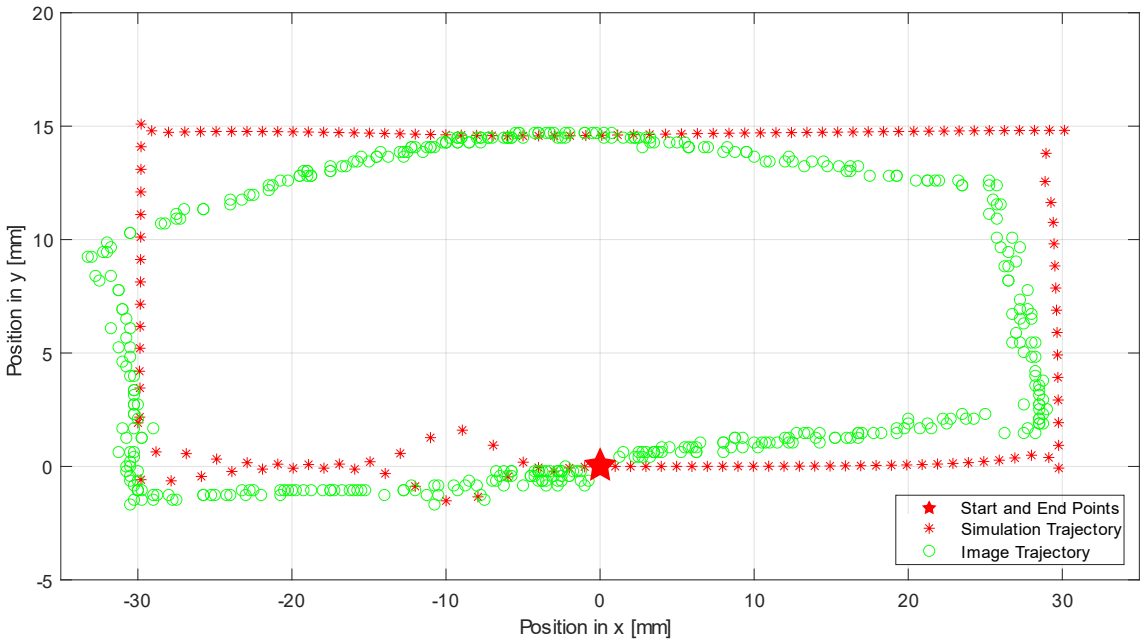


Fig.3.9 Trajectories form simulation and camera based on estimated Jacobian matrix

Fig.3.8 (c) shows the tendency of the difference between the expected target position and the inverse kinematics simulation results. At the start 50s, there is two maximum local error due to the oscillation in Y-axis that is 1.6mm and 1.2mm. After that, the error is keeping lower than 0.5mm until another peek in 110s. It can be seen from Fig.3.8 (a) and Fig.3.8 (b) there are both an obvious error in X-axis and Y-axis. The highest difference is 1.6mm, with a fluctuate around 0.4mm most of the time, which shows the estimated Jacobian matrix method performs well enough in simulation.

As section 3.2, there is also a real-world test for this algorithm. The camera also keeps recording the catheter tip position by image recognition in Section 2.2.

The trajectory guided by the estimated Jacobian matrix is almost the same as the last section, one whose Jacobian matrix is from the differential Jacobian method. The trajectory also shows a rectangular liked shape, and it also cannot reach the top left, top right, and bottom right point in the simulation result. Especially for the top left corner, the displacement error is about 5mm.

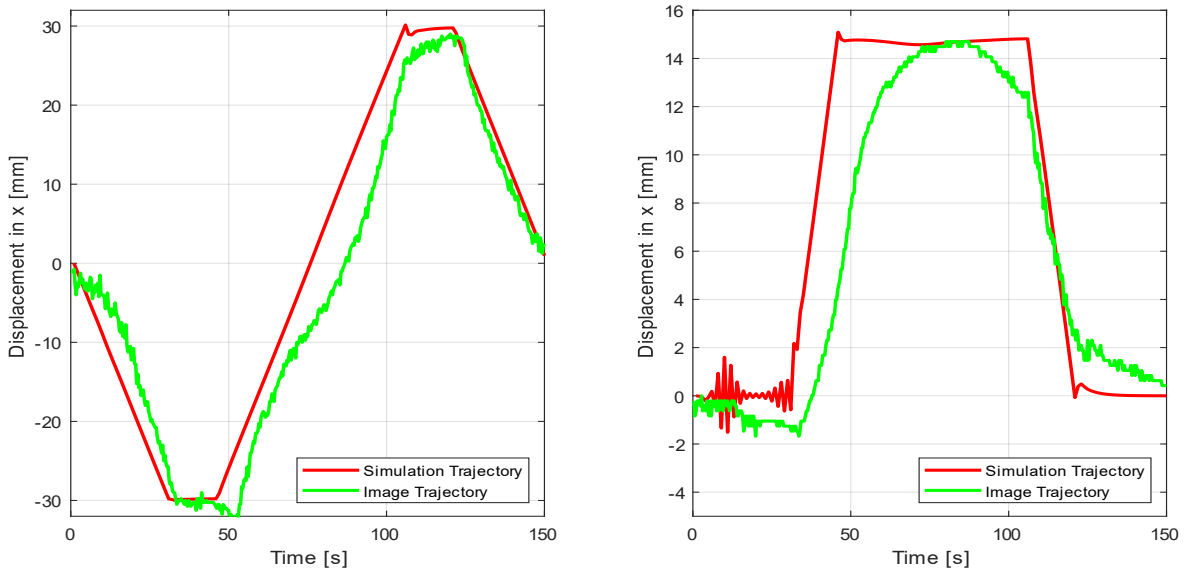


Fig.3.10 Comparison between simulation result and experiment result

As Fig.3.10 (a) shown the displacement in X-axis, for the first 30s, the catheter in the image leads the simulation trajectory about 1mm. For the 30s-50s, the movement is accurate as of the expected one. Nevertheless, after the 50s, the error starts to rise again, and the image position lag to the simulation one until it finishes the movement. It is noticed that in the last round of the X-axis movement in which the duration of 120s-150s, the error decrease to a low level again. In Fig.3.10 (b), the displacement in Y-axis, the image trajectory, also shows oscillation in the first 30s, which fits the simulation results. Moreover, before the 110s, the real-world trajectory always lags the simulation results.

3.4 Visual Servoing

In this section, all the algorithms introduced in previous sections will be combined for testing the visual servoing feedback control. The two different methods Differential Jacobian Matrix and Estimated Jacobian Matrix for updating the Jacobian matrix in damped least square inverse kinematics are both discussed as a comparison. As section 2.2 said, the test for this project is a blue target on a white paper. To see how accurate the method is, the blue target is drawn on the 3mm square top right corner so we can have a scaled reference system. The task aims are to try to let the image feedback guide the robot start in the bottom left corner to move to the blue target on the top right corner of the square.

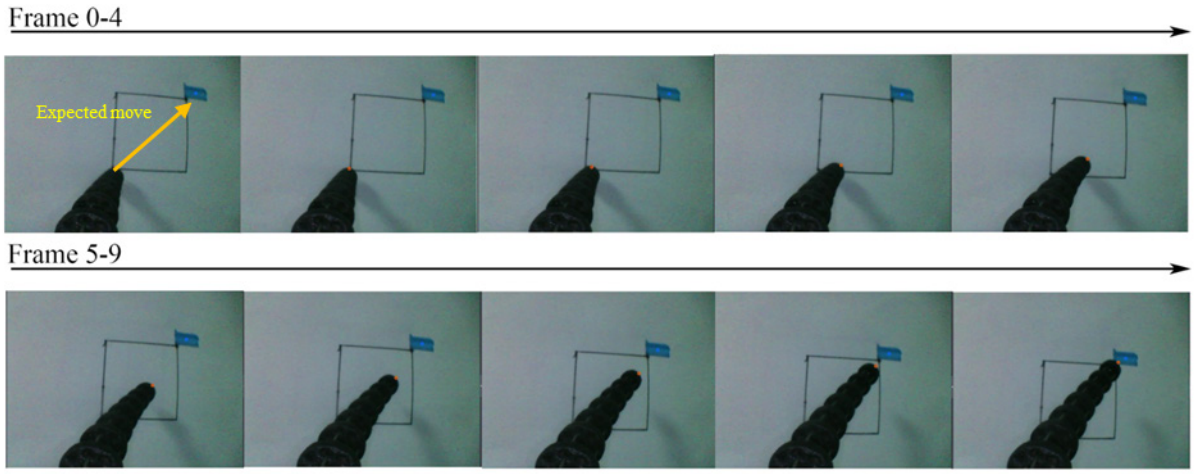


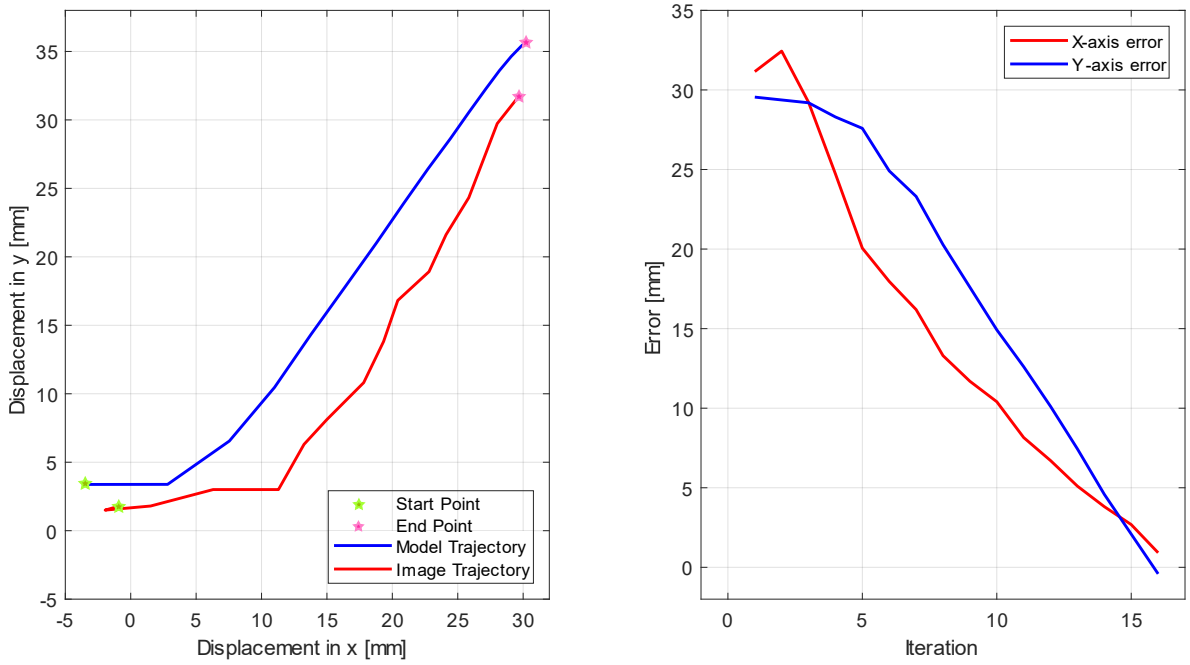
Fig.3.11 Video frames during visual servoing based on differential Jacobian matrix

3.4.1 Damped Least Squares with Differential Jacobian Matrix

The video frame after recognition is recorded by me, as Fig.3.11(a) shown the catheter does not need any time to converge the movement direction. It directly moves to the target step by step until it reaches the target and stops. There are ten frames from the video stream demonstrated as Fig.3.11.

Fig.3.11 shows that when the catheter arrives at the target, there is still a small difference. The reason is that to avoid catheter obscuring blue target position in the camera first view. A threshold is set for the error, which means when the distance between the catheter tip and the target is less than 2mm, the movement will stop.

To get more detailed information, the catheter tip position is kept tracking and saved. Fig.3.12 (a) compares the model trajectory and the real-time catheter trajectory on the image. The constant curvature model calculates the blue line model trajectory with the angle value from damped least square inverse kinematics. The red line is the automatic recognition result from the video frame. It is noticed that the model trajectory always is higher than the image one. Moreover, in the beginning, the red line moved away from the target about 2mm and then started to converge to the right result. Overall, the catheter tip in the image moved from (-0.86,-1.86)mm to (29.8,31.83)mm, which is from the bottom left corner to the top right.



(a) Trajectory in camera and trajectory from model (b) Error in X-axis and Y-axis

Fig.3.12 Trajectory based on differential Jacobian matrix tracked by camera

In Fig.3.12 (b), the displacement error in X-axis and Y-axis are demonstrated separately. The error is an image error, which means the catheter position and target position are both extracted from image information by Section 4.5 algorithm. It is easy to notice that the catheter moves to a negative Y-axis in the first-time movement that is the same as a result shown in Fig.(a). In the beginning, the converging speed on X-axis is faster than Y-axis. Then after 13th iteration, the Y-axis converge speed surpass X-axis' one. Both X-axis and Y-axis decrease from about 30mm to around 0mm after 16 times iterations. The errors after servoing control are 1.9mm in X-axis and 0.8 on Y-axis.

There are two more visual servoing tasks tested, which are 20mm square and 30mm square. Both of the tasks is to let the catheter move from the bottom left corner of the square to the top right corner. As can be seen from Fig.3.13, all three trajectories can coverage under the visual servoing algorithm. However, the trajectory still shows some oscillation during the movement which caused the trajectory was not the shortest path from the start point to the end point. As table 3.1 shown, each target is tested ten times and the iteration times are both from 13 to 22 that means iteration speed is not related to the target distance.

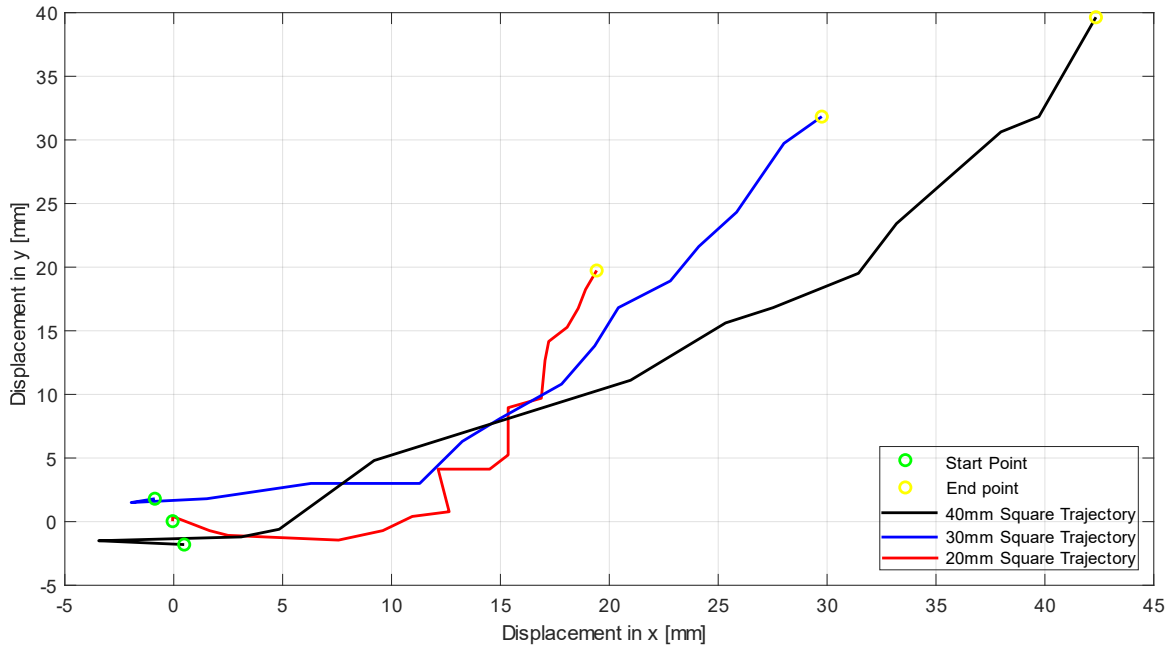


Fig.3.13 Trajectories of three different distance targets by differential Jacobian matrix

	1	2	3	4	5	6	7	8	9	10
20mm	17	21	21	13	18	22	15	19	19	17
30mm	21	13	16	14	20	22	19	16	17	15
40mm	14	17	17	20	21	13	15	17	22	16

Table 3.1 Iteration times of three different distance targets

3.4.2 Damped Least Squares with Estimated Jacobian Matrix

Same as Section 3.4.1, the video frame is recorded and demonstrated what is the procedure during the visual servoing control. This time the catheter took more time to reach the target. Fig.3.13 shows 15 frames of recognition result video stream.

At beginning, the tip of the catheter moved to the lower-left direction, which is opposite from the target. Then it changed the direction to the right of the X-axis with a small bias on the position of Y-axis. The following movement is to the bottom right, followed by a direction changing to higher left. After almost reaching the top left corner of the square, it turns to converge to the target and move to position X-axis direction, then the catheter reached the target area and stopped moving.

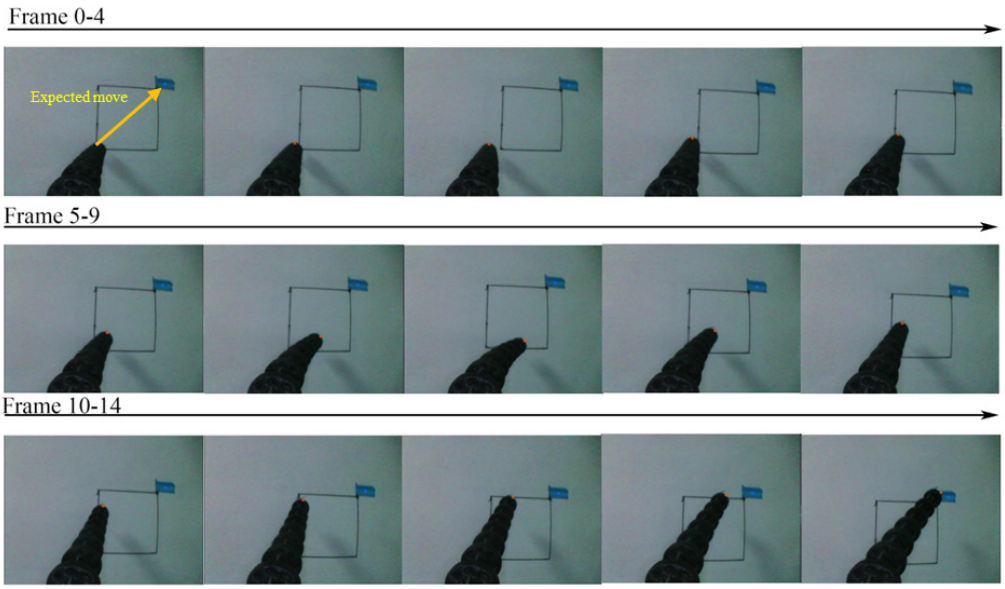


Fig.3.14 Video frames during visual servoing based on estimated Jacobian matrix

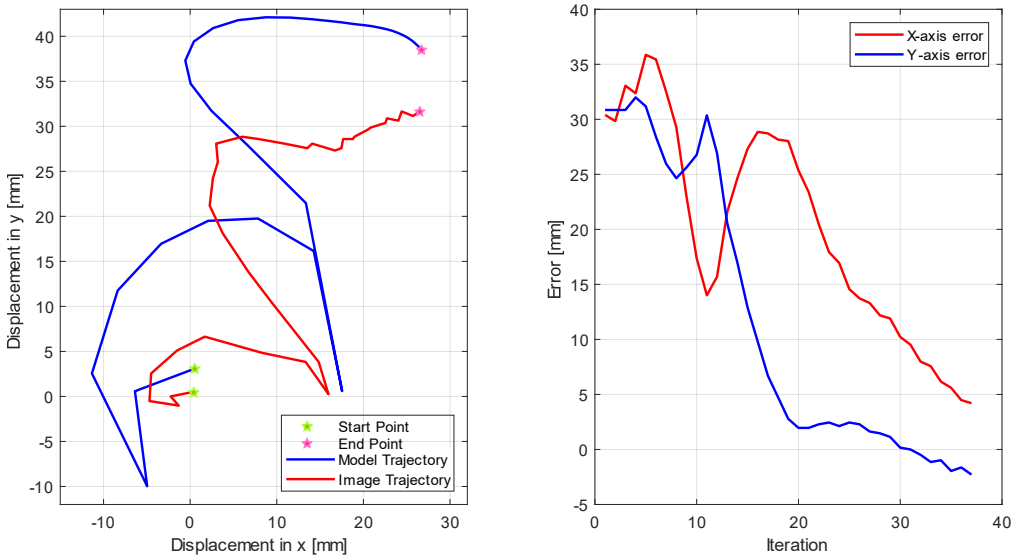


Fig.3.15 Trajectory based on estimated Jacobian matrix tracked by camera

Fig.3.14 (a) shows the constant curvature trajectory during the movement (blue line) and trajectory extracted by imaging recognition (red line). The two trajectories are almost the same, which means the movement matched the constant curvature kinematics model. There are five steps before the catheter reaches the blue target: 1. Move away from the target to lower left get (-4,0). 2. Change to the higher right direction. 3. After getting the point (1,6), change direction to move to (15,0) 4. Start to increase height to top left corner (3,28). 5. Move directly to target in the position X-axis direction and stopped in (27,31).

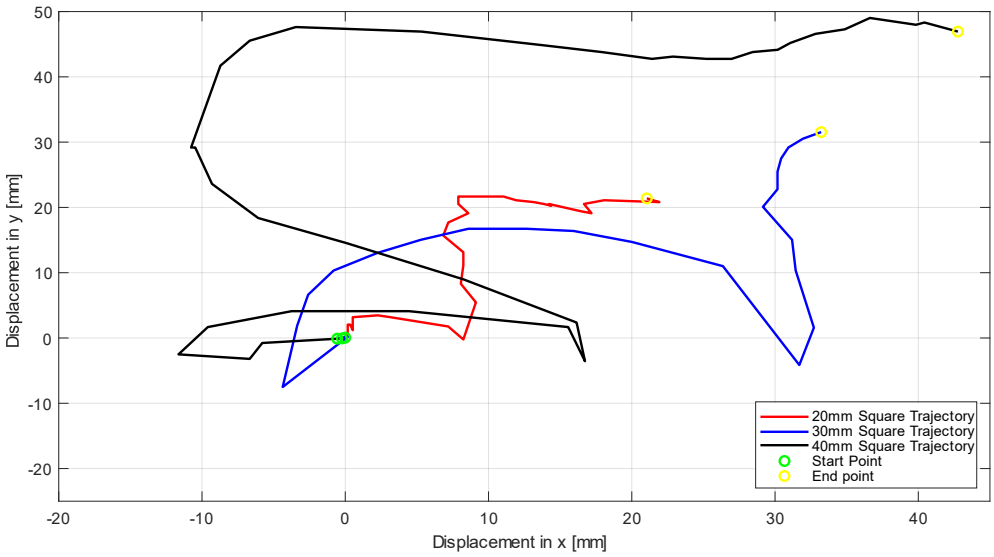


Fig.3.16 Trajectories of three different distance targets by the estimated Jacobian matrix

As Fig.3.14 (b) shown, the tendency of error in the X-axis and Y-axis looked the same. Both of them reduce at the beginning and raise, followed by a continuous dropping. The error in X-axis started in 30mm, which is the original point and soar up to 36mm, which corresponding to the behaviors of moving away to the lower left in Fig.3.14 (a). Then the error decrease to 15mm, followed by another time increase to 28mm. After that, the error keeps slumping to 4.7mm and stops. For the Y-axis error, it converges after the 11^{th} iteration. After a 2mm increase to 32mm, the error drop to 25mm and crept up to 30mm in 11^{th} iteration. Then the error keeps plummeting to reach -1mm in 38^{th} iteration and stop moving. The final errors after visual servoing guided movement are 4.7mm in X-axis and 1mm in Y-axis.

Same as the last section of the differential Jacobian matrix method, there also two more different distance targets were tested. It is easy to see from the Fig.3.16 the estimated Jacobian converged slower than the differential one. The trajectory shows that the catheter moved away from the target at the beginning which will cause an unstable movement. The converge time of estimated Jacobian is much more than the differential method which is from 27 to 41, as Table 3.2 shown. However, because the catheter sometime will move away from the target, which would cause the movement of the camera. And the target point would go out of camera view. As a result, during the test of visual servoing by the estimated Jacobian matrix, there are several times failed to converge to the target.

Index / Distance	1	2	3	4	5	6	7	8	9	10
20mm	39	37	40	38	34	27	X	40	38	31
30mm	36	40	27	X	41	35	27	31	X	29
40mm	X	33	32	X	38	31	38	41	41	35
X means the test did not converge										

Table 3.2 Iteration times of three different distance targets

3.4.3 Contact Location Estimation

During the visual servoing control, a contact force location estimation was inspired. From formula (13) we can get the equation of Jacobian matrix in X-Y plane as :

$$J(\theta)=\begin{bmatrix}\frac{\partial P_x}{\partial\alpha} & \frac{\partial P_x}{\partial\beta}\\ \frac{\partial P_y}{\partial\alpha} & \frac{\partial P_y}{\partial\beta}\end{bmatrix}=l\begin{bmatrix}\frac{1}{\beta}sin\alpha(cos\beta-1) & \frac{1}{\beta}cos\alpha sin\beta+\frac{1}{\beta^2}cos\alpha(cos\beta-1)\\ -\frac{1}{\beta}cos\alpha(cos\beta-1) & \frac{1}{\beta^2}sin\alpha(cos\beta-1)+\frac{1}{\beta}sin\alpha sin\beta\end{bmatrix} \quad (18)$$

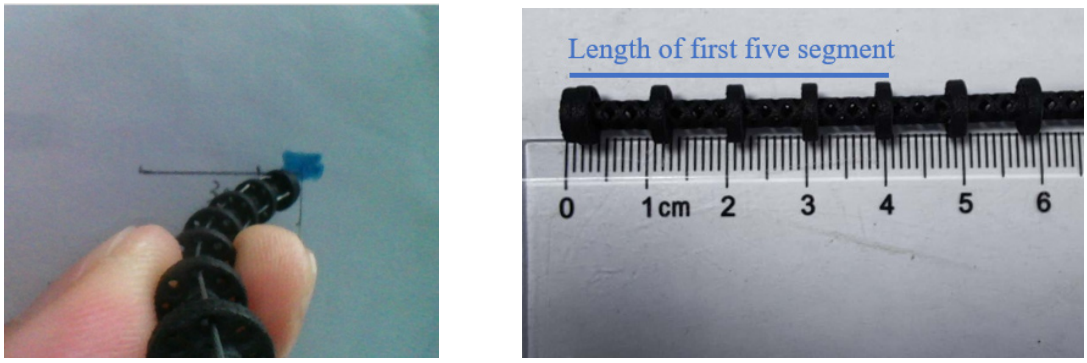
It is easy to calculate the determinant of the matrix on the right side of the formula (18).

$$det\left(\frac{1}{l}J(\theta)\right)=\frac{(cos\beta-1)(cos\beta+\beta sin\beta-1)}{\beta^3} \quad (19)$$

In the procedure of visual servoing, the catheter must have a bending angle, and it is also impossible to be bent over $90°$ so the range of bending angle will be $\beta\in(-\frac{\pi}{2},0)\cup(0,\frac{\pi}{2})$. Because the differential of $(cos\beta+\beta sin\beta-1)$ is $\delta\times cos\beta$ that is still larger than zero in the range of bending angle. It is easy to get the value of $(cos\beta+\beta sin\beta-1)$ is in the range of $\left[-1-\frac{\pi}{2},0\right)\cup\left(0,\frac{\pi}{2}-1\right)$, which means the matrix is full-ran. As a result, the formula (18) can be rewritten to calculate the length of the working part of the catheter as formula (20).

$$l=\begin{bmatrix}\frac{\partial P_x}{\partial\alpha} & \frac{\partial P_x}{\partial\beta}\\ \frac{\partial P_y}{\partial\alpha} & \frac{\partial P_y}{\partial\beta}\end{bmatrix}\begin{bmatrix}\frac{1}{\beta}sin\alpha(cos\beta-1) & \frac{1}{\beta}cos\alpha sin\beta+\frac{1}{\beta^2}cos\alpha(cos\beta-1)\\ -\frac{1}{\beta}cos\alpha(cos\beta-1) & \frac{1}{\beta^2}sin\alpha(cos\beta-1)+\frac{1}{\beta}sin\alpha sin\beta\end{bmatrix}^{-1}$$

$$\Rightarrow\ l=\begin{bmatrix}\frac{\partial P_x}{\partial\alpha} & \frac{\partial P_x}{\partial\beta}\\ \frac{\partial P_y}{\partial\alpha} & \frac{\partial P_y}{\partial\beta}\end{bmatrix}\begin{bmatrix}\frac{-\beta sin\alpha}{1-cos\beta} & \frac{-\beta cos\alpha}{1-cos\beta}\\ \frac{\beta^2 cos\alpha}{cos\beta+\beta sin\beta-1} & \frac{\beta^2 sin\alpha}{cos\beta+\beta sin\beta-1}\end{bmatrix} \quad (20)$$



(a) The catheter with an obstruction (b) Length of the catheter's first five segments

Fig.3.17 Catheter with external contact

The camera can extract the position of target and end-effector, and the operation time can be recorded by PC. Hence the end-effector speed $\partial P_x, \partial P_y$ can be calculated. At the same time, the joint information (rotation angle α and bending angle β) is an output of the damped least square inverse kinematics in visual servoing procedure, which means the joint speed $\partial\alpha, \partial\beta$ can also be calculated based on the time recorded by PC. Hence, the formula (20) can calculate the working length of the catheter.

A simplified experiment without the time recorded is tested in this section. As Fig.3.17 (a) shows, give an obstruction (the finger in this example) to block the fifth segment of the catheter and place a target on the 3mm right-side of the catheter. Then use visual servoing control with the estimated Jacobian matrix algorithm to guide the catheter reached the target. After the catheter gets the target, the current bending angle and rotation angle can be calculated by damped least square which are $\alpha = 0.0002316\,rad, \beta = 2.39862673\,rad$ in this example. Through the camera, we can abstract the distance between the target and catheter tip so the expected position after servoing control can be easily get as (3,0)mm in this example. Form formula (1) the end-effector position in X-axis is $\frac{l}{\beta}cos\alpha(1 - cos\beta)$ where l is the length of the catheter. The equation can be rewritten as:

$$l = \frac{P_x \times \beta}{cos(\alpha) \times (1 - co\,s(\beta))} = \frac{3 \times 2.39862673}{cos(0.0002316) \times (1 - co\,s(2.39862673))} = 4.1mm$$

Hence the length of the catheter with the contact force is 41mm. It can be seen from Fig.3.17 (b), the 41mm catheter corresponding to the original catheter with an obstruction in the fifth segment of it.

Chapter IV

Discussion

The demonstrated works aims to design, implement, and model a stepper motor driven soft endoscopic robot that we called microDART and provide a solution for visual servoing feedback control. Two different catheter designs have been developed, including disk cutting and helix design. The helix design catheter was selected as the final structure to provide the robot bending ability. A stepped motor actuated control system with Wi-Fi wireless communication was introduced. A visual marker tracking program also has been developed and implemented to provide the target and catheter position for visual servoing. The overall system is integrated with an Arduino based microcontroller Teensy 4.1 with a Wi-Fi modular Node-MCU ESP-6266 and a PC using program architecture developed by Python3. To generate accurate motion of the microDART catheter, a kinematics model can map the relationship among actuator space, joint space, and end-effector working space was introduced based on constant curvature. To realize the visual servoing control, inverse kinematics can converge by an error in the image that was developed based on damped least squares. Due to the requirement of updating the Jacobian matrix in the inverse kinematics model, two different methods were discussed, which are differential Jacobian matrix and estimated Jacobian matrix. The physical robot has tested both algorithms for one generated rectangular trajectory and one visual target each.

4.1 System design

4.1.1 Flexible Manipulator Design

As it is shown, the segmented soft robot design shows excellent performance in bending ability. Both of the disks cut design and helix design in this project can be easy to drive by a steel cable. The catheters are 3D printed with high-performance nylon by a Chinese company, which makes it a highly cost-efficient product whose 3D printed price is 0.3 GBP for each. For the driven

cable, there are three different kinds of materials tested, which are nylon cable, steel cable, and steel cable with a plastic shell. After testing, the steel cable with shell was selected because of the excellent bending performance and ability to shape memory.

The disk cut design is easy to be deformed into be an irregular shape, which will be hard to find the control model. The helix design became the final choice served for this project. When applied a strain by driven cable to the helix design catheter, the catheter showed a regular curve shape. However, the longtime use of nylon based catheter will also cause deformation, which could cause the inaccuracy problem in controlling..

4.1.2 Actuation

The cable-driven flexible robot system shows good quality in applications. The accuracy of stepper motors is enough for this project. Because the stepper motor is a cycle moving actuator, the movement of the catheter is non-linear. Especially for the base platform movement, it is hard to find a suitable velocity and acceleration. The delay is quite long, with a small velocity and acceleration, but the platform will shake a lot if there is a high value set to velocity and acceleration.

4.1.3 Communication

The UDP shows a high communication speed, which is much higher than the serial port. Mainly it is a wireless communication based on ethernet. Because the protocol does not consider the data, there is almost no latency during the data receiving. This design shows that the wireless control for microDART is possible. Hence, in the future, replacing the Wi-Fi modular by a 5G chip can achieve the no distance limitation remote control. Nevertheless, because after the Node-MCU received data, the connection between it and Teensy is still a serial port, the speed cannot reach its highest limitation. Directly connecting the ESP-8266 modular to Teensy with SPI Bus should be a right solution.

4.2 Image Sensing

The image recognition algorithm shows an excellent performance, which can keep tracking the position of catheter and target. However, it is highly influenced by ambient lighting conditions. Once the system is set up, the frequencies can reach 39Hz with a 60 FPS raw video stream input. The area and rotated rectangular detection can correctly extract the target and catheter tip position. For the given applications, the catheter position extracted from the image almost matched the inverse kinematics results.

Because the camera is placed on the multi-channel tube to monitor the whole catheter change, however, the system is a monocular system that cannot measure the depth of the environment. In this project, the catheter keeps contacting the target plane, which means the camera can only consider the X-Y plane movement. Another problem is that when the catheter bends to Y-axis negative direction, the catheter tip will be blocked by the catheter body. In this situation, the detection algorithm cannot track the actual catheter tip position.

4.3 Forward Kinematics

The constant curvature forward kinematics provides a highly efficient solution to calculate the position of catheter based on the driven cable difference. It gives a good mapping relationship for the cable-driven soft robot among end-effector working space, joint space, and driven cable space. The movement of the catheter under the guidance of the forward kinematics model is very close to the expected target, as it can be seen in section 5.1. Because the camera cannot get the information on Z-axis, there is compensation in the kinematics model to keep catheter contacting with the target plane. Occasionally the control will show non-linear behavior. It is considered as the initial position of control cable platform, which will cause the strain applied to each direction of the catheter is different. From the transformation matrix formula(1), it is easy to see the end-effector position inverse ratio to the catheter length. It is also noticed that the distance tracked by the camera is always longer than the model result, which means the real bent catheter length is shorter than the calculated one.

4.4 Inverse Kinematics

The damped least square inverse kinematics shows excellent performance in MATLAB simulation. For the differential Jacobian matrix method, the maximum difference between assigned and simulation end-effector position is less than 0.1mm. The error in the physical experiment with the camera shows results with a slightly higher error than the simulation one. The maximum deviation between the expected rectangular and experimental trajectory is about 3mm in X-direction and 2.4mm in Y-direction. For the estimated Jacobian matrix method, the maximum error in simulation is 1.6 mm due to the oscillation in Y-axis at the beginning of the movement. The maximum differences between simulation in MATLAB and image recognized position are 5mm in X-axis and 3.5 mm in Y-direction. As section 6.3 described, the error could form the initial driven cable position. Another reason could be the camera position change. Because of the Z-axis compensation, the camera position will be closer to the target plane when the catheter moves far away from the original position. As a result, the distance between the start point and catheter in the image will be smaller.

4.5 Visual Servoing

Both of the differential Jacobian matrix and Estimated Jacobian matrix methods provided satisfactory results in the visual servoing task. Especially, the differential Jacobian matrix method can directly guide the catheter to move to the target. As section 5.4.1 which shows the error between the target and catheter tip position on image keep dropping until less than 2mm. Furthermore, the speed of converging to the target is also noticeable. The catheter only took 16-time iterations and got the target position. For the estimated Jacobian matrix, it needs time to converge then find the right direction to the target. At the beginning of servoing control, it moved in the opposite direction from the target, but it still found the target position after 38 times iterations. Even the estimated Jacobian matrix method performs worse than the differential one. However, it is undeniable that it is a good algorithm because the bending ability of the catheter is non-linear, as mentioned in previous sections. The estimated Jacobian matrix with damped least squares inverse kinematics model provides us a solution to do visual

servoing without any pre-knowledge about the kinematics. Moreover, because there is no requirement of kinematics the robot with a unknown external force still can be controlled by the visual servoing algorithm. Then the contact location estimation was discussed and shown great potential to detect the external force of the flexible robot manipulator.

4.6 Applicability and further development

In this report, the advanced maneuverability based on image feedback of the catheter tip in Micro Dexterous Assistive Robotic Therapy (MicroDART) can follow given target locations. In the practical surgery, given target marks will be made by the surgeon's decision during an MIS. According to an electromagnetic-guided endoscopic intracranial surgery for children, the navigation position accuracy is 2-9mm and optoelectronic navigation ranges between 0.71–3.51mm [46] and 0.7–4.4 mm [47]. All three systems only can give planning trajectory, which still needs manual control to the endoscope. The measured accuracies of the visual servoing algorithm in this project are 0.8-1.9mm for the differential Jacobian matrix method and 1.7-4.7mm for the estimated Jacobian matrix. For the external contact location estimation, it can give a rough position of the obstacle related to the robot manipulator. In the medical application, it provides a possible solution for avoiding contact with patient body. Thus, this project improved the traditional endoscopic surgery by providing actuators driven flexible endoscopic robot with visual feedback servoing automatic control.

However, the measured position accuracy is excluding depth with the situation, which assumed that the catheter tip kept contacting with target plane by a kinematics Z-axis compensation. Considering the working environment size limitation, there cannot be a binocular vision on the catheter. Besides, the Haptic feedback is another solution to guarantee the contact of the catheter tip with the target plane and can guarantee the accuracy of force applied to the patient. The haptic feedback is already available in King's College London Hammer Lab.

Therefore, in the future, the microDART system with dual feedback control (visual servoing and haptic feedback control) can give a more accurate endoscopic dissection surgery. It is well known that a surgeon needs many years of training, so the automatic dissection endoscopic robot system would address the shortage of doctors and reduce the difficulty of endoscopic

surgery. Moreover, the Wi-Fi modular in this project can be replaced by a 5G mobile controller. After the implementation of the image and haptic data compression techniques can realize low latency data transmission so that a hyper remote surgical system based on 5G can be developed. This system can be applied in some impoverished areas where medical resources are sparse, someplace without specialized surgeons like an ocean liner or the surgery for infectious disease. It will make no difference between traditional surgery and this kind of hyper remote surgery for the doctor because of the combination of imaging and haptic sensing.

4.7 Conclusion

The algorithm shows great potential of solving the cable-driven flexible robot visual servoing problem. The project successfully provides a bendable catheter on a flexible robot and a highly efficient low-cost actuation system based on stepper motors and steel cable. Furthermore, the wireless communication system has been developed to provide a Wi-Fi-based UDP wireless control. A kinematics model can map the working space, joint space, and driven space was found to give an accuracy control, which assumes each joint on the catheter has the same curvature. To realize the visual servoing control, damped least squares inverse kinematics was used to calculate expected bending angle and rotation angle from the Jacobian matrix and the error between target and catheter position. There are two methods, which are a differential Jacobian matrix with 0.1 mm error and the estimated Jacobian matrix with 1.6mm error in simulation. The system has also been tested physical robot visual servoing task, which shows good results with error in 0.8-1.9mm for differential Jacobian matrix method and 1.7-4.7mm for estimated Jacobian matrix. The contact location estimation can also give a correct position of the obstacle. This project shows a high applicability of the endoscopic surgery which automatically operated by a flexible robot under visual information guidance.

Reference

[1] Pearl JP, Ponsky JL. Natural orifice translumenal endoscopic surgery: a critical review. J Gastrointest Surg. 2008;12: 1293–300.

[2] Trivedi, D., Rahn, C. D., Kier,W. M., andWalker, I. D. (2008). Soft robotics: Biological inspiration, state of the art, and future research. Applied Bionics and Biomechanics,5(3):99–117.

[3] WALKER I D, CARRERAS C, MCDONNELL R, et al. Extension versus bending for continuum robots [J]. International Journal of Advanced Robotic Systems, 2006, 3(2): 171 - 178.

[4] GRAVAGNE I A, RAHN C D, WALKER I D. Large deflection dynamics and control for planar continuum robots [J]. IEEE/ASME Transactions on Mechatronics, 2003, 8(2): 299 - 307.

[5] WALKER I D, HANNAN M W. A novel ‘elephant’s trunk’ robot [C]//Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Sept. 19 - 23, 1999, Atlanta, USA. IEEE, 1999: 410 – 415.

[6] JONES B A, MCMAHAN W, WALKER I D. Design and analysis of a novel pneumatic manipulator [c]//Proceedings of 3rd IFAC Symposium on Mechatronic Systems, Sept. 6 - 8, 2004, Sydney, Australia. 2004:745 – 750.

[7] JONES B A, MCMAHAN W, WALKER I D. Design and analysis of a novel pneumatic manipulator [c]//Proceedings of 3rd IFAC Symposium on Mechatronic Systems, Sept. 6 - 8, 2004, Sydney, Australia. 2004: 745 - 750.

[8] MCMAHAN W, CHITRAKARAN V, CSENCSITS M, et al. Field trials and testing of the OctArm continuum manipulator [C]//Proceedings of IEEE International Conference on Robotics and Automation, May 15 - 19, 2006, Orlando, Florida. IEEE, 2006: 2336 - 2341.

[9] CHEN G, PHAM M T, REDARCE T, et al. Development and kinematic analysis of a silicone rubber bending tip for colonoscopy[C]//Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System, Oct. 9 - 15, 2006, Beijing, China. IEEE/RSJ, 2006: 168 – 173.

[10] CHEN G, PHAM M T, REDARCE T. Sensor -based guidance control of a continuum robot for a semiautonomous colonoscopy [J]. Robotics and Autonomous System, 2009(57): 712 – 722.

[11] Kwoh YS, Hou J, Jonckheere EA, Hayati S. A robot with improved absolute positioning accuracy for CT guided stereotactic brain surgery. IEEE Trans Biomed Eng 1988 Feb;35(2):153–60.

[12] Kalloo AN, Singh VK, Jagannath SB, Niiyama H, Hill SL, Vaugh CA, et al. Flexible transgastric peritoneoscopy: a novel approach to diagnostic and therapeutic interventions in the peritoneal cavity. Gastrointest Endosc 2004;60(1):114e7.

[13] Phee SJ, Low SC, Huynh VA, Kencana ZL, Yang SK. Master and slave translumenal endoscopic robot (MASTER) for natural orifice translumenal endo-scopic surgery (NOTES). Conf Proc IEEE Eng Med Biol Soc 2009;4:1192e5.
[14] Phee SJ, Reddy N, Chiu PW, Rebala P, Rao GV, Wang Z, Sun Z, Wong JY, Ho KY. Robot-assisted endoscopic submucosal dissection is effective in treating patients with early-stage gastric neoplasia. Clin Gastroenterol Hepatol 2012; 10: 1117-1121.
[15] D. J. Abbott, C. Becke, R. I. Rothstein, W. J. Peine, "Design of an endoluminal NOTES robotic system", Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp. 410-416, 2007.
[16] Rothstein RI, Ailinger RA, Peine W. Computer-assisted endoscopic robot system for advanced therapeutic procedures. Gastrointest Endosc 2004;59(5):P113.
[17] Webster, R. J. & Jones, B. A. Design and kinematic modeling of constant curvature continuum robots: a review. Int. J. Robot. Res. 29, 1661–1683 (2010).
[18] Rus, D. and Tolley, M. T. (2015). Design, fabrication and control of soft robots. Nature, 521(7553):467–475.
[19] JONES B A, WALKER I D. Kinematics for multisection continuum robots [J]. IEEE Transactions on Robotics, 2006, 22(1): 43 - 55.
[20] JONES B A, WALKER I D. Practical kinematics for real-time implementation of continuum robots [J]. IEEE Transactions on Robotics, 2006, 22(6): 1087 - 1099.
[21] Renda, F., Giorelli, M., Calisti, M., Cianchetti, M. & Laschi, C. Dynamic model of a multi-bending soft robot arm driven by cables. IEEE Trans. Robot. 30, 1109–1122 (2014).
[22] Webster, R. J. & Jones, B. A. Design and kinematic modeling of constant curvature continuum robots: a review. Int. J. Robot. Res. 29, 1661–1683 (2010).
[23] Neppalli, S., Csencsits, M. A., Jones, B. A. and Walker, I. D. (2009). Closed-form inverse kinematics for continuum manipulators. Advanced Robotics, 23: 2077–2091.
[24] Sears, P. and Dupont, P. E. (2007). Inverse kinematics of concentric tube steerable needles. IEEE International Conference on Robotics and Automation, pp. 1887–1892.
[25]Rucker, D. C. and Webster, R. J., III (2008). Mechanics-based modeling of bending and torsion in active cannulas. IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics, pp. 704–709.
[26] E Malis, F Chaumette, S Boudet. 2 1/2 D visual servoing[J]. IEEE Transactions on Robotics and Automation, 1999, 15(2): 238-250.
[27] Yanting Duan, Chensheng Cai, Pengfei Wang, Ning Wang, Ping Chen. Summary of Development of Robot Vision Servo Technology [J]. ServoControl,2007(06):16-18.
[28] Chaumette, F., Hutchinson, S.: Visual servo control Part I: Basic approaches. IEEE Robot. Autom. Mag. 13(4), 82–90 (2006).
[29] Chaumette, F., Hutchinson, S.: Visual servo control Part II: Advanced approaches. IEEE Robot. Autom. Mag. 14(1), 109–118 (2007).
[30] G. Hu, W. Mackunis, N. Gans, W. E. Dixon, J. Chen, A. Behal, D.M. Dawson, “Homography-Based Visual Servo Control with Imperfect Camera Calibration,” IEEE

Transactions on Automatic Control, vol. 54, no. 6, pp. 1318-1324, 2009.
[31] Y. H. Liu, H. Wang, C. Wang and K. Lam, “Uncalibrated visual servoing of robots using a depth-Independent interaction matrix,” IEEE Trans. on Robotics, vol. 22, no. 4, pp.804-817, 2006.
[32]H. Wang, Y. H. Liu and D. Zhou, “Adaptive visual servoing using point and line features with an uncalibrated eye-in-hand camera,” IEEE Trans. on Robotics, vol. 24, no. 4, pp. 843-857, 2008.
[33]Y. H. Liu and H. Wang, “An adaptive controller for image-based visual servoing of robot manipulators,” The 8th World Congress on Intelligent Control and Automation, pp. 988-993, 2010.
[34] Wang, H. et al. Visual servo control of cable-driven soft robotic manipulator. In Proc. International Conference on Intelligent Robots and Systems 57–62 (2013).
[35] Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985).
[36] Toussaint, Godfried T. (1983). "Solving geometric problems with the rotating calipers". Proc. MELECON '83, Athens. CiteSeerX 10.1.1.155.5671.
[37] Y. Tian, S. Yang, H. Geng, W. Wang and L. Li, "Kinematic modeling of the constant curvature continuum line drive robot", 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO), pp. 289-294, Dec 2016.
[38] S. R. Buss. "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods." Typeset manuscript. Available from World Wide Web (http://math.ucsd.edu/~sbuss/ResearchWeb/), 2004.
[39] Y. Nakamura and H. Hanafusa, Inverse kinematics solutions with singularity robustness for robot manipulator control, Journal of Dynamic Systems, Measurement, and Control, 108 (1986), pp. 163{171.
[40] C. W. Wampler, Manipulator inverse kinematic solutions based on vector formulations and damped least squares methods, IEEE Transactions on Systems, Man, and Cybernetics, 16 (1986), pp. 93{101.
[41] C. L. Lawson and R. J. Hanson, Solving Least Squares Problems. Englewood Cliffs, N. J.: Prentice-Hall, 1974.
[42] M. Shahamiri and M. Jagersand, "Uncalibrated visual servoing using a biased Newton method for on-line singularity detection and avoidance", Proc. IEEE/RSJ Int. Conf. Int. Rob. Syst., pp. 3953-3958, 2005.
[43] M. Jagersand, “Visual Servoing using trust Region Methods and Estimation of the Full Coupled Visual-Motor Jacobian”, In Proc. Of IASTED Application of Control and Robotics, pp. 105-108, 1996.
[44] M. Jagersand, “Imaged Based Visual Simulation and Tele-Assisted Robot Control”, Workshop on Recent Trends in Visual Servoing, IROS, 1997.
[45] B. H. Yoshimi, P. K. Allen “Active Uncalibrated Visual servoing”, IEEE Int. Conference

on Robotics and Automation, pp. 156-161, 1993.
[46] Rosenow JM, Sootsman WK (2007) Application accuracy of an electromagnetic field-based image-guided navigation system. Stereotact Funct Neurosurg 85:75–81.
[47] Barszcz S, Roszkowski M, Daszkiewicz P, Jurkiewicz E, Maryniak A (2007) Accuracy of intraoperative registration during electromagnetic neuronavigation in intracranial procedures performed in children. Neurol Neurochir Pol 41:122–127.
[48] Wang Z, Sun Z, Phee SJ. Haptic feedback and control of a flexible surgical endoscopic robot. Comput Methods Programs Biomed 2013; 112: 260-271 [PMID: 23561289 DOI: 10.1016/j.cmpb.2013.01.018.

Appendix

Source Code

1. Arduino

```
#include <Keyboard.h>

#include <string>

#include <sstream>

#include <math.h>

#include <AccelStepper.h>



//*** CONSTANTS ***

const bool IS_USE_LIMITS = true;

const int XY_MM_LIMIT [2] = { 25 , 30 }; // +-mm from original position, [Soft limit, hard limit]

const int Z_MM_LIMIT [2] = { 1500 , 120 }; // +/0

const float XY_STEP_TO_MM_RATIO = 0.0026;

const float Z_STEP_TO_MM_RATIO = 0.0152;

const float POSITION_TOLERANCE_MM = 0.0; //0.75; // +-

const int MAX_SPEED = 8000; //500; // The desired maximum speed in steps per second. Speeds of more than
1000 steps per second are unreliable.

const int MAX_ACCELERATION = 10000; // The desired acceleration in steps per second per second.



AccelStepper stepper1(AccelStepper::DRIVER, 1, 2);    // Stepper motor 1 - Top Left - UP
```

```
AccelStepper stepper2(AccelStepper::DRIVER, 3, 4); // Stepper motor 2 - Top Right

AccelStepper stepper3(AccelStepper::DRIVER, 5, 6); // Stepper motor 3 - Bottom Right - DOWN

AccelStepper stepper4(AccelStepper::DRIVER, 7, 8); // Stepper motor 4 - Bottom Left

AccelStepper stepperC(AccelStepper::DRIVER, 9, 10); // Stepper motor C - Center (Translation/Z)



void runSteppers() {

  stepper1.run();

  stepper3.run();

  stepper2.run();

  stepper4.run();

  stepperC.run();

}



void setTargets(float abs_target_mm[5])

{

  long abs_target_steps[5];

  abs_target_steps[0] = floor(abs_target_mm[0] / XY_STEP_TO_MM_RATIO);

  abs_target_steps[1] = floor(abs_target_mm[1] / XY_STEP_TO_MM_RATIO);

  abs_target_steps[2] = floor(abs_target_mm[2] / XY_STEP_TO_MM_RATIO);

  abs_target_steps[3] = floor(abs_target_mm[3] / XY_STEP_TO_MM_RATIO);

  abs_target_steps[4] = floor(abs_target_mm[4] / Z_STEP_TO_MM_RATIO);

    // Apply position tolerance and set target positions
```

```
  if( abs(stepper1.targetPosition()-abs_target_steps[0]) >
floor(POSITION_TOLERANCE_MM/XY_STEP_TO_MM_RATIO) ) {
    stepper1.moveTo(abs_target_steps[0]);
  }
  if( abs(stepper2.targetPosition()-abs_target_steps[1]) >
floor(POSITION_TOLERANCE_MM/XY_STEP_TO_MM_RATIO) ) {
    stepper2.moveTo(abs_target_steps[1]);
  }
  if( abs(stepper3.targetPosition()-abs_target_steps[2]) >
floor(POSITION_TOLERANCE_MM/XY_STEP_TO_MM_RATIO) ) {
    stepper3.moveTo(abs_target_steps[2]);
  }
  if( abs(stepper4.targetPosition()-abs_target_steps[3]) >
floor(POSITION_TOLERANCE_MM/XY_STEP_TO_MM_RATIO) ) {
    stepper4.moveTo(abs_target_steps[3]);
  }
  if( abs(stepperC.targetPosition()-abs_target_steps[4]) >
floor(POSITION_TOLERANCE_MM/Z_STEP_TO_MM_RATIO) ) {
    stepperC.moveTo(-1*abs_target_steps[4]);
  }
}

void setup() {
  Serial.begin(9600);
```

```
    stepper1.setMaxSpeed(MAX_SPEED);
    stepper1.setAcceleration(MAX_ACCELERATION);
    stepper2.setMaxSpeed(MAX_SPEED);
    stepper2.setAcceleration(MAX_ACCELERATION);
    stepper3.setMaxSpeed(MAX_SPEED);
    stepper3.setAcceleration(MAX_ACCELERATION);
    stepper4.setMaxSpeed(MAX_SPEED);
    stepper4.setAcceleration(MAX_ACCELERATION);
    stepperC.setMaxSpeed(10000); //20000
    stepperC.setAcceleration(10000); //10000
}

bool data_flag=false;

void loop() {
    bool reset_flag=false;
    String intchars="";
    float setting_motor[5];
    int index=0;
    while(Serial.available()>0)
    {
        char inchar=Serial.read();
```

```
    if(inchar == 's')
    {
      char *motor_info = &String("Stepper positions: "
                                 + String(stepper1.currentPosition()) + " " + String(stepper2.currentPosition()) + " " +
String(stepper3.currentPosition()) + " " + String(stepper4.currentPosition()) + " " +
String(stepperC.currentPosition()) + ".")[0];
      Serial.println(motor_info);
      break;
    }
    if(inchar =='r')
    {
      data_flag=true;
      reset_flag=true;
      break;
    }
    if(inchar != ',' and inchar != '&')
    {
      intchars+=inchar;
    }
    if(inchar == ',' )
    {

      setting_motor[index]=intchars.toFloat();
```

```
            index++;
            intchars="";
        }
        if(inchar =='&')
        {
            char *re_command = &String("Command: "
                                + String(setting_motor[0]) + " " + String(setting_motor[1]) + " " +
String(setting_motor[2]) + " " + String(setting_motor[3]) +" " + String(setting_motor[4]) + ".")[0];
            Serial.println(re_command);
            data_flag=true;
            break;
        }
    }
    if(data_flag)
    {
        if(reset_flag==true)
        {
            Serial.println("Reset");
            stepper1.moveTo(0);
            stepper2.moveTo(0);
            stepper3.moveTo(0);
            stepper4.moveTo(0);
            stepperC.moveTo(0);
```

```
        reset_flag=false;
    }
    else
    {
        float abs_target_mm[5];
        abs_target_mm[0] = setting_motor[0];
        abs_target_mm[1] = setting_motor[1];
        abs_target_mm[2] = setting_motor[2];
        abs_target_mm[3] = setting_motor[3];
        abs_target_mm[4] = setting_motor[4];
        abs_target_mm[0] = min( max(abs_target_mm[0],-1*XY_MM_LIMIT[0]) , XY_MM_LIMIT[0]);
        abs_target_mm[1] = min( max(abs_target_mm[1],-1*XY_MM_LIMIT[0]) , XY_MM_LIMIT[0]);
        abs_target_mm[2] = min( max(abs_target_mm[2],-1*XY_MM_LIMIT[0]) , XY_MM_LIMIT[0]);
        abs_target_mm[3] = min( max(abs_target_mm[3],-1*XY_MM_LIMIT[0]) , XY_MM_LIMIT[0]);
        abs_target_mm[4] = min( max(abs_target_mm[4],0) , Z_MM_LIMIT[0]);
        setTargets(abs_target_mm);
        char *motor_info = &String("Move to: "
                                + String(stepper1.targetPosition()) + " " + String(stepper2.targetPosition()) + " " +
String(stepper3.targetPosition()) + " " + String(stepper4.targetPosition()) + " " +
String(stepperC.targetPosition()) + ".")[0];
        Serial.println(motor_info);
    }
    data_flag=false;
```

```
    }
    runSteppers();
    delay(0.1);
}
```

2. Simulation Code - MATLAB

2.1 working_space.m

```
clc;
clear all;
length=56;
i=1;
r=1.8
for alpha=0:0.01:2*pi
    for beta= 0:0.01:pi
        ux(i)=cos(alpha)^2*cos(beta)+sin(alpha);
        uy(i)=cos(alpha)*sin(alpha)*cos(beta);
        uz(i)=-cos(alpha)*sin(beta);

        vx(i)=cos(alpha)*sin(alpha+beta)-cos(alpha)*sin(beta);
        vy(i)=sin(alpha)^2*cos(beta)+ cos(beta)^2;
        vz(i)=-sin(alpha)*sin(beta);

        wx(i)=cos(alpha)*sin(beta);
```

```
            wy(i)= sin(alpha)*sin(beta);

            wz(i)= cos(beta);



            tx(i)= (length/beta)*cos(alpha)*(1-cos(beta));

            ty(i)=(length/beta)*sin(alpha)*(1-cos(beta));

            tz(i)=(length/beta)*sin(beta);



            l1(i)=r*beta*cos(alpha);

            l2(i)=r*beta*cos(alpha-pi/2);

            l3(i)=r*beta*cos(alpha-pi);

            l4(i)=r*beta*cos(alpha-3*pi/2);

            i=i+1;

        end

end

for index=1:1:i-1

    T(:,:,index)=[ux(index) vx(index) wx(index) tx(index);uy(index) vy(index) wy(index) ty(index);uz(index)
vz(index) wz(index) tz(index);0 0 0 1];

end

end_effctor = [0 0 0 1];

end_effctor.*T(:,:,1)

for index=1:1:i-1

    result_temp(:,:,index)= end_effctor.*T(:,:,index);

    result(:,1,index) = result_temp(1:3,4,index);
```

```
    x(:,index)=result(1,1,index);

    y(:,index)=result(2,1,index);

    z(:,index)=result(3,1,index);

end

figure(1)

plot3(x,y,z);

title("MicroDART End-effecter Working Space (Constant Curvature)")

xlabel("x/mm")

ylabel("y/mm")

zlabel("z/mm")

patch(x,y,z,z,'edgecolor','flat','facecolor','none')

view(3)

grid on;

figure(2)

plot3(tx,ty,tz);

title("MicroDART End-effecter Working Space (Constant Curvature)")

xlabel("x/mm")

ylabel("y/mm")

zlabel("z/mm")

patch(x,y,z,z,'edgecolor','flat','facecolor','none')

view(3)

grid on;
```

2.2 simulation.m

```
clear; clc;

close all

%% target generation

t=1;

for x1=0:-1:-30

    target_p(:,t)=[x1 0 56];

    t=t+1;

end

for y1=1:1:15

     target_p(:,t)=[-30 y1 56];

     t=t+1;

end

for x2=-29:1:30

    target_p(:,t)=[x2 15 56];

    t=t+1;

end



for y2=14:-1:0

     target_p(:,t)=[30 y2 56];

     t=t+1;

end
```

```
for x3=29:-1:0
    target_p(:,t)=[x3 0 56];
    t=t+1;
end
%% initialization
length=55;
lamba=0.5; % damping coefficient of Jacobain inverse
angle_change=[0.01,0.01];
angle_diff=[0.001,0.002];
angle_change=angle_change+angle_diff;
position_changed = Forward(length,angle_change(1),angle_change(2))
position_pre = position_changed;
Jac_estimate=Jacobian(length,angle_change(1),angle_change(2))
%% literation
for i=1:t
    diff_t=target_p(1:2,i)-position_changed;
    angle_diff =Jac_estimate'*inv(Jac_estimate*Jac_estimate'+lamba*eye(2))*diff_t; %jacobian with
dampping
    angle_change=angle_change+angle_diff';
    position_changed = Forward(length,angle_change(1),angle_change(2));
    changing = position_changed-position_pre;
    update_p(i,:)=position_changed;
    error_n= target_p(1:2,i)-position_changed
```

```
error = sqrt(error_n(1)^2+error_n(2)^2);

error_plot(i)=[error];

error_plotxy(i,:)=[error_n(1),error_n(2)];

Jac_estimate = est_Jac(Jac_estimate, angle_diff, changing);

%Jac_estimate = Jacobian(length,angle_change(1),angle_change(2))

position_pre = position_changed;

update_p(i,:)=position_pre;

figure(1)

plot(error_plot,'r-','LineWidth',2);

axis([0 150 -0.2 2])

%title('(c) Total error in [x,y] over time','position',[50 -0.6]);

ylabel('Displacement [mm]');

xlabel("Time [s]");

grid on

drawnow;

if i>95

   print('EJ','-dpdf','-r600');

end

subplot(1,2,1)

plot(target_p(1,:),'b-','LineWidth',1.5);

hold on

plot(update_p(:,1),'r--','LineWidth',1.5);
```

```
    axis([0 150 -32 32])
    xlabel("Time [s]");
    ylabel('Displacement in x [mm]');
    legend('Trajectory','Experiment','location','southeast')
    grid on
    drawnow;
    subplot(1,2,2)
    plot(target_p(2,:),'b-','LineWidth',1.5);

    hold on
    plot(update_p(:,2),'r--','LineWidth',1.5);
    axis([0 150 -5 16])
    xlabel("Time [s]");
    ylabel('Displacement in y [mm]');
    legend('Trajectory','Experiment','location','southeast')
    grid on
    drawnow;
    if i>149
       print('EJ_xy','-dpdf','-r600');
    end
end
```

2.3 forward_kinematics.m

```
function P = Forward(length,alpha,beta)

        if beta==0

        tx= (length)*cos(alpha)*(1-cos(beta));

        ty=(length)*sin(alpha)*(1-cos(beta));

        %tz=(length)*sin(beta);

        else

        tx= (length/beta)*cos(alpha)*(1-cos(beta));

        ty=(length/beta)*sin(alpha)*(1-cos(beta));

        %tz=(length/beta)*sin(beta);

        end

        %P=[tx; ty; tz];

        P=[tx; ty;]% tz];

end
```

2.4 Jacobian.m

```
function J=Jacobian(L,A,B)

J=[(L*sin(A)*(cos(B) - 1))/B (L*cos(A)*sin(B))/B + (L*cos(A)*(cos(B) - 1))/B^2;

    -(L*cos(A)*(cos(B) - 1))/B (L*sin(A)*(cos(B) - 1))/B^2 + (L*sin(A)*sin(B))/B];

    %0 (L*cos(B))/B - (L*sin(B))/B^2];

end
```

2.5 estimat_jacobian.m

```
function Jac_estimation = est_Jac(Jac_e_prev, d_q, d_p)

kappa = 0.2;
```

```
temp1= d_p-(Jac_e_prev*d_q)

d_q'

temp2=temp1*d_q'

ov = d_q'*d_q

temp3=kappa*temp2/ov

temp3+Jac_e_prev

Jac_estimation = (kappa*(((d_p-(Jac_e_prev*d_q))*d_q')/(d_q'*d_q)))+Jac_e_prev;
```

3. Robot Control – Python

3.1 MicroDART.py

```
import numpy as np

import math



class MicroDART:

    def __init__(self,L):

        self.length = L

        print("MicroDART Created, Catheter Length",L,"mm")



    def forward(self,alpha,beta):

        tx=(self.length/beta)*math.cos(alpha)*(1-math.cos(beta))

        ty=(self.length/beta)*math.sin(alpha)*(1-math.cos(beta))

        tz=(self.length/beta)*math.sin(beta)

        return np.array([tx,ty,tz])
```

```
    def jacobian(self,alpha,beta):
        J11= (self.length*math.sin(alpha)*(math.cos(beta) - 1))/beta
        J12= (self.length*math.cos(alpha)*math.sin(beta))/beta +
(self.length*math.cos(alpha)*(math.cos(beta) - 1))/math.pow(beta,2)
        J_1=np.array([J11,J12])
        J21= -(self.length*math.cos(alpha)*(math.cos(beta) - 1))/beta
        J22= (self.length*math.sin(alpha)*(math.cos(beta) - 1))/math.pow(beta,2) +
(self.length*math.sin(alpha)*math.sin(beta))/beta
        J_2=np.array([J21,J22])
        J31=0
        J32=(self.length*math.cos(beta))/beta - (self.length*math.sin(beta))/pow(beta,2)
        J_3=np.array([J31,J32])
        return np.array([J_1,J_2,J_3])

    def forward_noZ(self,alpha,beta):
        tx=(self.length/beta)*math.cos(alpha)*(1-math.cos(beta))
        ty=(self.length/beta)*math.sin(alpha)*(1-math.cos(beta))
        #tz=(self.length/beta)*math.sin(beta)
        return np.array([tx,ty])

    def jacobian_noZ(self,alpha,beta):
        J11= (self.length*math.sin(alpha)*(math.cos(beta) - 1))/beta
```

```
            J12= (self.length*math.cos(alpha)*math.sin(beta))/beta +
(self.length*math.cos(alpha)*(math.cos(beta) - 1))/math.pow(beta,2)

            J_1=np.array([J11,J12])

            J21= -(self.length*math.cos(alpha)*(math.cos(beta) - 1))/beta

            J22= (self.length*math.sin(alpha)*(math.cos(beta) - 1))/math.pow(beta,2) +
(self.length*math.sin(alpha)*math.sin(beta))/beta

            J_2=np.array([J21,J22])

            return np.array([J_1,J_2])



        def wire_change(self,alpha,beta):

            radius=2.8

            l1=radius*beta*math.cos(alpha)    #l

            l2=radius*beta*math.cos(alpha-math.pi/2)       #u

            l3=radius*beta*math.cos(alpha-math.pi)    #r

            l4=radius*beta*math.cos(alpha-3*math.pi/2)    #d

            return [-l2,-l4,l1,l3,self.z_displace(alpha,beta)]



        def z_displace(self,alpha,beta):

            return self.length-(self.length/beta)*math.sin(beta)



        def est_jac(self,jac_pre,angle_diff,changing):

            kappa = 0.2

            temp1 = changing-np.matmul(jac_pre,angle_diff)
```

```
        angle_diff=np.array([[angle_diff[0],angle_diff[1]]])
        temp1 = np.array([[temp1[0]],[temp1[1]]])
        temp2 = np.matmul(temp1,angle_diff)
        temp3 = temp2/np.matmul(angle_diff,angle_diff.T)
        Jac_estimation = (kappa*temp3)+jac_pre
        print("ov",np.matmul(angle_diff,angle_diff.T))
        print("J:",Jac_estimation)
        return Jac_estimation

    def cable2angle(self,cable_change):
        radius=2.8
        alpha = math.atan(-(-cable_change[0]/cable_change[1]))
        beta = (cable_change[1])/(radius*math.cos(alpha))
        return [alpha,beta]
```

3.2 ractangular.py

```
import numpy as np
import MicroDART
import serial
from matplotlib import pyplot as plt

def withZ():
    robot = MicroDART.MicroDART(50)
```

```
## targets

target_list = []

for x1 in range(0,-11,-1):

    target_list.append(np.array([x1,0,56]))

for y1 in range(1,11,1):

    target_list.append(np.array([-10,y1,56]))

for x2 in range(-9,11,1):

    target_list.append(np.array([x2,10,56]))

for y2 in range(9,-1,-1):

    target_list.append(np.array([10,y2,56]))

for x3 in range(9,-1,-1):

    target_list.append(np.array([x3,0,56]))

target_array=np.array(target_list)



## initialization

lengeth = 50

lamda = 0.5

angle_change=np.array([0.01,0.01])

angle_diff=np.array([0.001,0.002])

angle_change=angle_change+angle_diff

position_changed=robot.forward(angle_change[0],angle_change[1])    # initial position

position_pre = position_changed # save position
```

```
jac_estimate=robot.jacobian(angle_change[0],angle_change[1])

fig, ax = plt.subplots(figsize=(8,4))

draw_temp=[]

draw_result=[]

## Run

for loop in range(np.shape(target_array)[0]):

    diff_target = target_array[loop]-position_changed

    temp=np.matmul(jac_estimate,jac_estimate.T)+lamda*np.identity(3)

    angle_diff = np.matmul(np.matmul(jac_estimate.T,np.linalg.inv(temp)),diff_target)

    angle_change=angle_change+angle_diff

    position_changed = robot.forward(angle_change[0],angle_change[1])

    changing = position_changed-position_pre;

    jac_estimate = robot.jacobian(angle_change[0],angle_change[1])

    position_pre = position_changed

    draw_result.append(position_changed)

    draw_temp.append(target_array[loop])

    for point in draw_temp:

        ax.plot(point[0],point[1],'ro')

    for point in draw_result:

        ax.plot(point[0],point[1],'g*')

    ax.set_xlim(-11,11)

    ax.set_ylim(-1,11)
```

```
        plt.draw()

        plt.pause(0.5)



def withoutZ():

    length = 55

    port=serial.Serial("/dev/ttyACM0",9600,timeout=0.5)

    robot = MicroDART.MicroDART(length)

    ## targets

    target_list = []

    for x1 in range(0,-31,-1):

        target_list.append(np.array([x1,0]))

    for y1 in range(1,21,1):

        target_list.append(np.array([-30,y1]))

    for x2 in range(-29,31,1):

        target_list.append(np.array([x2,20]))

    for y2 in range(19,-1,-1):

        target_list.append(np.array([30,y2]))

    for x3 in range(29,-1,-1):

        target_list.append(np.array([x3,0]))

    target_array=np.array(target_list)

    ## initialization

    lamda = 0.5
```

```
angle_change=np.array([0.01,0.01])

angle_diff=np.array([0.001,0.002])

angle_change=angle_change+angle_diff

position_changed=robot.forward_noZ(angle_change[0],angle_change[1])    # initial position

position_pre = position_changed # save position

jac_estimate=robot.jacobian_noZ(angle_change[0],angle_change[1])

fig, (ax,ax2) = plt.subplots(1,2,figsize=(12,4))

## Run

estemate = False

for loop in range(np.shape(target_array)[0]):

    diff_target = target_array[loop]-position_changed

    #print(diff_target,"=",target_array[loop],"-",position_changed)

    temp=np.matmul(jac_estimate,jac_estimate.T)+lamda*np.identity(2)

    angle_diff = np.matmul(np.matmul(jac_estimate.T,np.linalg.inv(temp)),diff_target)

    #print("A:",angle_change,"+",angle_diff)

    angle_change=angle_change+angle_diff

    #print("A_new",angle_change)

    position_changed = robot.forward_noZ(angle_change[0],angle_change[1])

    changing = position_changed-position_pre

    if estemate is True:

        jac_estimate=robot.est_jac(jac_estimate,angle_diff,changing)

    else:
```

```
            jac_estimate = robot.jacobian_noZ(angle_change[0],angle_change[1])
        position_pre = position_changed
        # draw
        ax.plot(target_array[loop][0],target_array[loop][1],'g*')
        ax.set_xlim(-35,35)
        ax.set_ylim(-30,30)
        ax.set_title('Target')
        ax2.plot(position_changed[0],position_changed[1],'r.')
        ax2.set_title('MicroDART')
        ax2.set_xlim(-35,35)
        ax2.set_ylim(-30,30)
        plt.draw()
        # control
        wires = robot.wire_change(angle_change[0],angle_change[1])
        plt.pause(0.2)
        command = "%5f,%5f,%5f,%5f,%5f,&"%(wires[0],wires[1],wires[2],wires[3],wires[4])
        print(command)

print("%5f,%5f,%5f,%5f,%5f,&"%(wires[0]/0.0026,wires[1]/0.0026,wires[2]/0.0026,wires[3]/0.0026,wires[4])
)
        port.write(command.encode('utf-8'))

    port.write("0,0,0,0,0&".encode('utf-8'))
```

```
    port.close()

withoutZ()
```

3.3 keyboard.py

```
from evdev import InputDevice

from select import select

import numpy as np

import threading

import MicroDART

import serial



up_flag = False

down_flag=False

left_flag=False

right_flag=False

reset_flag = False

forward_flag = False

back_flag = False



#cat /proc/bus/input/devices

def detectInputKey():

    global up_flag,down_flag,left_flag,right_flag,reset_flag,forward_flag,back_flag

    dev = InputDevice('/dev/input/event3')
```

```
while True:
    select([dev],[],[])
    for event in dev.read():
        if event.type == 1:             # Keyboard
            if event.code == 72:     # key-8-up
                if event.value == 2:       # move
                    up_flag=True
                elif event.value == 0: # stop
                    up_flag=False
            elif event.code== 80:     #key 2 down
                if event.value == 2:       # move
                    down_flag=True
                elif event.value == 0: # stop
                    down_flag=False
            elif event.code== 75:     #key 4 left
                if event.value == 2:       # move
                    left_flag=True
                elif event.value == 0: # stop
                    left_flag=False
            elif event.code== 77:     #key 6 right
                if event.value == 2:       # move
                    right_flag=True
```

```
                elif event.value == 0: # stop
                    right_flag=False
            elif event.code ==19:
                if event.value == 1:
                    reset_flag = True
            elif event.code ==78:
                if event.value == 2:
                    forward_flag=True
                elif event.value == 0 :
                    forward_flag=False
            elif event.code ==74:
                if event.value == 2:
                    back_flag=True
                elif event.value == 0 :
                    back_flag=False


def control():
    global up_flag,down_flag,left_flag,right_flag,reset_flag,forward_flag,back_flag
    wire = np.array([0.001,0.001,0.001],dtype=float)
    speed = 0.000008
    speed_base = 0.001
    count = 0
```

```
port=serial.Serial("/dev/ttyACM0",9600,timeout=0.5)

length = 50

robot = MicroDART.MicroDART(length)

z_displace = 0

show_calcu= True

while True:

    count =count+1

    if up_flag is True:

        wire[0]=wire[0]+speed

    elif down_flag is True:

        wire[0]=wire[0]-speed

    elif left_flag:

        wire[1]=wire[1]-speed

    elif right_flag:

        wire[1]=wire[1]+speed

    elif forward_flag:

        wire[2]=wire[2]+speed_base

    elif back_flag:

        wire[2]=wire[2]-speed_base

        if wire[2] < 0:

            wire[2]=0
```

```
        angle_change = robot.cable2angle(wire)
        z_displace = robot.z_displace(angle_change[0],angle_change[1])

        if reset_flag:
            print("Reset")
            command = "%5f,%5f,%5f,%5f,%5f,&"%(0,0,0,0,0)
            print("Command: ",command)
            port.write(command.encode('utf-8'))
            reset_flag = False
            wire = np.array([0.001,0.001,0.001],dtype=float)

        if count == 8000:
            print(wire)
            command = "%5f,%5f,%5f,%5f,%5f,&"%(-wire[0],wire[0],wire[1],-wire[1],wire[2]+z_displace)
            print("Command: ",command)
            if show_calcu:
                print("angles:",angle_change)
                print("positions:",robot.forward_noZ(angle_change[0],angle_change[1]))
            port.write(command.encode('utf-8'))
            count = 0;

if __name__=='__main__':
```

```
    t1 = threading.Thread(target=detectInputKey)

    t2 = threading.Thread(target=control)

    t1.start()

    t2.start()
```

3.4 joystick.py

```
from evdev import InputDevice

from select import select

import numpy as np

import threading

import MicroDART

import serial



up_flag = False

down_flag=False

left_flag=False

right_flag=False

reset_flag = False

forward_flag = False

back_flag = False



#cat /proc/bus/input/devices
```

```
def detectInputKey():
    global up_flag,down_flag,left_flag,right_flag,reset_flag,forward_flag,back_flag
    dev = InputDevice('/dev/input/event26')
    while True:
        select([dev],[],[])
        for event in dev.read():
            if event.type == 3:


                if event.code == 1:
                    if event.value == 0:       # move
                        up_flag=True
                    elif event.value == 2:
                        down_flag=True
                    elif event.value == 1: # stop
                        up_flag=False
                        down_flag=False


                elif event.code== 0:     #key 4 left
                    if event.value == 0:       # move
                        left_flag=True
                    elif event.value == 2: # stop
                        right_flag=True
```

```
                elif event.value == 1: # stop
                    left_flag=False
                    right_flag=False
            elif event.type ==1:

                if event.code ==311:
                    if event.value == 1:
                        forward_flag=True
                    elif event.value == 0 :
                        forward_flag=False
                elif event.code ==310:
                    if event.value == 1:
                        back_flag=True
                    elif event.value == 0 :
                        back_flag=False

                elif event.code ==305:
                    if event.value == 1:
                        reset_flag = True
def control():
    global up_flag,down_flag,left_flag,right_flag,reset_flag,forward_flag,back_flag
    wire = np.array([0.0001,0.0001,0.0001],dtype=float)
```

```
speed = 0.000005

speed_base = 0.001

count = 0

port=serial.Serial("/dev/ttyACM0",9600,timeout=0.5)

length = 55

robot = MicroDART.MicroDART(length)

z_displace = 0

while True:

    count =count+1

    if up_flag is True:

        wire[0]=wire[0]+speed

    elif down_flag is True:

        wire[0]=wire[0]-speed

    elif left_flag:

        wire[1]=wire[1]-speed

    elif right_flag:

        wire[1]=wire[1]+speed

    elif forward_flag:

        wire[2]=wire[2]+speed_base

    elif back_flag:

        wire[2]=wire[2]-speed_base

        if wire[2] < 0:
```

```
                wire[2]=0


        angle_change = robot.cable2angle(wire)
        z_displace = robot.z_displace(angle_change[0],angle_change[1])


        if reset_flag:
            print("Reset")
            command = "%5f,%5f,%5f,%5f,%5f,&"%(0,0,0,0,0)
            print("Command: ",command)
            port.write(command.encode('utf-8'))
            reset_flag = False
            wire = np.array([0.001,0.001,0.001],dtype=float)


        if count == 8000:
            print(wire)
            command = "%5f,%5f,%5f,%5f,%5f,&"%(-wire[0],wire[0],wire[1],-wire[1],wire[2]+z_displace)
            print("Command: ",command)
            port.write(command.encode('utf-8'))
            count = 0;


if __name__=='__main__':
    t1 = threading.Thread(target=detectInputKey)
```

```
    t2 = threading.Thread(target=control)

    t1.start()

    t2.start()
```

3.5 servoing.py

```
import cv2

import numpy as np

import MicroDART

import serial

from matplotlib import pyplot as plt



def targetDetector_p(src):

    x_r=0

    y_r=0

    x_b=0

    y_b=0

    b,g,r=cv2.split(src)

    ret,b = cv2.threshold(b,80,255,cv2.THRESH_BINARY)

    ret,g =    cv2.threshold(g,100,255,cv2.THRESH_BINARY)

    ret,r =    cv2.threshold(r,30,255,cv2.THRESH_BINARY)

    binery_r=r-b

    ret,binery_r =    cv2.threshold(binery_r,50,255,cv2.THRESH_BINARY)
```

```
        erotion = cv2.erode(binery_r,np.ones((4,4),np.uint8))
        dilation= cv2.dilate(erotion,np.ones((6,6),np.uint8))
        #cv2.imshow("red",dilation)
        contours, hierarchy =
cv2.findContours(dilation,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        for contour in contours:
            area = cv2.contourArea(contour)
            if area>500 and area<1500:
                rect = cv2.minAreaRect(contour)
                box = cv2.boxPoints(rect)
                x_r=np.int0((box[0][0]+box[1][0]+box[2][0]+box[3][0])/4)
                y_r=np.int0((box[0][1]+box[1][1]+box[2][1]+box[3][1])/4)
                #cv2.drawContours(src, [box], 0, (255, 0, 0), 2)
                cv2.circle(src, (x_r, y_r), 3, (0, 0, 255), 2)
        binery_b=b-r
        ret,binery_b =    cv2.threshold(binery_b,50,255,cv2.THRESH_BINARY)
        erotion_b = cv2.erode(binery_b,np.ones((6,6),np.uint8))
        dilation_b= cv2.dilate(erotion_b,np.ones((6,6),np.uint8))
        #cv2.imshow("blue",dilation_b)
        contours_b, hierarchy_b =
cv2.findContours(dilation_b,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        for contour in contours_b:
            area = cv2.contourArea(contour)
```

```
            if area>500 and area<2000:
                rect = cv2.minAreaRect(contour)
                box = cv2.boxPoints(rect)
                x_b=np.int0((box[0][0]+box[1][0]+box[2][0]+box[3][0])/4)
                y_b=np.int0((box[0][1]+box[1][1]+box[2][1]+box[3][1])/4)
                #cv2.drawContours(src, [box], 0, (255, 0, 0), 2)
                cv2.circle(src, (x_b, y_b), 3, (255, 0, 0), 2)

        cv2.imshow("target",src)
        return [x_r,479-y_r],[x_b,479-y_b]

def targetDetecot(src):
        src_temp=src.copy()
        b,g,r=cv2.split(src_temp)
        ret,b = cv2.threshold(b,80,255,cv2.THRESH_BINARY)
        ret,g =    cv2.threshold(g,100,255,cv2.THRESH_BINARY)
        ret,r =    cv2.threshold(r,30,255,cv2.THRESH_BINARY)
        binery = r+b+g
        ret,binery =    cv2.threshold(binery,50,255,cv2.THRESH_BINARY_INV)
        erotion = cv2.erode(binery,np.ones((4,4),np.uint8))
        dilation= cv2.dilate(erotion,np.ones((6,6),np.uint8))
        #cv2.imshow("binary",dilation)
```

```
    contours, hierarchy =
cv2.findContours(dilation,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(src_temp, contours, -1, (150, 100, 255), 2)
    tips_x=[]
    tips_y=[]
    for contour in contours:
        area = cv2.contourArea(contour)
        if area>2000 and area<35000:
            rect = cv2.minAreaRect(contour)
            box = np.int0(cv2.boxPoints(rect))
            cv2.drawContours(src_temp, [box], 0, (0, 0, 255), 2)
            y_p=np.array([box[0][1],box[1][1],box[2][1],box[3][1]])
            y_p=np.argsort(y_p)
            slope = (box[y_p[0]][1]-box[y_p[1]][1])/(box[y_p[0]][0]-box[y_p[1]][0])
            bias = box[y_p[0]][1]-slope*box[y_p[0]][0]
            for p in contour:
                temp = slope*p[0][0]+bias-p[0][1]
                if abs(temp) < 2:
                    tips_x.append(p[0][0])
                    tips_y.append(p[0][1])
    tips=np.empty(2)
    if len(tips_x) is not 0 and len(tips_y) is not 0:
        tips = np.int0(np.array([np.mean(tips_x),np.mean(tips_y)]))
```

```
            #print("Tips",tips)
            cv2.circle(src_temp,(tips[0],tips[1]), 3, (0, 255, 255), 2)
            cv2.circle(src,(tips[0],tips[1]), 3, (0, 100, 255), 2)
        x_b=0
        y_b=0
        binery_b=b-r
        ret,binery_b =    cv2.threshold(binery_b,50,255,cv2.THRESH_BINARY)
        erotion_b = cv2.erode(binery_b,np.ones((6,6),np.uint8))
        dilation_b= cv2.dilate(erotion_b,np.ones((6,6),np.uint8))
        #cv2.imshow("blue",dilation_b)
        contours_b, hierarchy_b =
cv2.findContours(dilation_b,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        for contour in contours_b:
            area = cv2.contourArea(contour)
            if area>300 and area<2000:
                rect = cv2.minAreaRect(contour)
                box = cv2.boxPoints(rect)
                x_b=np.int0((box[0][0]+box[1][0]+box[2][0]+box[3][0])/4)
                y_b=np.int0((box[0][1]+box[1][1]+box[2][1]+box[3][1])/4)
                cv2.drawContours(src_temp, [np.int0(cv2.boxPoints(rect))], 0, (255, 0, 0), 2)
                cv2.circle(src_temp, (x_b, y_b), 3, (0, 255, 255), 2)
                cv2.circle(src, (x_b, y_b), 3, (255, 100, 0), 2)
```

```
        cv2.imshow("preview",src_temp)

        cv2.imshow("target",src)

        return [tips[0],479-tips[1]],[x_b,479-y_b]



lamda = 0.5

length = 50

robot = MicroDART.MicroDART(length)

port=serial.Serial("/dev/ttyACM0",9600,timeout=0.5)

servo_flag = False

init_flag = True

cam_name1 = "/dev/video0"

cap = cv2.VideoCapture(cam_name1)



angle_change=np.array([0.01,0.01])

angle_diff=np.array([0.001,0.002])

angle_change=angle_change+angle_diff

jac_estimate=robot.jacobian_noZ(angle_change[0],angle_change[1])



draw_position_old = robot.forward_noZ(angle_change[0],angle_change[1])    # initial position

fig,ax = plt.subplots(figsize=(8,4))

ax.set_title('Trajectory (simulation)')

ax.plot(draw_position_old[0],draw_position_old[1],'ro')
```

```
motor_moved = False

estimate =    False

changing = 0

while 1:

    ret, frame = cap.read()

    #cv2.imshow("cap", frame)

    #tip,target = targetDetector(frame)

    tip,target = targetDetecot(frame)

    # print(tip,target)

    key = cv2.waitKey(10)

    if key == ord('q'):

        break

    if key == ord('s'):

        servo_flag=True

    if key == ord('n'):

        servo_flag=False

    if key == ord('r'):

        port.write("0,0,0,0,0,0&".encode('utf-8'))

    if target!=[0,0] and tip!=[0,0] and init_flag is True:

        if target!=[0,0] and tip!=[0.5,478]:

            position_pre = np.array(tip)

            init_flag=False
```

```
if motor_moved is True and estimate is True:
    position_changed = np.array(tip)
    #changing = (position_changed-position_pre)/100
    #changing = robot.forward_noZ(angle_change[0],angle_change[1])-draw_position_old
    #jac_estimate=robot.est_jac(jac_estimate,angle_diff,changing)
    motor_moved = False

if servo_flag is True and init_flag is not True:
    if target!=[0,0] and tip!=[0.5,478]:
        position_changed = np.array(tip)
        target_array = np.array(target)
        if estimate:
            diff_target = (target_array-position_changed)/50
        else:
            diff_target = (target_array-position_changed)/50
        print("tip:",position_changed,"-target:",target_array,"-->diff:", diff_target)
        temp=np.matmul(jac_estimate,jac_estimate.T)+lamda*np.identity(2)
        angle_diff = np.matmul(np.matmul(jac_estimate.T,np.linalg.inv(temp)),diff_target)
        angle_change=angle_change+angle_diff
        position_pre = position_changed

        if estimate is not True:
```

```
                jac_estimate = robot.jacobian_noZ(angle_change[0],angle_change[1])

            wires = robot.wire_change(angle_change[0],angle_change[1])

            command = "%5f,%5f,%5f,%5f,%5f,&"%(wires[0],wires[1],wires[2],wires[3],wires[4])

            print("Command: ",command)

            port.write(command.encode('utf-8'))

            servo_flag=False

            motor_moved = True

            # draw

            draw_position=robot.forward_noZ(angle_change[0],angle_change[1])

            print("Forward kinamatic:",draw_position)

            ax.plot(draw_position[0],draw_position[1],'b*')

            ax.plot([draw_position_old[0],draw_position[0]],[draw_position_old[1],draw_position[1]],'g-')

            changing = draw_position-draw_position_old

            print("Change:",changing)

            jac_estimate=robot.est_jac(jac_estimate,angle_diff,changing)

            draw_position_old=draw_position

    plt.draw()

    plt.pause(0.001)

port.close()

cap.release()

cv2.destroyAllWindows()
```

4. Data recording

4.1 Catheter_record.py

```
import cv2

import numpy as np

import MicroDART

import serial

from matplotlib import pyplot as plt

import csv



def targetDetecot(src):

    src_temp=src.copy()

    b,g,r=cv2.split(src_temp)

    ret,b = cv2.threshold(b,80,255,cv2.THRESH_BINARY)

    ret,g =  cv2.threshold(g,100,255,cv2.THRESH_BINARY)

    ret,r =  cv2.threshold(r,30,255,cv2.THRESH_BINARY)

    binery = r+b+g

    ret,binery =  cv2.threshold(binery,50,255,cv2.THRESH_BINARY_INV)

    erotion = cv2.erode(binery,np.ones((4,4),np.uint8))

    dilation= cv2.dilate(erotion,np.ones((6,6),np.uint8))

    #cv2.imshow("binary",dilation)

    contours, hierarchy =
cv2.findContours(dilation,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
```

```
cv2.drawContours(src_temp, contours, -1, (150, 100, 255), 2)

tips_x=[]

tips_y=[]

for contour in contours:

    area = cv2.contourArea(contour)

    if area>2000 and area<28000:

        rect = cv2.minAreaRect(contour)

        box = np.int0(cv2.boxPoints(rect))

        cv2.drawContours(src_temp, [box], 0, (0, 0, 255), 2)

        y_p=np.array([box[0][1],box[1][1],box[2][1],box[3][1]])

        y_p=np.argsort(y_p)

        slope = (box[y_p[0]][1]-box[y_p[1]][1])/(box[y_p[0]][0]-box[y_p[1]][0])

        bias = box[y_p[0]][1]-slope*box[y_p[0]][0]

        for p in contour:

            temp = slope*p[0][0]+bias-p[0][1]

            if abs(temp) < 2:

                tips_x.append(p[0][0])

                tips_y.append(p[0][1])

tips=np.empty(2)

if len(tips_x) is not 0 and len(tips_y) is not 0:

    tips = np.int0(np.array([np.mean(tips_x),np.mean(tips_y)]))

    #print("Tips",tips)
```

```
            cv2.circle(src_temp,(tips[0],tips[1]), 3, (0, 255, 255), 2)
            cv2.circle(src,(tips[0],tips[1]), 3, (0, 100, 255), 2)


        x_b=0
        y_b=0
        binery_b=b-r
        ret,binery_b =    cv2.threshold(binery_b,50,255,cv2.THRESH_BINARY)
        erotion_b = cv2.erode(binery_b,np.ones((6,6),np.uint8))
        dilation_b= cv2.dilate(erotion_b,np.ones((6,6),np.uint8))
        #cv2.imshow("blue",dilation_b)
        contours_b, hierarchy_b =
cv2.findContours(dilation_b,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        for contour in contours_b:
            area = cv2.contourArea(contour)
            if area>500 and area<2000:
                rect = cv2.minAreaRect(contour)
                box = cv2.boxPoints(rect)
                x_b=np.int0((box[0][0]+box[1][0]+box[2][0]+box[3][0])/4)
                y_b=np.int0((box[0][1]+box[1][1]+box[2][1]+box[3][1])/4)
                cv2.drawContours(src_temp, [np.int0(cv2.boxPoints(rect))], 0, (255, 0, 0), 2)
                cv2.circle(src_temp, (x_b, y_b), 3, (0, 255, 255), 2)
                cv2.circle(src, (x_b, y_b), 3, (255, 100, 0), 2)
```

```
        cv2.imshow("preview",src_temp)
        cv2.imshow("target",src)
        return [tips[0],479-tips[1]],[x_b,479-y_b]


cam_name1 = "/dev/video0"
cap = cv2.VideoCapture(cam_name1)
fig, (ax,ax2) = plt.subplots(1,2,figsize=(12,4))
with open('data2.csv','w',newline='') as f_csv:
    while(1):
        ret, frame = cap.read()
        cv2.imshow("cap", frame)
        tip,target = targetDetecot(frame)
        cv2.waitKey(100)
        f_writer = csv.writer(f_csv)
        f_writer.writerow(tip)
```