1/21/2024

Revolutionizing Surgery: The Introduction of the Da Vinci Surgical System

Graduation Project (1)

IDs	Students' Names
7925	Faress Ahmed Mohamed Amin
7543	Ahmed Mohamed Kotp
7818	Mohamed Hussein Elzoheiry
7438	Abdallah Ossama Elsayed
7717	Omneya Haytham Mohamed
7735	Kholoud Waleed Ali Moustafa
7416	Abdelrahman Gamal Bakr

Table of Contents

1. Introduction: 3
2. Da vinci surgical robot: 7
3. Aims and objectives: 10
4. Market study: 112
5. Brief History of Robotic Surgery 13
6. Our Project: 15
   6.1. Project description and operation: 15
   6.2. Design, Simulation, & Hardware: 16
   6.3. Computer Vision: 77
   6.4. Path planning: 140
6. Project Timeline: 150
7. Cost analysis: 151
8. Conclusion: 152
9. References: 153

Abstract

Robot-assisted surgical systems are becoming increasingly common in medical procedures as they embrace many of the benefits of minimally invasive surgery including less trauma, recovery time and financial costs associated to the treatment after surgery. These robotic systems allow the surgeons to navigate within confined spaces where an operator's human hand would normally be greatly limited. This dexterity is further strengthened through motion scaling, which translates large motions by the operator into diminutive actions of the robotic end effector. An example of this is the Da Vinci System which is coupled to the EndoWrist end effector tool.

Nevertheless, these systems also have some drawbacks such as the high cost of the surgery itself and the lack of tactile or haptic feedback. This means that as the surgeon is performing the procedures outside the patient's body, he/she cannot feel the resistance of the human tissue's when cutting. Therefore, one can risk damaging healthy tissues if force is not controlled or, when sewing, one can exert an exaggerated force and break the thread.

In this project, a new system is created based on the UR5 robot (Universal Robots) and an EndoWrist needle to mimic the behavior of the Da Vinci System and implement some improvements regarding the maneuverability and haptic feedback performance.

1. Introduction:

1.1. *Objectives*

Robot-assisted surgery is a minimally invasive surgical technique that supposed a revolution in the medicine's field. However, its great complexity requires constant development of methods which improve and facilitate the procedures carried out during interventions. In the last years, the improvement of maneuverability and haptic feedback has become quite significant in the advancement of this instrumentation.

The world-leader surgical robot system is the *Da Vinci System* (Surgical Intuitive, Inc.). This modern groundbreaking technology enables a robotic arm to precisely translate the movements performed by a surgeon through three main elements: a Surgeon's Console, a Surgical Cart and a Vision System. This project is focused on translating the movements from the Surgeon's Console to the end-effectors with maximum precision and with no delay.

The first objective of this project is to analyze the maneuverability performance of minimally invasive surgical robots, specifically the one implemented in the *Da Vinci* system: the *EndoWrist* end effector tool. This performance involves the speed at which the movement is transmitted from the motors to the end effector (the tweezers), the precision with which the robotic arm tries to mimic the surgeon's maneuvers, the stability of the movement and the tremor filtering.

This study is made to comprehend the functioning of the *Da Vinci System* to be able to reproduce the maneuverability performance, by means of the UR5 (*Universal Robots*) and a set of end-effectors.

To accomplish the main goal, one must get acquainted with the functioning of the surgical robotic systems regarding the hardware elements that make possible the reproduction of the movement as well as the software interface that reads and transmits the information.

An in-deep study of the different software frameworks or robotics middleware such as the *RoboDK, Arduino* or *Robotic Operating System (ROS)* will be examined. Furthermore, different robotic arms will be examined. At the end, one will assess which is the most suitable configuration (of both software and hardware components) that allows a better performance in terms of maneuverability.

The final purpose of this project is to design a haptic pen tool capable of mimicking the surgeon movements. This pen will be designed with some extra capabilities to improve both the maneuverability and the haptic feedback.

Taking all this into account, one will evaluate the capabilities of UR5 robot attached to the *EndoWrist* by a designed 3D printed piece and compare it to the motion capabilities accomplished by current surgical robots such as *Da Vinci Robotic System*. Finally, an economic study on the cost of the instrumentation needed will have to be fulfilled.

To sum up, the following list exemplifies the different aims this project aspires to accomplish:

- Familiarizing with the minimally invasive surgery robots.

- Learning about the performance and interface necessary for this purpose: *Robotic Operating System (ROS)*, *Arduino IDE* and *RoboDK*.

- Comparative study of the capabilities among different surgical robots currently used in the market.

- An exhaustive analysis of the hardware components and software environments that can be implemented in these technologies.

- Development of the programming code with *Arduino* and *RoboDK* that transmits the RPY angles from the computer to the servomotors (end effector tool) and vice versa.

- Development of the programming code: advancement of the *EndoWrist* in the direction of motion and tweezer's opening and closing.

- Mechanically assembly of the arm and the end-effector with the 3D printing.

- Final validation and evaluation of its performance in the laboratory. If conditions are met, recording of the interlocking of the servomotors with the *EndoWrist* gears and overall performance of the system.

1.2. State of the art of robotic arm assisted surgery

A procedure is considered to be surgical when it involves cutting a patient's tissues or closing a previously sustained wound. With the passage of time, many revolutions have occurred in this field such as the introduction of anesthesia in the 19th century, the first successful organ transplantations during the 20th century or the arrival of robotic surgery at the end of the 20th century as well. Although nowadays robotics is widely and routinely used, its entrance in the field of medicine has been slow and progressive.

Robotic surgery, or robot-assisted surgery, is a still emerging technology that allows minimally invasive procedures. This generates a great interest among health professionals because it means, fundamentally, shorter hospitalization and faster recovery of the patients. On top of that, there is a reduced risk of infection, less blood loss and less scarring.

Authors often differ in the definition of the first robotic prototype as the way we know it. Nevertheless, for most of them, it is considered to be the PUMA (Programmable Universal Machine for Assembly) 560 robotic system, which was employed in 1985 in a neurosurgical biopsy. This served as a starting point for many companies and universities to develop robotic systems such as PROBOT, specialized in transurethral resection of the prostate; ROBODOC for hip replacement surgeries, the robotic arm AESOP (Automated Endoscopic System for Optimal Positioning) controlled by voice commands to manipulate the endoscopic camera and so on. Later modifications led to the development of two recognized and rival systems: Da Vinci and the Zeus System, which are similar in their capabilities but different in their approach to robotic surgery.

The Zeus (Computer Motion, Inc., Goleta, Ca) is a three-armed platform that makes use of the AESOP camera: one arm holds the voice-controlled camera and the other two (controlled by the surgeon) are used to hold the surgical instruments. It has two separate hubs: the patient side where the procedure is done and the surgeon side controlling the first. It received the FDA approval for limited used in 2001. In the Zeus System (see Figure 1), both the monitor and handles are ergonomically positioned to maximize dexterity and allow complete visualization. The system allows the articulation of the end-effector through 7 degrees of freedom (DOF).

There are three main types of robotic systems currently in use in the surgical field:

1. **Active systems***:* the robot essentially works autonomously, or undertakes pre-programmed tasks, under the supervision of the surgeon. These systems can recognize the changes in the environment and organize its duties accordingly. An example of an active system is the *ROBODOC*.
2. **Semi-active systems***:* the robot's total autonomy is combined with a surgeon-driven element. *Neuromate* is an image-guided robotic system used in stereotactic surgery [6]
3. **Master-slave systems** lack of any pre-programmed or autonomous element. They allow the surgeon to directly telemanipulate the robot from a remotely placed command center. In this situation, the surgeon's hand movements are transmitted to the surgical end-effector instruments. *Zeus* and *Da Vinci* were the forerunners in the master-slave category.

Master-slave systems are also known as passive robots since it is the doctor who provides the motion inputs. The control of the system is achieved by using these inputs in its control algorithm during surgery. One of its most outstanding advantages is that it scales the motion received from the master system to increase the sensitivity of the slave system. In addition, it can have six or more DOF so the surgeon can enter inputs not only from his/her hands but also from fingers and elbow. Hence, flexibility increases. One of the most difficult challenges is to keep the hands steady during the entire surgical procedure. This problem can be improved by filtering the tremor with the master

2. Da vinci Surgical Robot:

2.1. Da Vinci Surgical System research

The Da Vinci system is a sophisticated robotic platform designed to expand the surgeon's capabilities in minimally invasive option for major surgery. The first prototype was introduced in 2000 by Intuitive Surgical and it was approved by the FDA at the same year. During these two decades, medical institutions have been evaluating the clinical and economic benefits of the robot – there are now more than 21.000 reviewed published articles that support the safety, efficacy, and benefits of Da Vinci surgical systems. In fact, the single port Da Vinci platform is now in use in more than 40 centers.

The complexity of this kind of technology, both electrically and mechanically, requires an environment far from the traditional operating room (OR). As a matter of fact, all the personnel present in the OR (nurses, technicians, or surgeons) must be trained to manage the equipment. This way, problems that emerge during the surgery can be easily identified and solved by any member of the team.

From its introduction in the market, there has been many generations of this surgical system: the Standard Da Vinci (which was introduced at 2000 although its commercialization stopped seven years later) only had three robotic arms, whereas S model (introduced at 2006), Si model (2011), Xi model (presented in April of 2014), X model (approved on April of 2017) and Single Port (2018) all have four arms.

Da Vinci Systems allow the introduction of miniaturized wristed instruments and a high-definition 3D camera. The system cannot be programmed nor can make decisions on its own, it requires that every surgical maneuver is performed with direct input from the surgeon. Even though every generation adds further improvements, all four-armed systems.

2.2. Components of Da Vinci

1. Surgeon's Console: from this element the surgeon can manipulate the arms of the robot. The individual grabs two handles which position and orientation trigger highly sensitive motor sensors that transfer the information to the end-effector tool. In addition, some models incorporate foot pedals to control electrocautery, camera focus and instrument/camera arm clutches.

2. Surgical Cart provides 3 degrees of freedom (pitch, yaw, insertion). Attached to the robot arm is the surgical instrument, the tip of which is a mechanical cable-driven wrist (EndoWrist) which adds 4 more degrees of freedom (internal pitch, internal yaw, rotation, and grip).

   Intuitive Surgical patented EndoWrist instruments which are designed to provide natural dexterity through several accessories such as scissors, graspers, needle holders, monopolar cautery instruments, clip appliers, scalpers, etc. All these tools provide the total 7 DOF, 90º of articulation, intuitive motion, fingertip control, motion scaling and tremor reduction. The wrist-like movement, responsiveness and robotic control afforded by the Da Vinci and its exclusive EndoWrist instruments provide surgeons fluid ambidexterity and unparalleled precision.

   The patient cart rolls on wheels and is moved and positioned over the patient. The robotic arms are designed like the human arm with a shoulder, an elbow and a wrist. The patient cart is connected through wires to the surgeon's console but before positioning the cart, It must be covered by an additional sterile coat to prevent coming into contact with non-sterile objects.

3. A Vision System controls the whole network that resides in the surgeon's console. It is an image processing computer that generates a 3-dimensional image with depth of field. The 3D camera is attached to the 4th robotic arm, which magnifies the surgical site. The vision cart consists of a left eye camera control unit, a right eye camera control unit, a light source, video synchronizer and focus controller, assistant monitors…

We must consider that in the operating room there is the surgeon working from the computer console and the surgical team supervising the robot at the patient's bedside.as shown in the following image:

3. Aims and objectives:

3.1. Aims:

1. Enhance Surgical Precision
    - Superior 3D visualization to see complex details of the surgical area.
    - Leverage robotic technology that mimics the human wrist movements for precision.
    - Higher stability and consistency to ensure steady and precise movements.

2. Expand Surgical Capabilities
    - Improved ergonomics and control during surgery can lead to a smoother workflow.
    - Enhanced dexterity for precise, delicate manoeuvres that is traditionally difficult.
    - Reduced blood loss during surgery, contributing to safer procedures.

3. Promote Minimally Invasive Surgery
    - Reduced trauma to surrounding tissues leading to more gentle surgical experience.
    - Less postoperative pain experienced by patients, reducing the need of painkillers.
    - Minimised scarring leading it to have visible scars.

4. Advance Medical Innovation in Egypt
    - Improved healthcare infrastructure by investing in modern medical facilities.
    - Attraction of skilled surgeons and medical professionals to Egypt, enhancing the local talent pool.
    - Reduction of hospital stay duration, due to shorter recovery times, freeing up hospital resources and improving bed availability.
    - Da Vinci robot allows a wider range of complex procedures to be performed, making specialized surgeries more accessible to patients across the country.

3.2. Objectives:

1. OAK-D Lite camera:
    - Develop detection and segmentation YOLLOv8 models for 3D visualisation of wounds.
    - Enhancement of surgical precision

2. Joystick controller:
    - Develop an electrical control system for the surgeon to manually stitch the wounds precisely.
    - Expansion of surgical capabilities.

3. Path Planning:
    - Develop an algorithm for autonomous stitching of wounds without any human interact.
    - Promotion of minimally invasive surgeries.

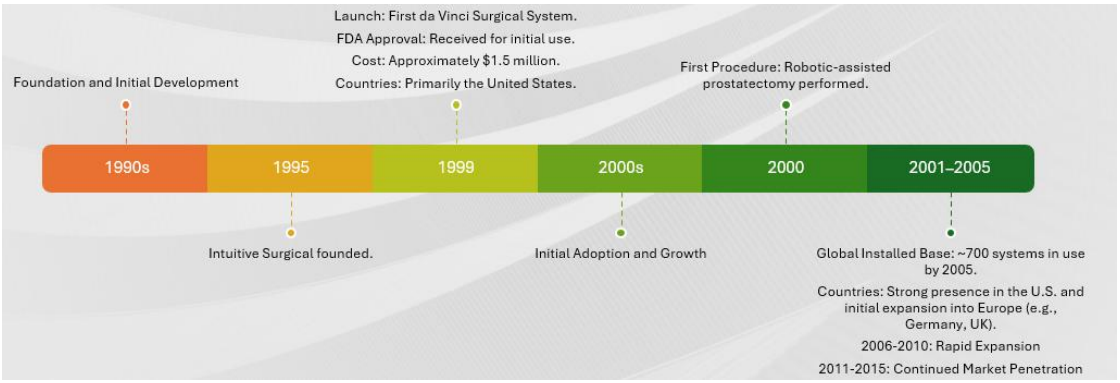
4. Market study:

The Surgical Robot was introduced to Egypt's healthcare system in the early 2000s, with private hospitals in Cairo and Alexandria being the first to adopt it.

It has been primarily used for urological, gynecological, and cardiothoracic surgeries, providing precision and shorter recovery times for patients. While its high cost has limited its widespread adoption, leading private and some public hospitals have invested in the technology.

Challenges include the high price of the system and the need for specialized training, but Egypt's healthcare modernization efforts and medical tourism potential have supported its growth.

The system's use is expanding, particularly in oncology and complex surgeries, with more surgeons receiving training in robotic-assisted techniques. The future of the Da Vinci system in Egypt looks promising, with ongoing investments in healthcare infrastructure.



5. Brief History of Robotic Surgery

The First Robotic Surgery (1985)

On March 3, 1985, the Unimate Puma 200 was used to perform a neurosurgical biopsy with unprecedented precision at the Memorial Medical Center in Long Beach, California. This marked the first time a robot was employed in a surgical procedure. The robot was programmed to assist in positioning a needle for a brain biopsy, reducing human error and improving accuracy. While the Unimate was originally designed for industrial automation, its adaptation for medical use demonstrated the potential of robotics in surgery.

Evolution of Robotic Surgery

Following the success of the Unimate in 1985, the field of robotic surgery began to evolve rapidly. Key developments include:

PROBOT (1988): Developed at Imperial College London, this robot was specifically designed for prostate surgery, showcasing the potential for robots to perform specialized tasks.

ROBODOC (1992): Introduced for orthopedic surgeries, particularly hip replacements, ROBODOC was the first robot approved by the FDA for surgical use.

da Vinci Surgical System (2000): Developed by Intuitive Surgical, the da Vinci system revolutionized robotic surgery. It provided surgeons with enhanced dexterity, precision, and 3D visualization, making it a cornerstone of modern robotic-assisted surgery. The da Vinci system remains the most widely used robotic surgical platform today.

Impact of Robotic Surgery

Robotic surgery has transformed the medical field by enabling minimally invasive procedures, reducing patient recovery times, and improving surgical outcomes. While the Unimate Puma 200 was a rudimentary beginning, it laid the groundwork for the sophisticated robotic systems used in hospitals worldwide today. The integration of artificial intelligence and machine learning into robotic surgery promises even greater advancements in the future.

In summary, the first robotic surgery in 1985 using the Unimate Puma 200 marked the dawn of a new era in medicine, paving the way for the development of advanced robotic systems that continue to redefine surgical practices.

Surgical Robots Classes

Surgical robots can be empirically classified into two classes:

- A fully autonomous class: pre-schedules surgeries and uses CT scans to make full plans of the surgery ahead (ROBODOC).
- A slave-master(teleoperated) class: follow the surgeons' movements in a slave-master style (Da Vinci).

6. Our Prototype Project Overview:

6.1. Project description and operation:

The Robotic Surgeon consists of two main stations:

- Control station (master station):
  It consists of a monitor and an analog controller.
- Arm station (slave station):
  It consists of the robotic arm and a camera tracking the end effector of the robot.

The Robotic Surgeon is a graduation project mimicking the already existing Da Vinci surgical robot used for minimally invasive surgeries as shown in figure 1.1.

Figure 1.1 Davonco SurgicalRobot

To operate the robot a specialized surgeon is seated at the control station. The surgeon uses the controller to control the end effector position and grip.

A camera tracks the end effector, and outputs live feed along with sample useful data (wound depth, stitch placement, etc.).

The axes of the controller are mapped to the surgical workspace. An inverse kinematic solver is responsible for calculating the servo angles for the given controller input.

6.2. Design, Simulation & Hardware:

Hardware:

I-End effector tool

The end effector used in this project is a recycled "Da Vinci endowrist, as illustrated in ***Figure 2.1.*** The wrist has four separate pulleys which we reverse engineered to control roll, pitch, yaw, and jaw grip.

Figure 2.1 (Da Vinci endowrist)

II-Controller

The controller of choice was a ps4 controller shown in ***Figure 2.2*** . At a reasonable price point, the ps4 controller provides 6 independent analog sensors. It also has DualShock motors for haptic feedback.

Figure 2.2 (DualShock controller)

II-Microcontroller

The controller of choice was the Raspberry Pi 4 model B 8GB shown in Figure 2.3. At a reasonable price point, the Raspberry Pi provides sufficient computing power, varied peripherals, and ROS compatibility.

Figure 2.3 (raspberry pi 4)

III-Servo Driver

A basic servo driver PCA9685 is used to control the movement of the endowrist servos shown in Figure 2.4.

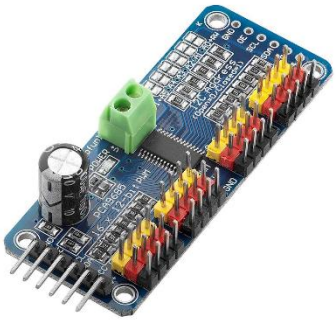


Figure 2.4 (PCA9685)

IV-Camera

The camera used is the Oak-D Lite Robotics Camera - Auto Focus.

Figure 2.5 (OAK D stereo cam)

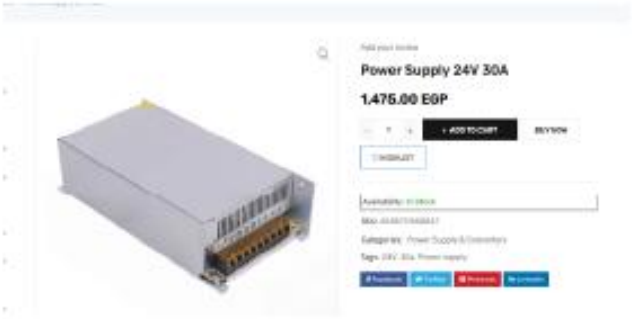
PCB Design:

PCB Design and Implementation This paragraph focuses on the Printed Circuit Board (PCB) aspect of the hardware. The following subtopics are covered:

Components

This subsection provides an overview of the key components used in the hardware design, along with their functions:

1. Power Supply 24V, 30A: A 24V DC power supply capable of delivering a maximum current of 30A. Used to provide stable power to a system that requires a relatively high current at a fixed voltage, as shown in figure.



(a) Power Supply

2. LM339 Quad Differential Comparator: A four-channel comparator IC that is used for comparing two voltages and producing a high or low output. It is commonly used in circuits for monitoring voltage levels, providing analog-to-digital conversion, or controlling other devices based on voltage conditions.

3. Buck XL4016: A DC-DC buck converter (step-down voltage regulator) capable of converting a higher voltage (typically from a 24V source) down to a lower voltage (e.g., 5V or 12V) with efficient power regulation. The XL4016 is often used in applications requiring a stable output voltage for powering low-voltage circuits.as shown in figure2

(b) Buck Converter

4. MCT2EM Optocoupler: A phototransistor optocoupler used to electrically isolate different sections of a circuit. It converts an electrical signal into light, which is then received by a phototransistor, providing isolation and protecting sensitive components from high voltages or electrical noise.

5. Resistance 4.7kΩ: A resistor with a resistance value of 4.7kΩ. Resistors are used to limit current flow, set voltage levels, and filter signals. A 4.7kΩ resistor is commonly used in signal processing or biasing applications.

6. Emergency Switch: A switch designed to shut down or disable a system in case of an emergency. This is typically used for safety purposes to quickly cut off power or stop a potentially dangerous process in various industrial or consumer devices.

7. Power Terminal Block (2-PIN and 3-PIN): Terminal blocks are used to connect electrical wires securely. A 2-pin terminal block can connect two wires, while a 3-pin block can connect three wires. These are commonly used in power supply and control circuits.

8. Fuses: A protective component designed to disconnect power from a circuit if the current exceeds a safe limit. Fuses help prevent damage to components or systems from overcurrent conditions.

9. Capacitor 10 μ F/25V: A 10 microfarad (μ F), 25-volt electrolytic capacitor used for smoothing or filtering applications in power supply circuits, decoupling signals, or stabilizing voltage levels. It stores electrical charge and helps maintain steady voltage in circuits.

10. Thyristor: A semiconductor device used to control power in electrical circuits. It acts like a switch, allowing current to flow when triggered and staying on until the power is turned off. Thyristors are used in high-power applications like motor control and lighting.as shown in figure 3

11. LM35CZ: A precision temperature sensor that outputs an analog voltage proportional to the temperature. It is typically used in systems that require accurate temperature monitoring, offering a direct temperature to-voltage relationship.as shown in figure 4

(a) THYEISTOR

(b) TEMPERATURE SENSOR

Figure 2: Illustrations of key components.

12. Zener Diode: A type of diode designed to allow current to flow in both directions, but with the ability to clamp the voltage across it to a specific value when in reverse breakdown. Zener diodes are commonly used for voltage regulation and overvoltage protection.

13. Fan: An electromechanical device used to circulate air, providing cooling for electronic circuits or systems that generate heat. Fans are essential in preventing overheating and maintaining optimal operating conditions.

14. AQZ102: A specific type of optocoupler (likely a solid-state relay), It is used to provide electrical isolation between high-voltage and low voltage sections of a circuit while still allowing control signals to pass through. Commonly used in control applications to switch AC or DC loads.

(a) AQZ102 (b) FAN

Figure 3: Illustrations of key components.

Protection

This section explains the protection mechanisms implemented on the PCB to ensure safe and reliable operation. It includes features like overcurrent protection, reverse polarity protection, and surge protection.

1. Crowbar circuit:

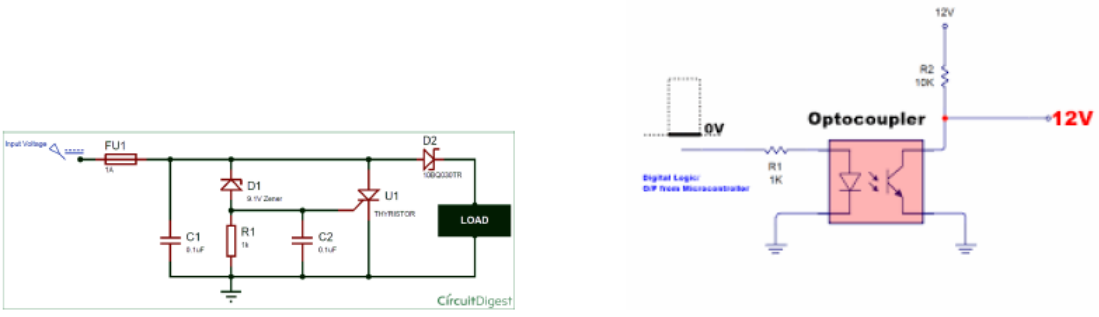
A crowbar circuit is a protection circuit used to prevent excessive voltage from damaging sensitive components. The term "crowbar" comes from its action of shorting the power supply to ground when the voltage exceeds a preset limit.

Operation: In a crowbar circuit, a voltage sensing device (like a Zener diode or comparator) monitors the output voltage. When the voltage exceeds a predetermined threshold, it triggers a thyristor (or SCR) to conduct, effectively shorting the power supply and causing a fuse to blow, thus protecting the load. Purpose: It provides over-voltage protection by quickly discharging excess voltage, typically used in power supplies or circuits with sensitive devices

2. Optocoupler circuit: An optocoupler (also known as an optoisolator):

is used to electrically isolate different sections of a circuit while allowing the transfer of signals through light. It is often used for safety and noise reduction in high-voltage circuits.

Operation: The circuit consists of an LED and a photodetector (like a phototransistor or photodiode) in a single package. The LED emits light when a signal is applied to it, and the photodetector receives this light and generates a corresponding electrical signal on the output side, thus isolating the input and output circuits. Purpose: It provides electrical isolation between high-voltage and low-voltage circuits, reducing the risk of electrical shock, noise, and ground loops. Common applications include signal isolation in microcontrollers, switching power supplies, and communication interfaces.



(a) CROWBAR (b) optocoupler

Figure 4: protection circuit.

PCB Design 1 Process:

The PCB1 is designed to manage and protect a system powered by a 24V, 30A supply. It is equipped with protection mechanisms for overheating and overcurrent conditions to ensure safe operation. Here is an overview of the key components and their roles:

Power Supply Input (24V, 30A): The PCB receives a 24V, 30A power input, providing sufficient power for the motor and other components.

Overheat and Overcurrent Protection: These protection features monitor the system’s temperature and current. If the values exceed preset thresholds, the circuit triggers protective actions like shutting down the power or activating a safety switch. Motor Output (24V, 25A): The PCB supplies power to the motor, ensuring the motor operates at a stable 24V with a maximum current of 25A.

Optocoupler for Signal Isolation: An optocoupler is used to isolate the signals between the Raspberry Pi and the motor, ensuring that electrical noise or spikes do not affect the Raspberry Pi’s control circuits. This provides safe communication between high-power and low-power sections of the system.

PCB Design 2 Process

This second PCB is designed to manage and protect a system powered by a 24V, 30A supply. It is equipped with protection mechanisms for overheating and overcurrent conditions to ensure safe operation. Here is an overview of the key components and their roles:

Power Supply Input (24V, 30A): The PCB receives a 24V, 30A power input, providing sufficient power for the motor and other components.

Overcurrent Protection: These protection features monitor the system’s temperature and current. If the values exceed preset thresholds, the circuit triggers protective actions like shutting down the power or activating a safety switch. Motor Output (8.4V, 3.5A): The PCB supplies power to the motor, ensuring the motor operates at a stable 8.4V with a maximum current of 3.5A.

Optocoupler for Signal Isolation: An optocoupler is used to isolate the signals between the Raspberry Pi and the motor, ensuring that electrical noise or spikes do not affect the Raspberry Pi’s control circuits. This provides safe communication between high-power and low-power sections of the system

PCB Design 3 Process

This third PCB is designed to manage and protect a system powered by a 24V, 30A supply. It is equipped with protection mechanisms for overheating and overcurrent conditions to ensure safe operation. Here’s an overview of the key components and their roles:

Power Supply Input (24V, 30A): The PCB receives a 24V, 30A power input, providing sufficient power for the motor and other components.

Overcurrent Protection: These protection features monitor the system’s temperature and current. If the values exceed preset thresholds, the circuit triggers protective actions like shutting down the power or activating a safety switch. Motor Output (12.6V, 8.3A): The PCB supplies power to the motor, ensuring the motor operates at a stable 8.4V with a maximum current of 3.5A.

Optocoupler for Signal Isolation: An optocoupler is used to isolate the signals between the Raspberry Pi and the motor, ensuring that electrical noise or spikes do not affect the Raspberry Pi’s control circuits. This provides safe communication between high-power and low-power sections of the system

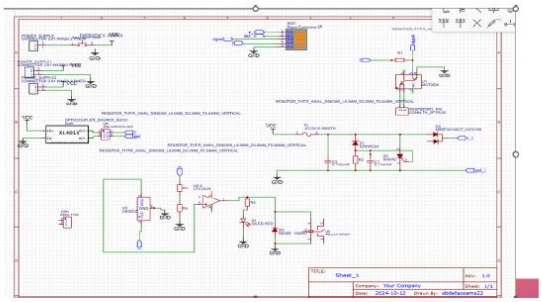
Fabrication

This section outlines the PCB fabrication process, covering: PCB3, ACID, copper thickness, and board layers.

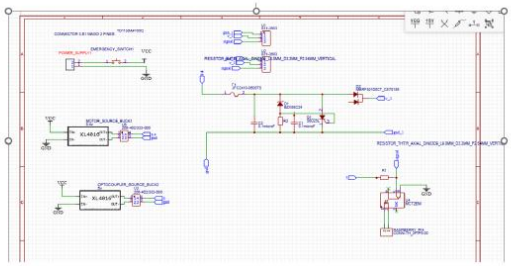
Simulation

This section discusses the simulation of the PCB design to verify functionality and ensure signal integrity. Tools like SPICE, Proteus, or dedicated PCB simulation software can be highlighted. Proteus, in particular, offers robust simulation capabilities, including microcontroller integration, making it ideal for testing complex circuits and embedded

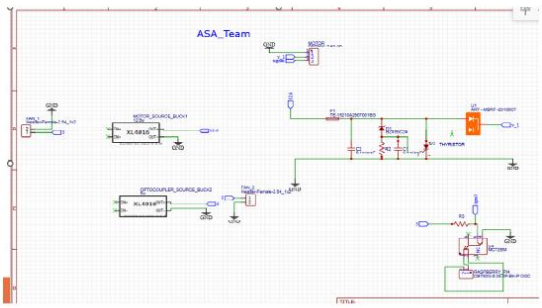
systems. These simulation tools help identify potential issues in the design, reduce the risk of failure, and optimize performance before fabrication.



(a) PCB1



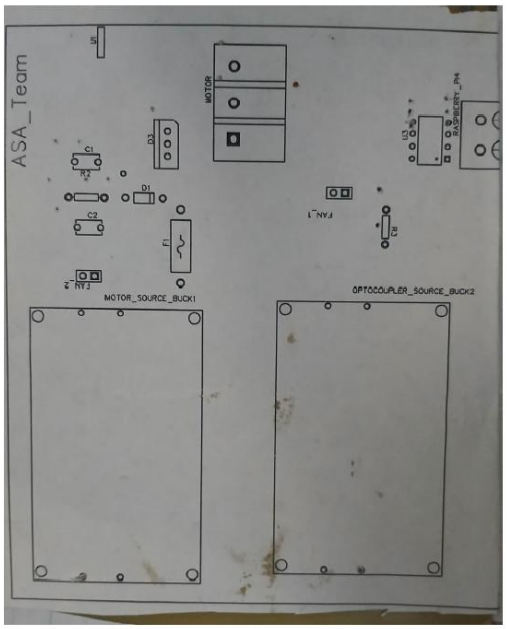
(b) pcb2



(c) pcb3

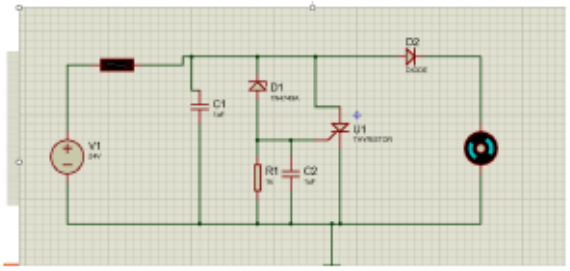
Figure 5: DESIGN OF PCB

(a) PCB3

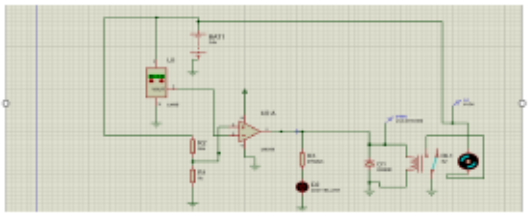


(b) pcb3

Figure 6: PCB3



(a) curent protection



(b) TEMPERATURE SENSOR

Figure 7: temperature protection

Redundancy:

This section focuses on the redundancy measures implemented in the PCB design to enhance reliability. Examples include duplicating critical traces, adding backup power paths, and designing fault tolerant circuits. For power supply redundancy, a 36V, 4A battery can be used as the primary power source for the system. In case of any power interruptions, a power bank can be employed to supply backup power to the Raspberry Pi. Additionally, a DC-DC boost converter can be utilized to step up the voltage and increase the current to meet the power requirements of the system. These measures ensure that the system remains operational even in the event of primary power failure, enhancing overall reliability and uptime.

(a) Battery

(b) Power bank

Figure 8: Redundancy

Robotic Arm Mechanical Design

Optimized Link Design

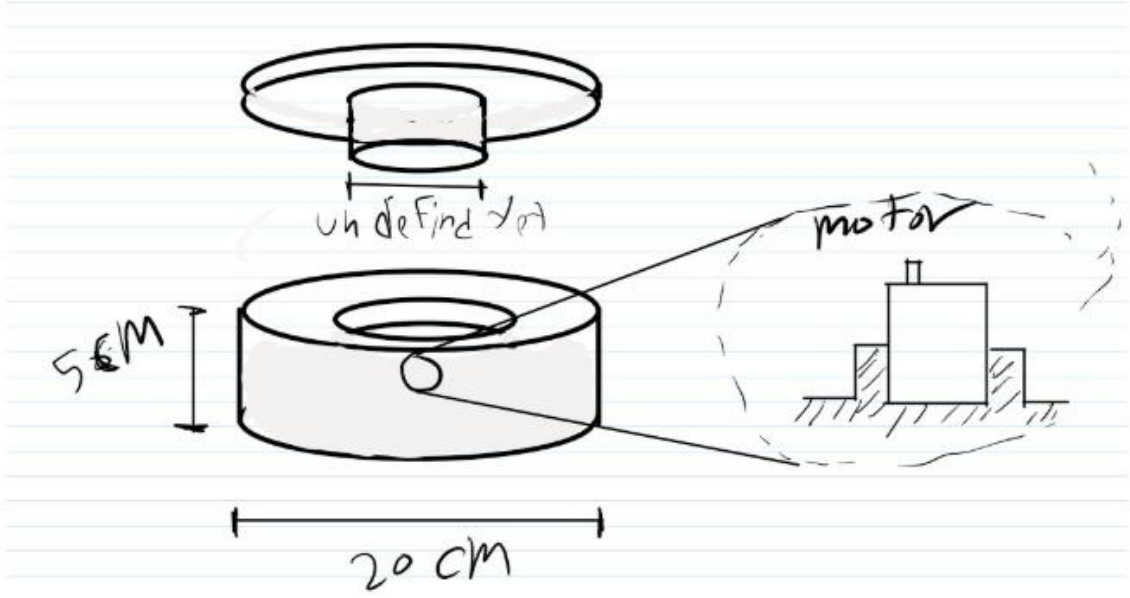
The lengths of the links in the medical robot were carefully selected based on the Da Vinci Surgical System's precision and dexterity:

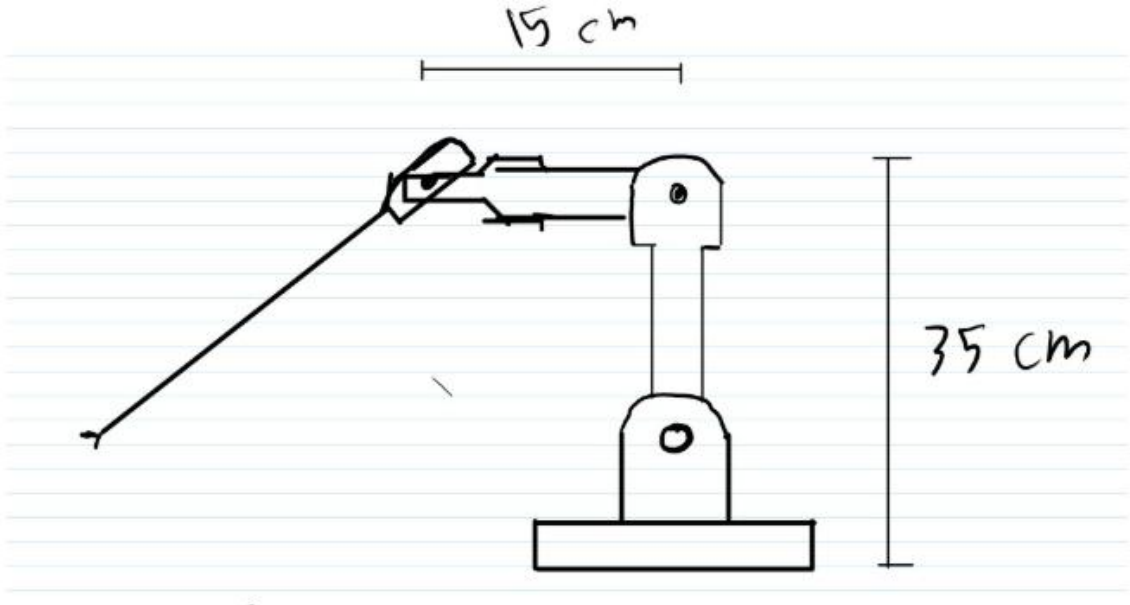
- **First Link (20 cm)**: Provides greater reach, covering a broader workspace, beneficial for procedures requiring extensive reach while maintaining stability and precision.
- **Second Link (15 cm)**: Balances between reach and control, enhancing the robot's ability to perform delicate tasks with accuracy.

These design choices offer a combination of reach, control, and stability, making the robot well-suited for intricate surgical or medical tasks.

Stage 1 (2D Modelling)

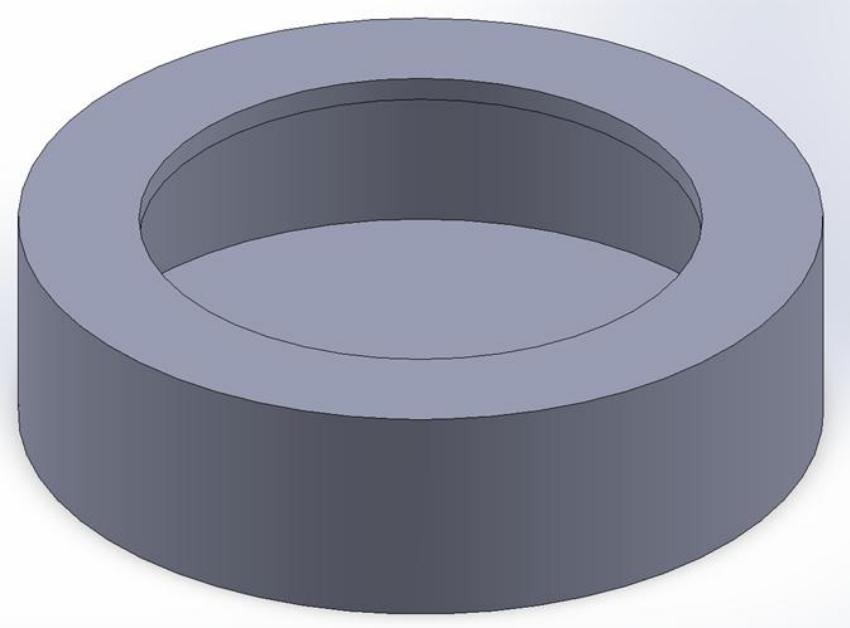
- Start by creating 2D sketches of the robotic arm.
- Include basic dimensions like arm lengths, joint locations, and base size.



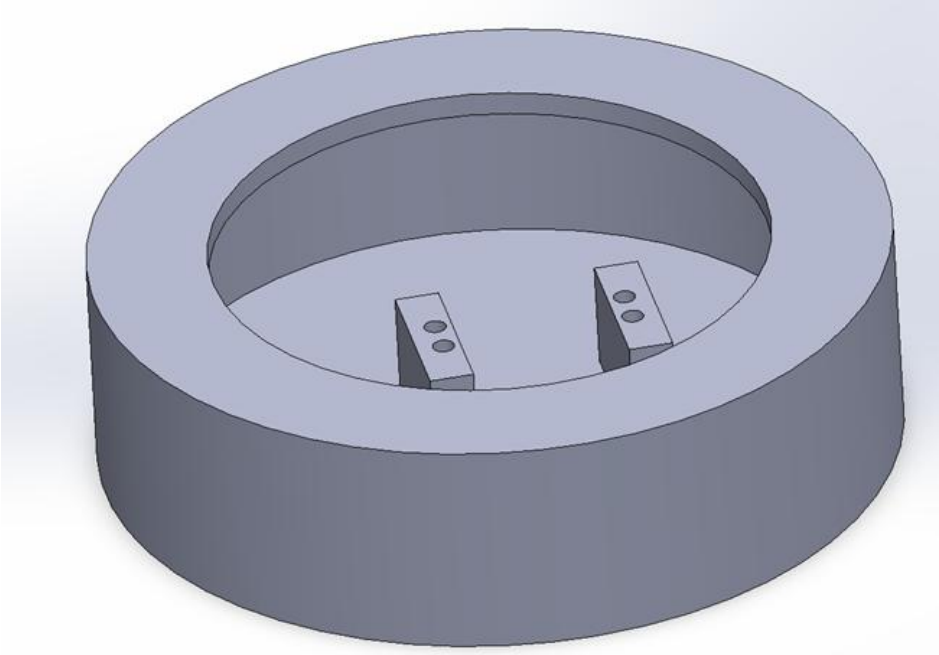


Stage 2 (3D Modelling)

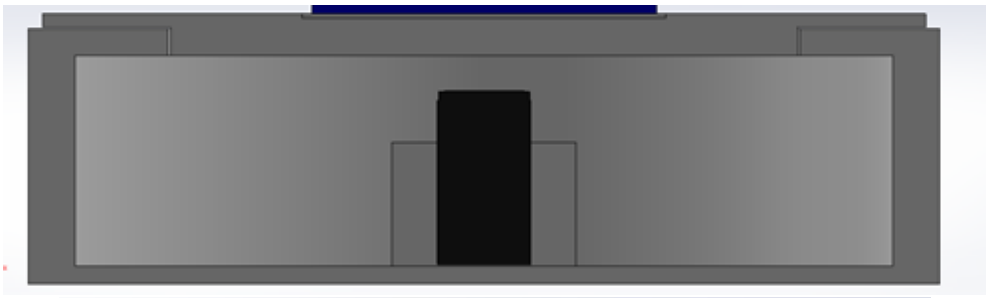
- Start by modelling the base of the arm.



- Add features like a motor mount for the first revolute joint.

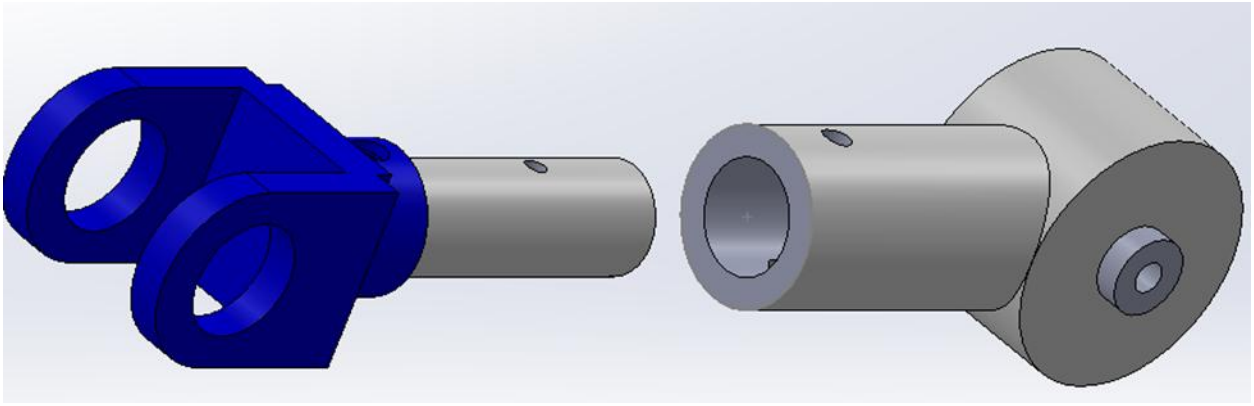


- Create a separate part for a rotating movement

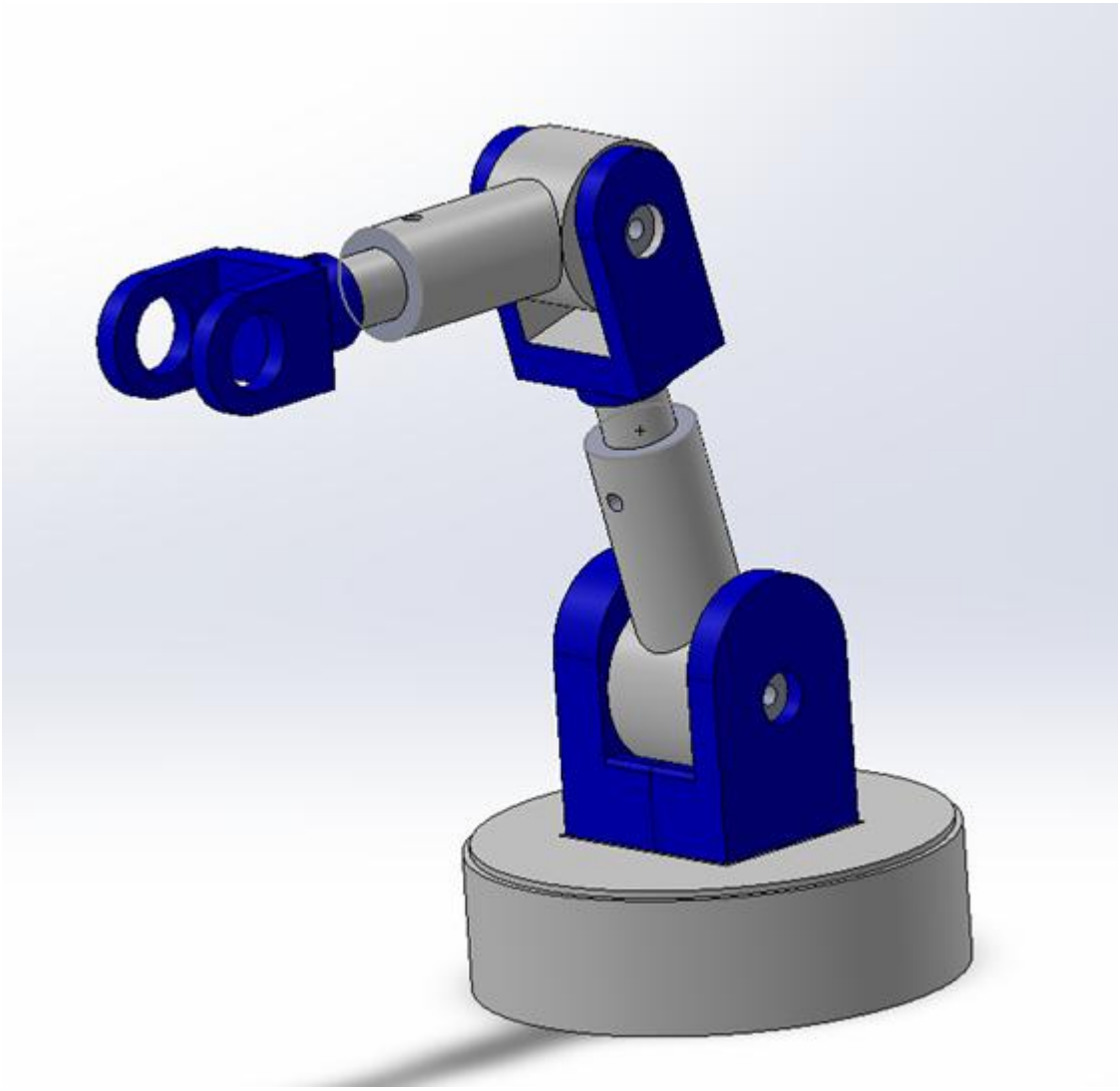


Links Design

- Design links as hollow beams(tubes).
- Add mounting holes or brackets to connect links to joints and motors.



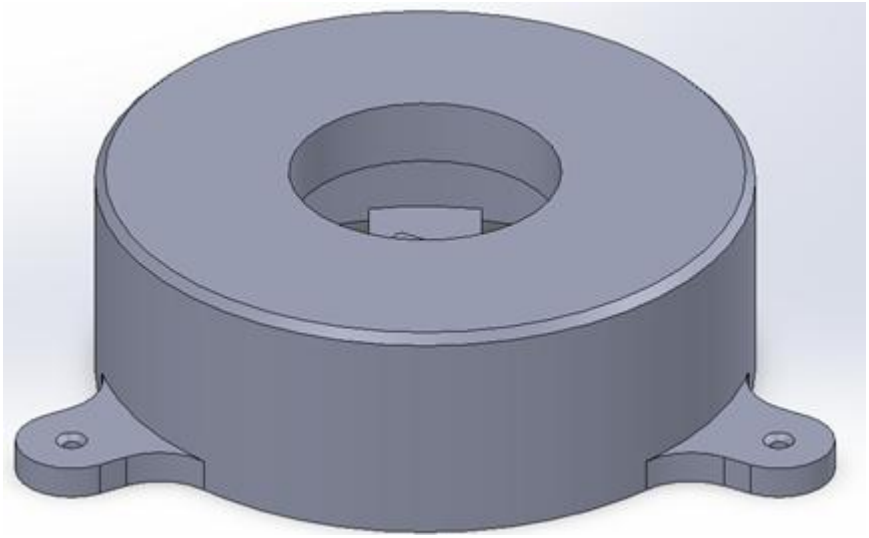
- Complete Assembly for joints and Base.

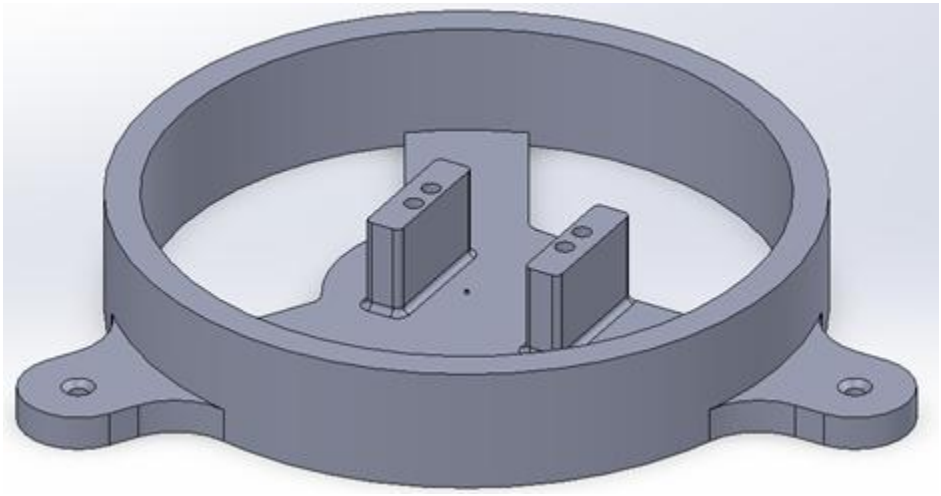


Stage 3 (primarily Enhancement)

Stationery Base

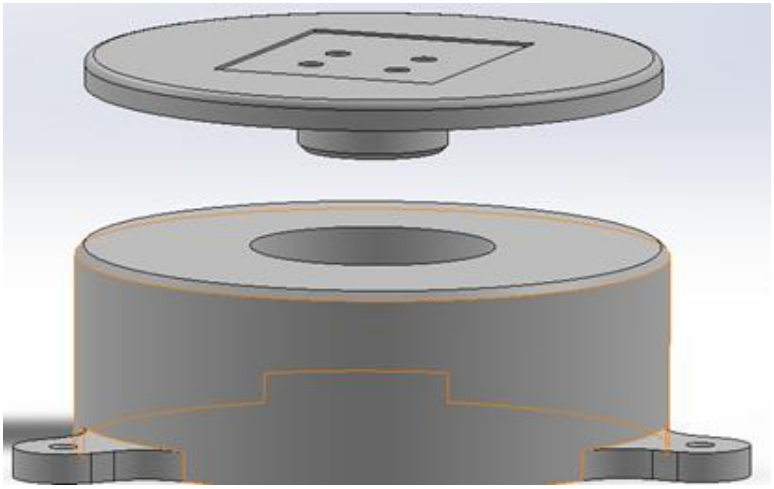
- Design a stable base capable of anchoring the arm securely.
- Add fillets to sharp edges (5-10 mm radius) for a polished look and safety.
- Include mounting holes for screws or bolts if it needs to attach to a table.
- Reducing material from the base due to cost efficiency.

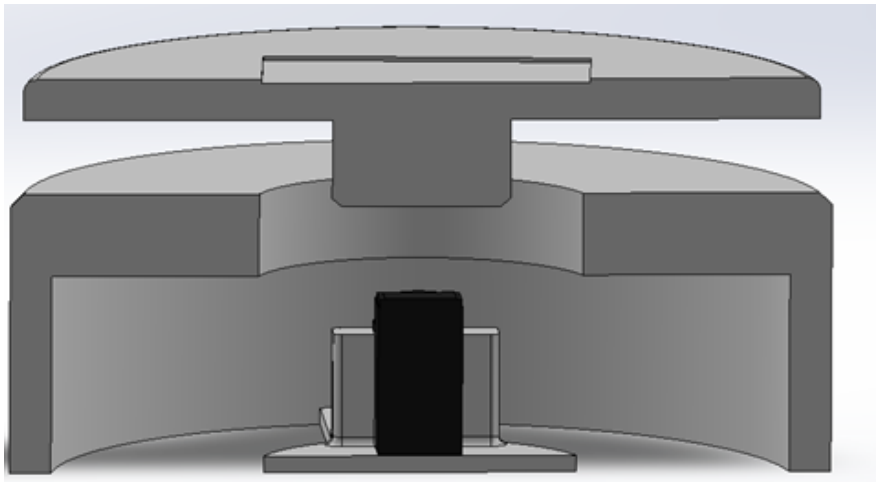




Rotating Plate

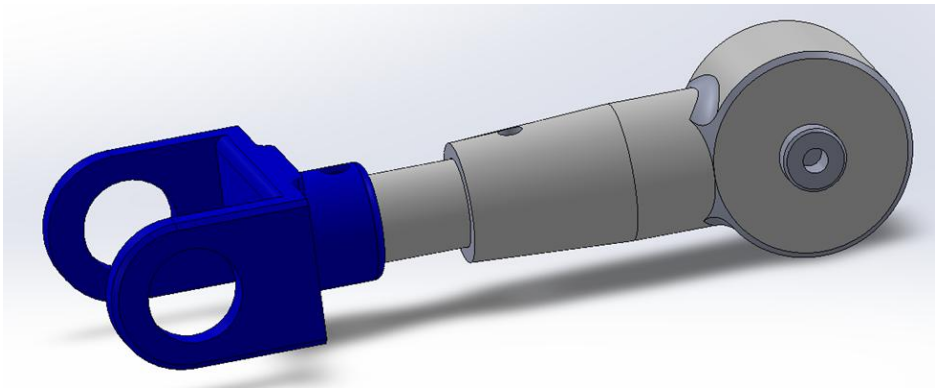
- Include mounting holes for screws to attach it to the shoulder Joint.
- Add fillets to sharp edges (5-10 mm radius) for a polished look and safety.



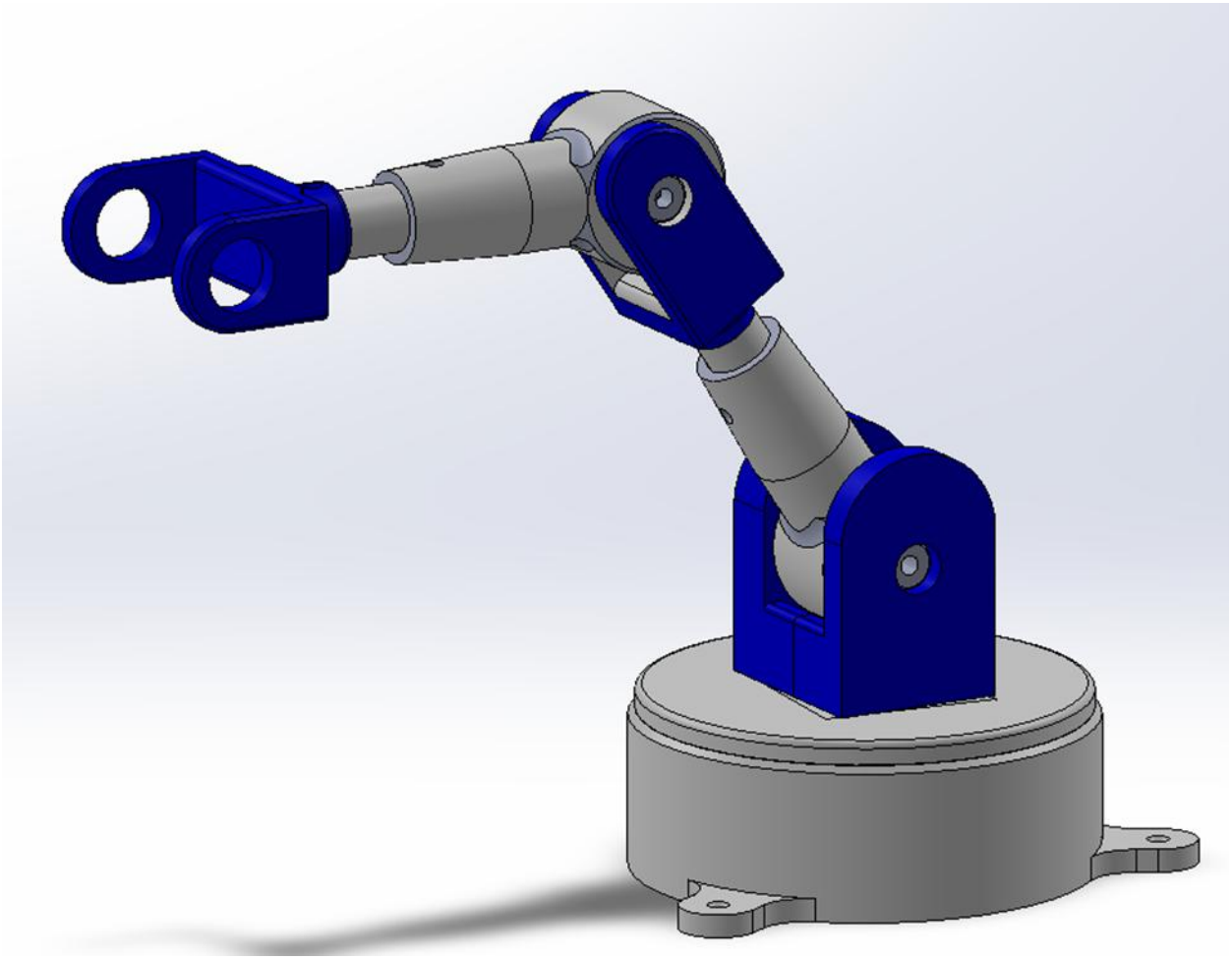


Links

- Apply fillets (2-5 mm) at corners where stress concentrations might occur, especially at link ends.
- Chamfer edges of bolt holes and mounting areas (1-2 mm) to make assembly easier and reduce wear on fasteners.



Assembly



Stage 4 (Torque calculations)

Step 1: Define Variables

1. Mass of each link (m): 200 g = 0.2 kg
2. Length of each link (L): 20 cm = 0.2 m
3. Mass at end effector (Meff): 600 g = 0.6 kg
4. Mass of wrist motor (Mwrist): 100 g = 0.1 kg
5. Mass of elbow motor (Melbow): 300 g = 0.3 kg

Assumption

- Center of mass is at the end of the link

Step 2: Calculate Forces

Sum the forces considering the additional masses at the respective points:

1. Force at the First Link Including Elbow Motor:

$F1=(m+Melbow)\cdot g=(0.2\ kg+0.3\ kg)\cdot 9.81\ m/s2=4.905N$

2. Force at the Second Link Including Wrist Motor:

$F2=(m+Mwrist)\cdot g=(0.2\ kg+0.1\ kg)\cdot 9.81\ m/s2=2.943N$

3. Force at the End Effector:

$F3=Meff\cdot g=0.6\ kg\cdot 9.81\ m/s2=5.886N$

Step 3: Calculate Torques

1. Torque at the First Joint (T1):

$T1=F1\cdot(L2)+F2\cdot L+F3\cdot 2L=4.905\cdot 0.2+2.943\cdot 0.4+5.886\cdot 0.5=5.1012\ Nm$

2. Torque at the Second Joint (T2):

$T2=F2\cdot(L2)+F3\cdot L=2.943\cdot 0.2+5.886\cdot 0.3=0.2943\ Nm+1.1772\ Nm=2.3544\ Nm$

3. Torque at the Third Joint (T3):

$T3=F3\cdot(L2)=5.886\cdot 0.1=0.5886\ Nm$

Summary

- Torque at Joint 1 (T_1): 5.1012 Nm
- Torque at Joint 2 (T_2): 2.3544 Nm
- Torque at Joint 3 (T_3): 0.5886 Nm

Converted to kg.cm (1 Nm = 10.197 kg.cm):

- Torque at Joint 1 (T_1): 52.01693 kg.cm
- Torque at Joint 2 (T_2): 24.007 kg.cm
- Torque at Joint 3 (T_3): 6.002 kg.cm

Factor of safety (FOS=2):

- Torque at Joint 1 (T_1): 104.03386 kg.cm
- Torque at Joint 2 (T_2): 48.014 kg.cm
- Torque at Joint 3 (T_3): 12.004 kg.cm

Negligible Inertia:

In the torque calculations for the medical robot, we have chosen to ignore inertia. This decision is based on the fact that the moments of inertia are relatively small due to the lightweight nature of the links and motors. Additionally, the robot's speed and acceleration are low, resulting in minimal angular acceleration. As a result, the impact of inertia on the overall torque is negligible and can be safely disregarded.

Stage 5 (Motors selection)

Base and wrist motor

- Voltage Range: 5–8.4V.
- Torque: 51 kg.cm @ 8.4V and 46 kg.cm @ 7.4V.
- Speed: 0.10s/60° @ 8.4V and 0.11s/60° @ 7.4V.
- Control Angle: 0–270° (500–2500 µSec).
- Dimensions: 40mm x 39mm x 20mm.
- Weight: 70g.

Shoulder motor

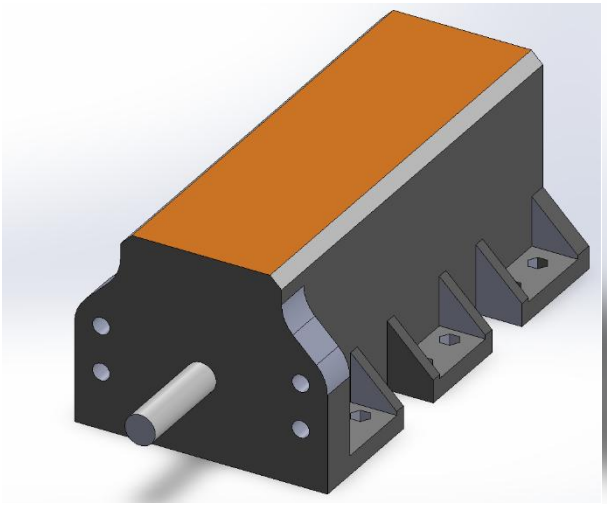
- Voltage: 16V–24V (Rated at 18V).
- Speed (18V): 0.16s/60°.
- Torque: Max Torque = 350 kg.cm and Rated Torque = 150 kg.cm.
- Angle Range: 0–360°.
- Input Signal: PWM (50µs–2200µs), 50–400 Hz.
- Size: 143mm (L) x 66mm (W) x 58mm (H).

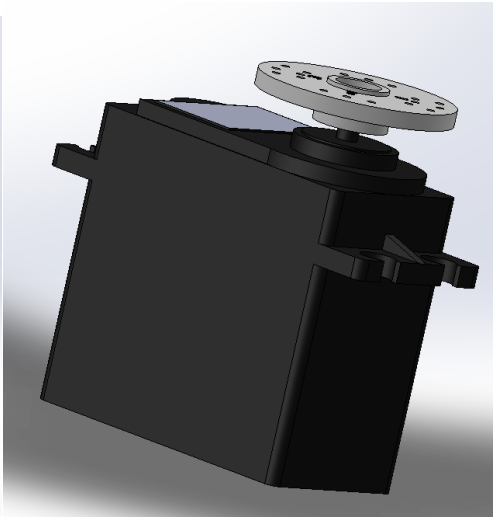
Elbow Motor

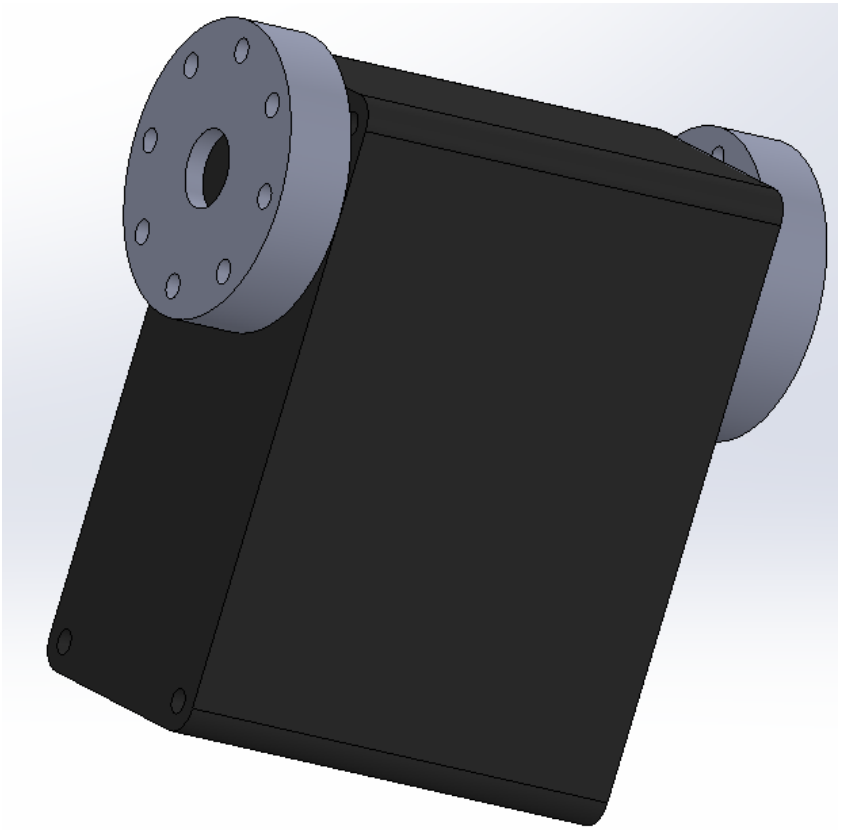
- Voltage Range: 9–12.6V.
- No-load Speed: 0.24s/60° @ 10V and 0.19s/60° @ 12.6V.
- Operating Angle: 270°
- Torque :150kg.cm @ 10V and 165kg.cm @ 12V.
- Stall Current (Locked): 7.4A @ 10V and 8.3A @ 12.6V

Stage 6 (Motors Assembly)

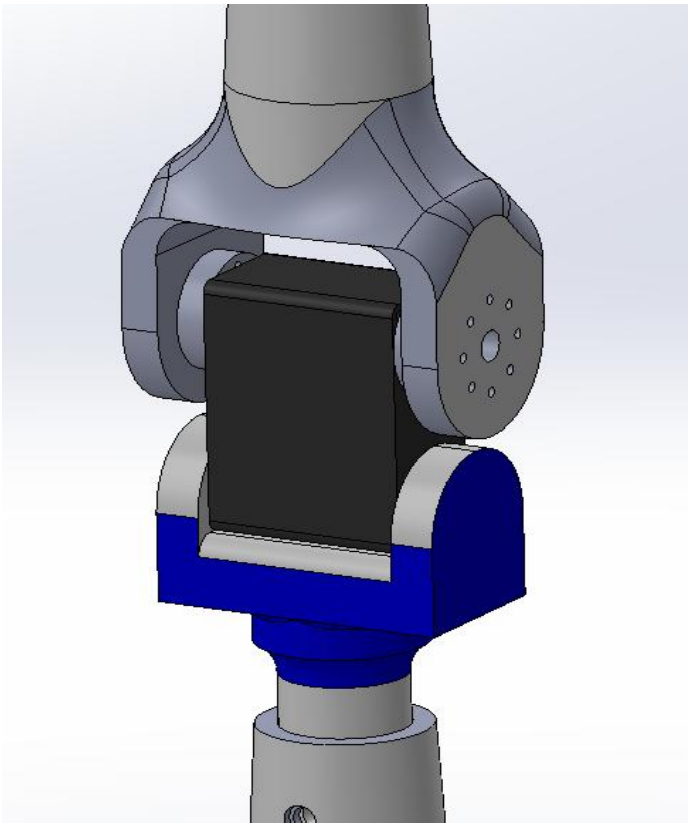
- Drawing a rough 3D model for the motors.



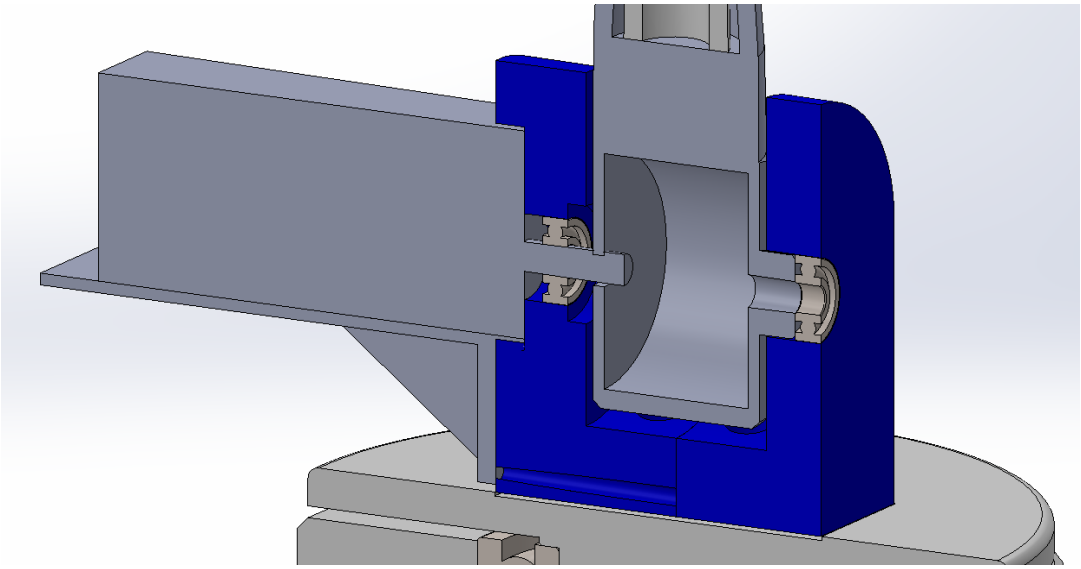


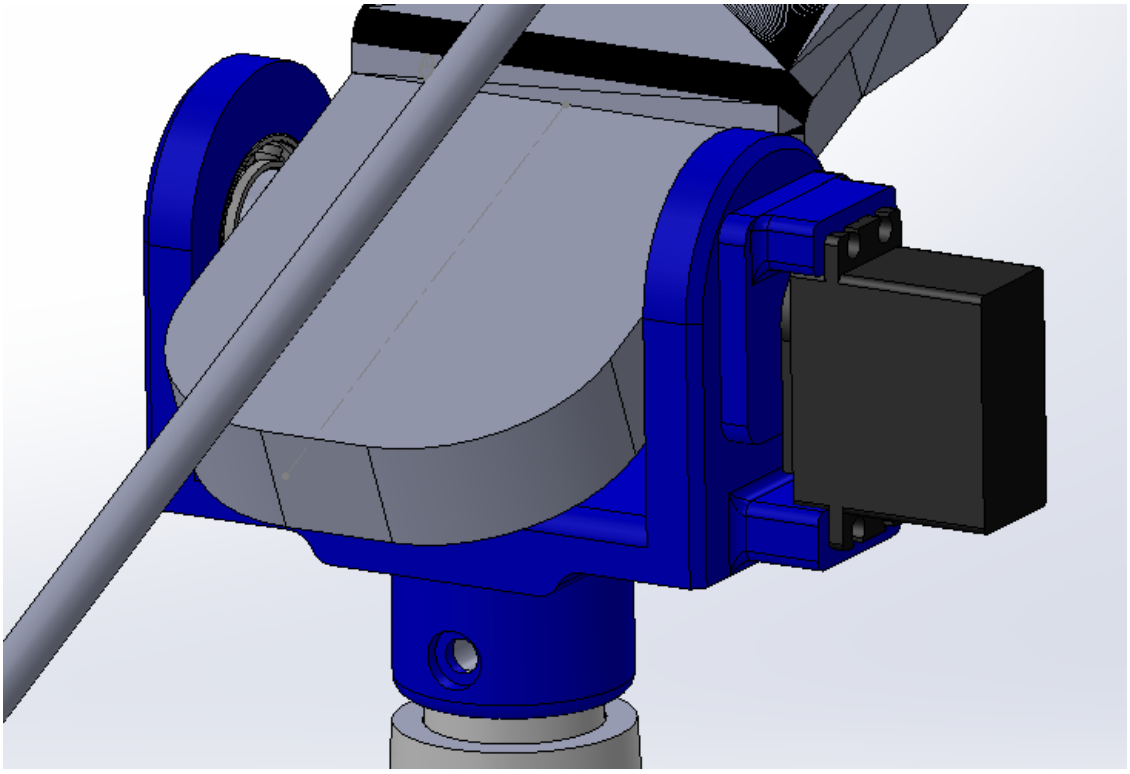


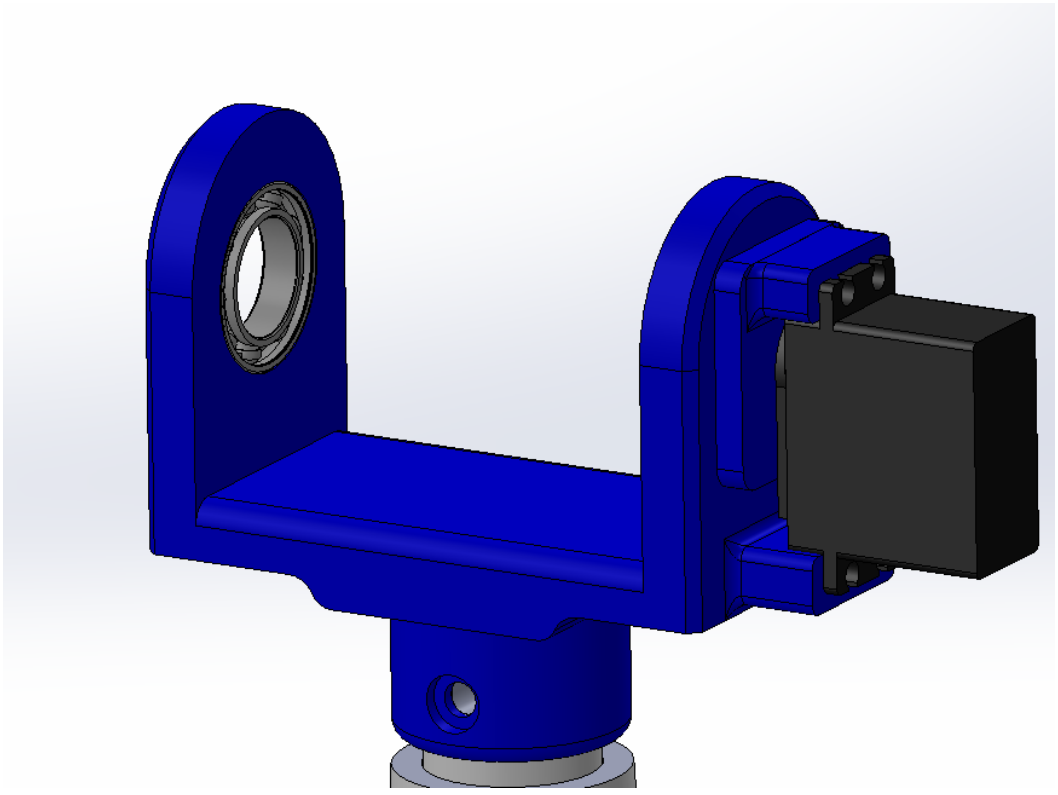
- Integrate motors into the 3D model by designing mounting brackets and ensuring proper alignment with joints.
- Adjusting links and Joint for fitting motors.



- Adding bearings for motors Shafts.







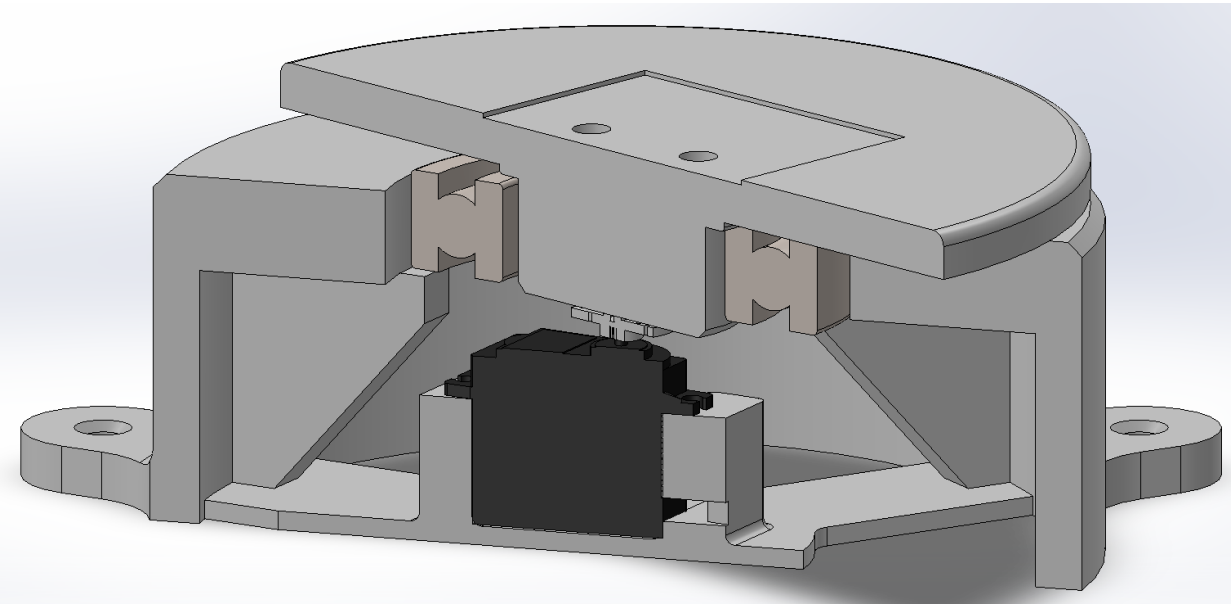
Stage 7 (Final Design Optimization)

- Enhance the Base

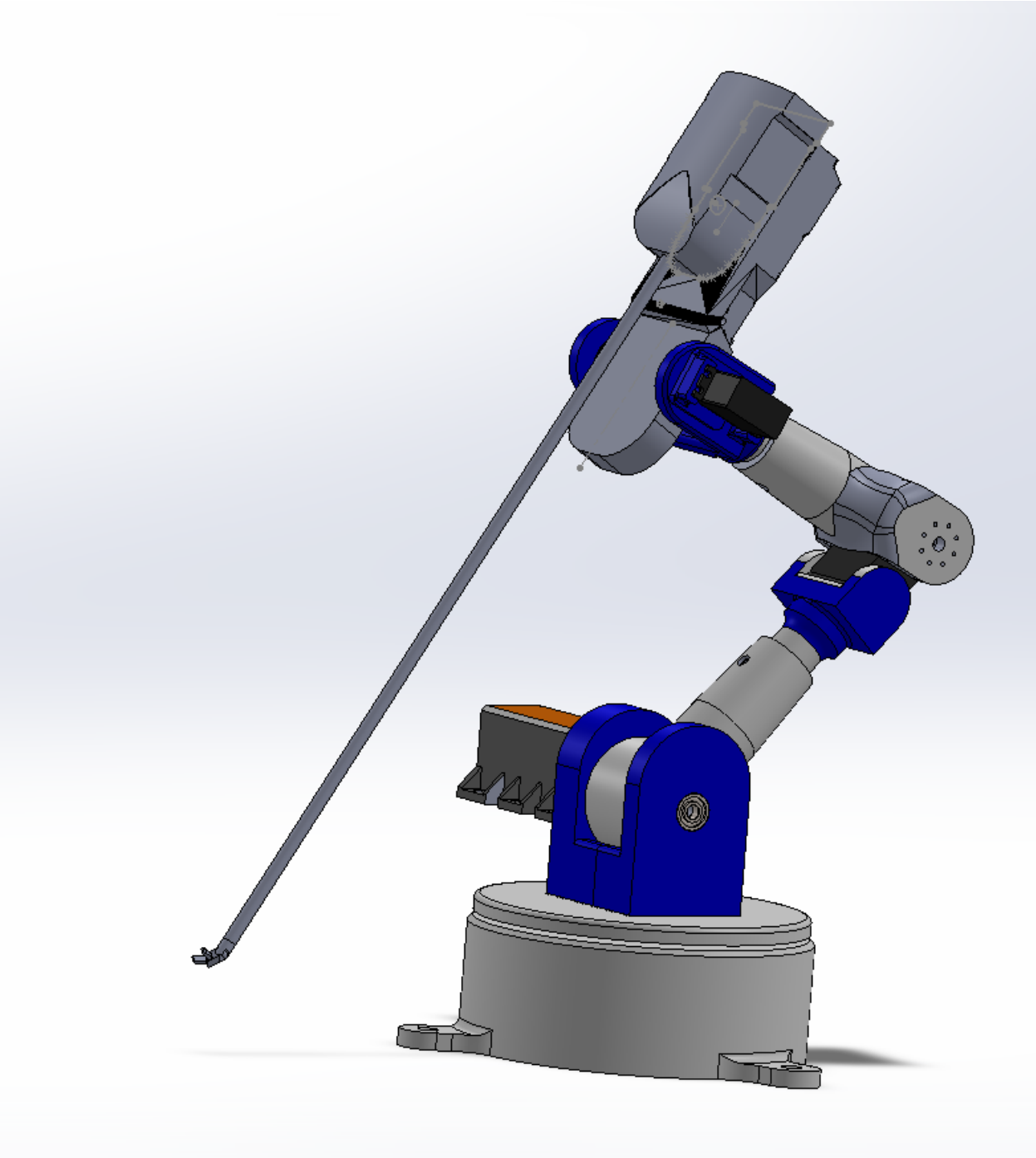
Bearing Selection:

For the base, we considered two types of bearings:

1. Normal Bearings: These include ball bearings and roller bearings, which can handle both axial and radial loads.
2. Thrust Bearings: These include thrust ball bearings and cylindrical roller thrust bearings, which are designed to handle only axial loads.



- **Final Design**

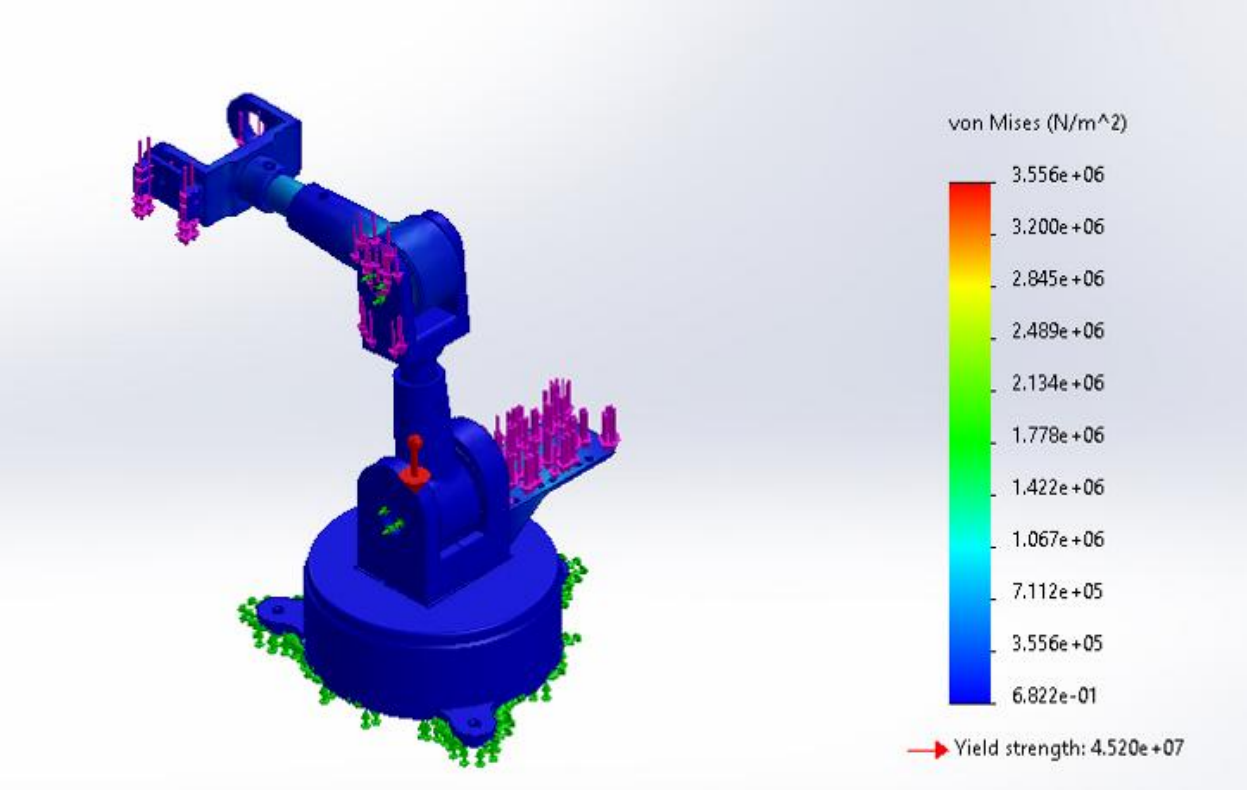


Primarily Stress Analysis

Study Results

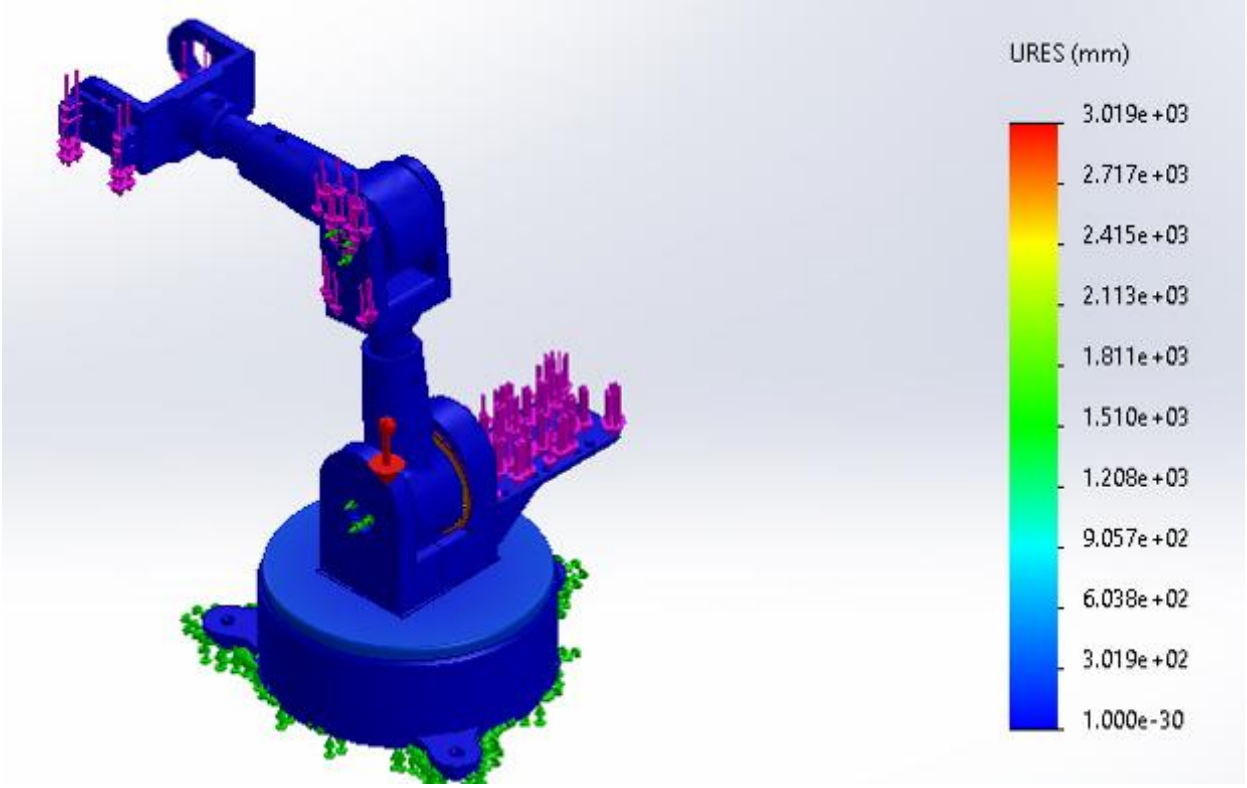
a. Stress

- Minimum Stress: 0.6822 N/m² (Node 16191)
- Maximum Stress: 3.556 MPa (Node 71670)
- Yield Strength of PLA: 45.2 MPa
- Conclusion: The maximum stress (3.556 MPa) is well below the yield strength of PLA (45.2 MPa), indicating that the material is safe under the applied loads.



b. Displacement

- Minimum Displacement: 0 mm
- Maxim um Displacement: 3.019 mm
- Conclusion: The maximum displacement is 3.019 mm, which is relatively small and acceptable for a robotic arm assembly.



c. Strain (Equivalent Strain)

- Minimum Strain: 7.057e-10 (Element 10891)
- Maximum Strain: 1.575e-03 (Element 38366)
- Conclusion: The strain values are within acceptable limits, indicating no permanent deformation.

d. Factor of Safety

- Minimum Factor of Safety: 12.711
- Maximum Factor of Safety: 66,255,804
- Conclusion: The factor of safety is very high, indicating that the design is robust and can handle much higher loads than those applied in this analysis.

End effector Holder Mechanical Design

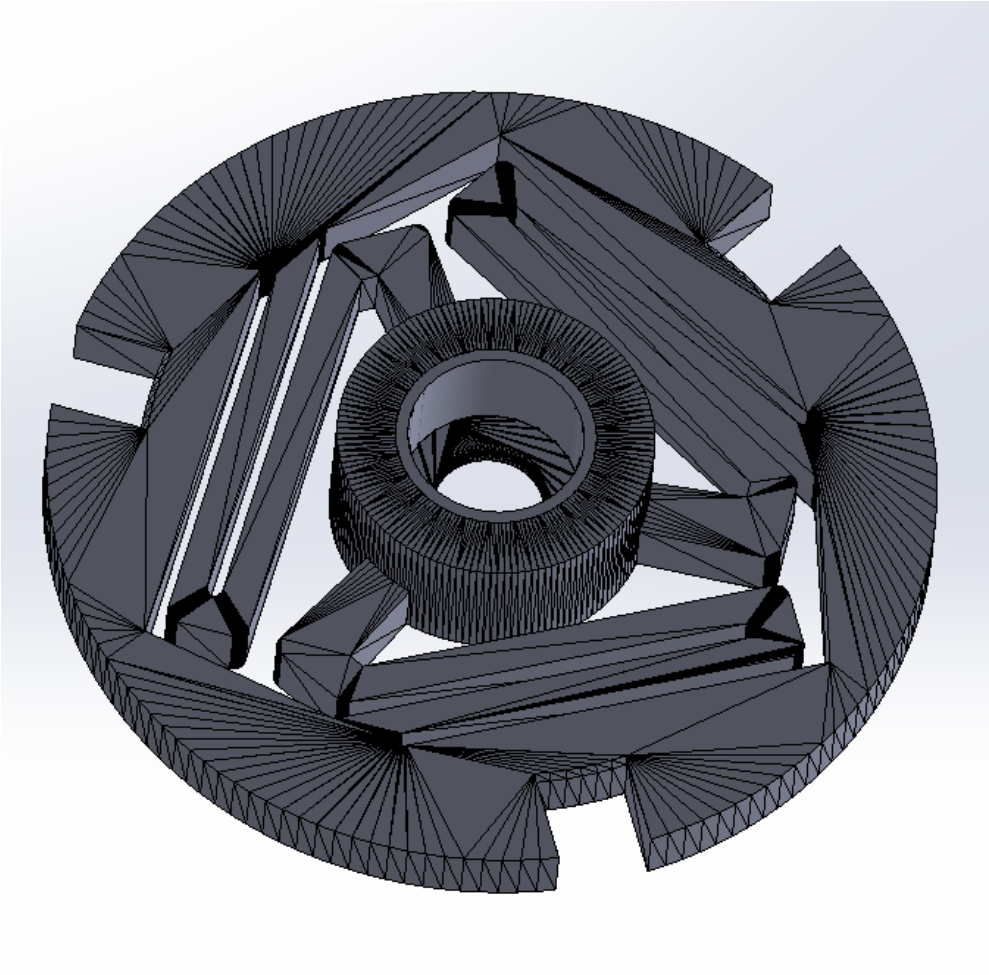
Stage 1 (Concept Analysis and Design Requirements Identification)

- To facilitate the efficient replacement of end-effector instruments, a spring-loaded four-bar mechanism was designed and integrated into the tool holder. This mechanism enables the motor pack to disengage from the instrument in a controlled manner. Upon disengagement, the instrument can be easily slid out and replaced with an alternate tool, ensuring quick and seamless interchangeability.

- The use of a spring-loaded design ensures reliable operation by providing the necessary force for controlled retraction and re-engagement, while the four-bar configuration offers mechanical stability and precision during the docking and undocking processes. This approach enhances the flexibility and adaptability of the robotic arm for performing a wide range of tasks.

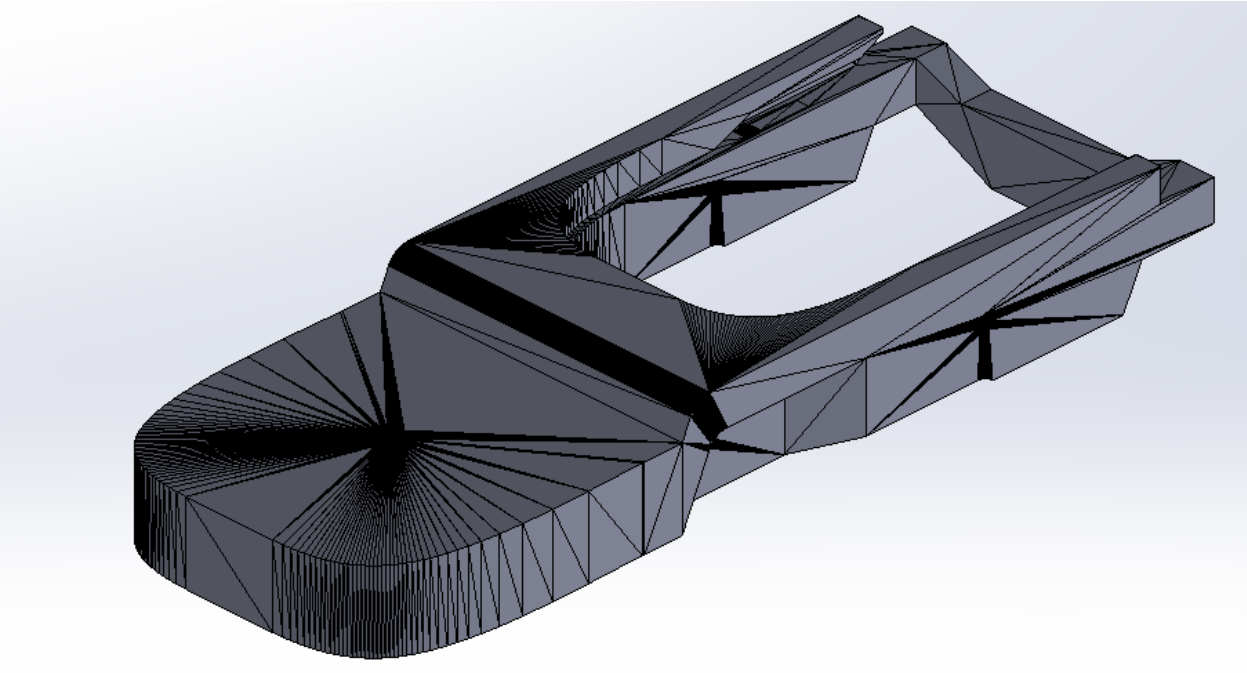
- A key feature of the tool holder design is the integration of a 3D-printed orthoplanar spring. The term "orthoplanar" refers to the spring's ability to deflect and rotate in directions that are orthogonal to one another. This innovative spring design enables compliant motion while maintaining structural integrity.

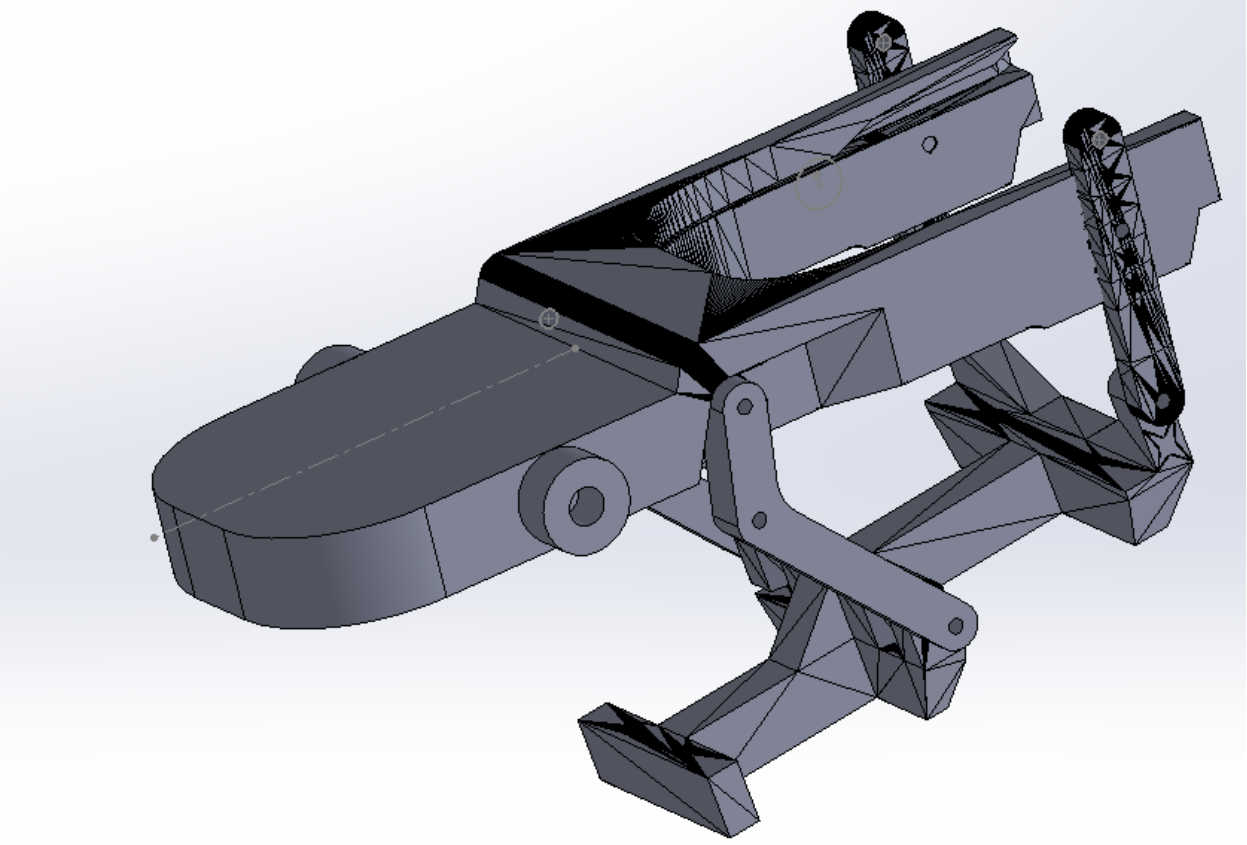
- This component is particularly advantageous in the application as it allows the connecting disk-like elements to flex vertically during the motor's homing sequence. This compliance ensures precise alignment of the motor and the instrument while effectively transmitting rotational motion between them. The orthoplanar spring's unique characteristics provide both flexibility and functionality, enhancing the reliability and performance of the docking system.



Stage 2 (Modifications and Adaptations in SolidWorks)

- During the design process, preliminary SolidWorks files were initially used as a reference. However, extensive modifications were made to adapt these files to meet the specific requirements of the tool holder and ensure compatibility with the four-bar mechanism and orthoplanar spring. The modifications included refinements in dimensions, geometry, and material properties to optimize the design for strength, precision, and performance. These adjustments were essential in achieving a robust and efficient solution for the end-effector tool holder, capable of meeting the demands of a dynamic and versatile robotic system.





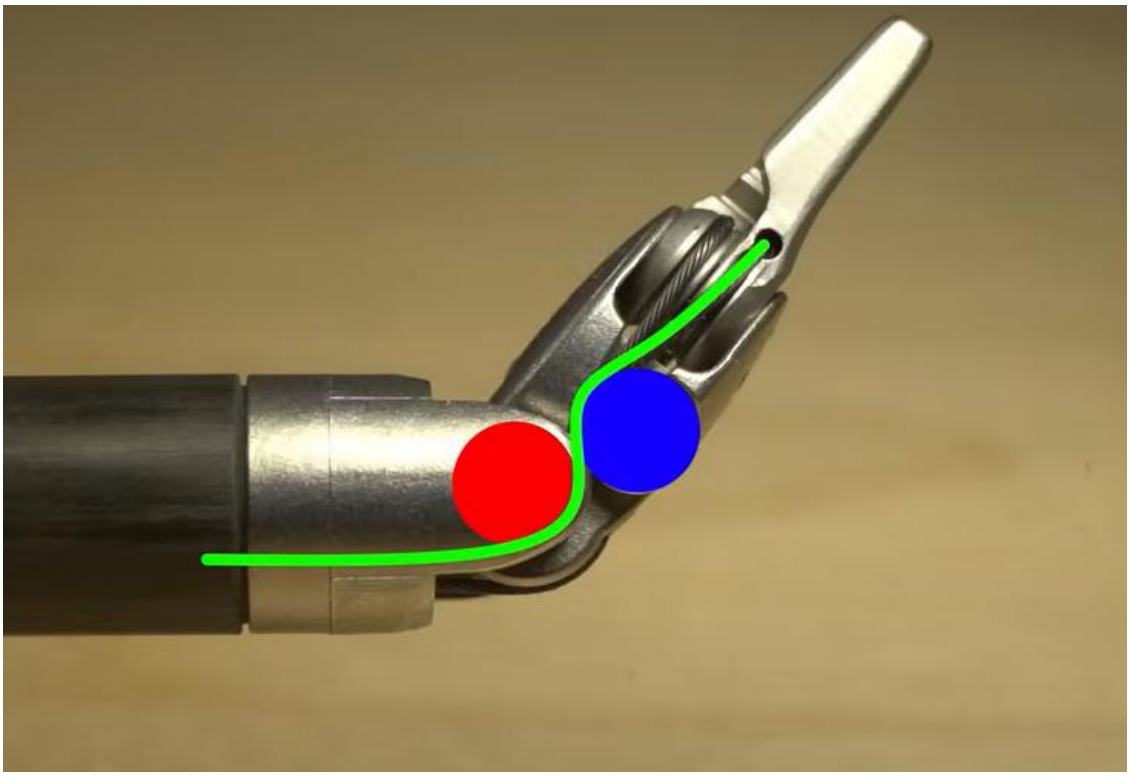
End effector

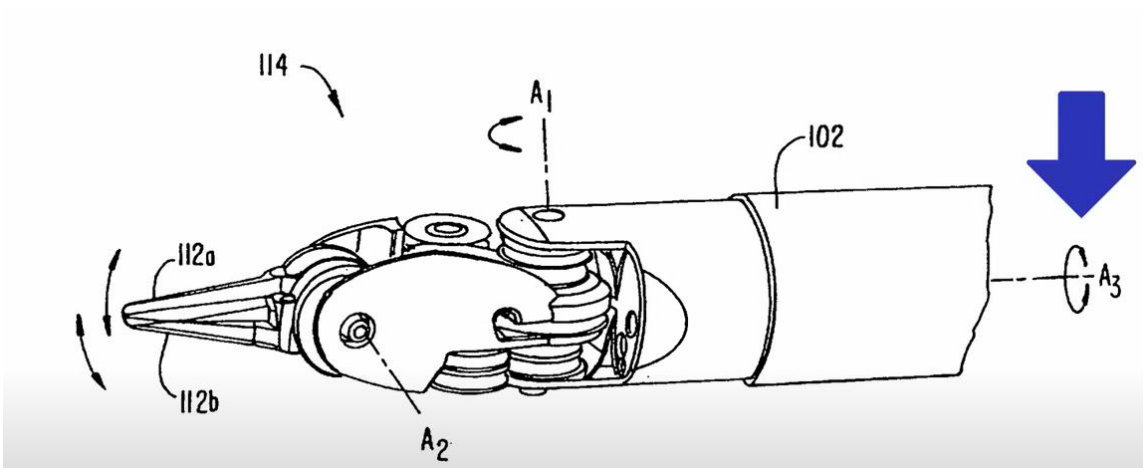
Stage 1 (Dimension Acquisition from Reference Sources)

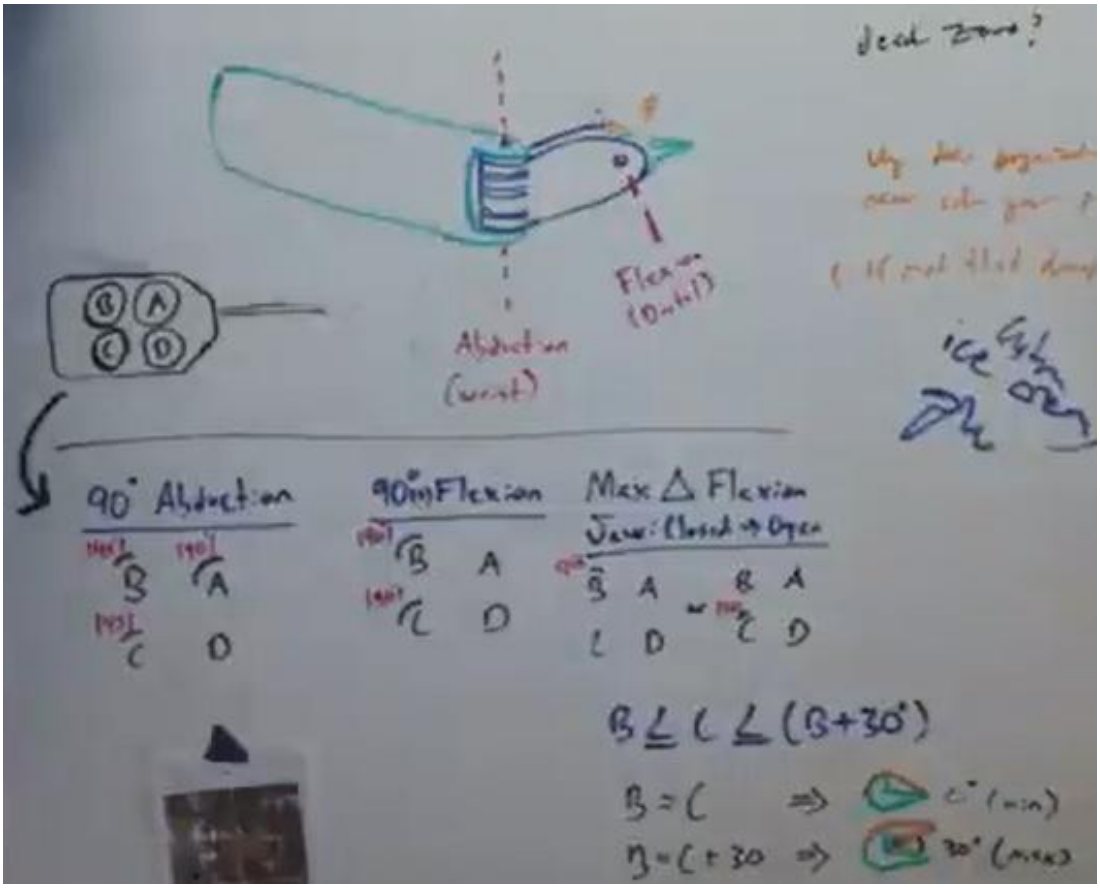
- Gather precise dimensional data and specifications from reference designs, existing models, or technical documentation.

Stage 2 (Functional Analysis and Reverse Engineering)

- Analyze the structure and functionality of the reference design to identify key features and understand the underlying mechanics.
- Reverse-engineer the reference model, focusing on optimizing or replicating its design for the robotic arm application.







How It Works

Pulley System:

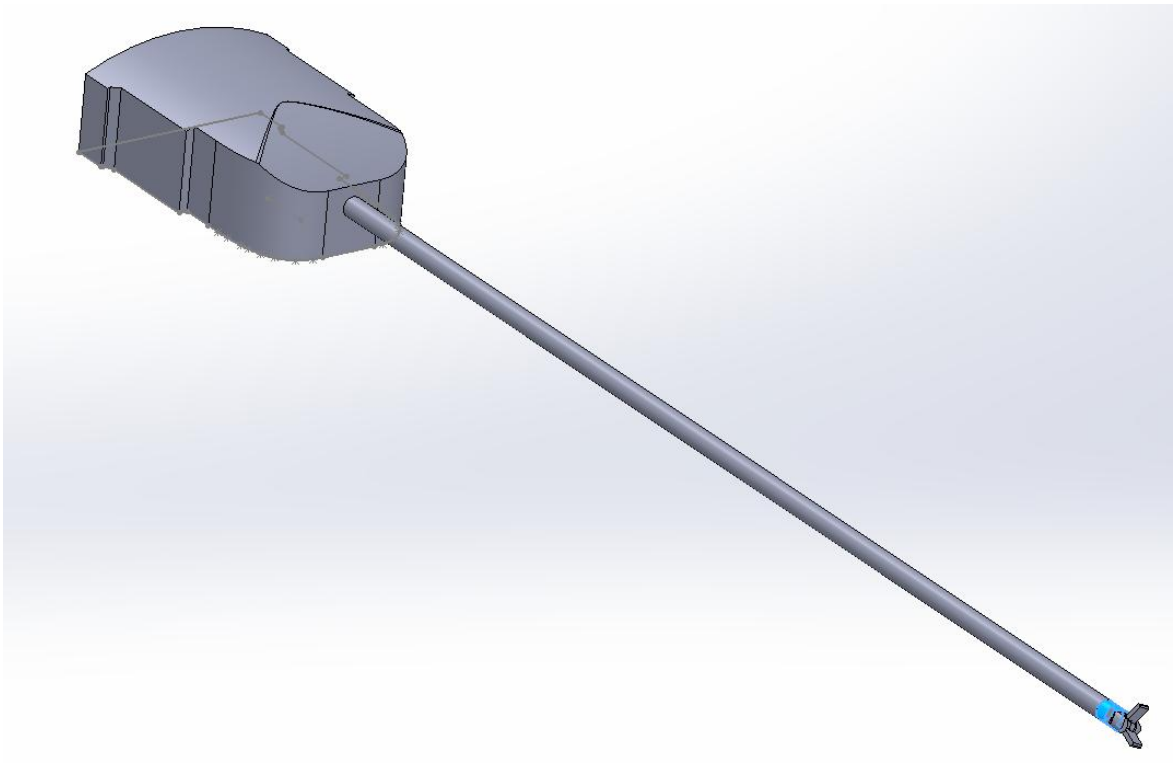
- The four pulleys are connected to cables or tendons, which are driven by external motors.
- Each cable corresponds to a specific motion:
    - Roll: Rotational motion along the longitudinal axis.
    - Pitch: Up-and-down movement.
    - Yaw: Side-to-side movement.
    - Jaw grip: The opening and closing of the tool's jaws.

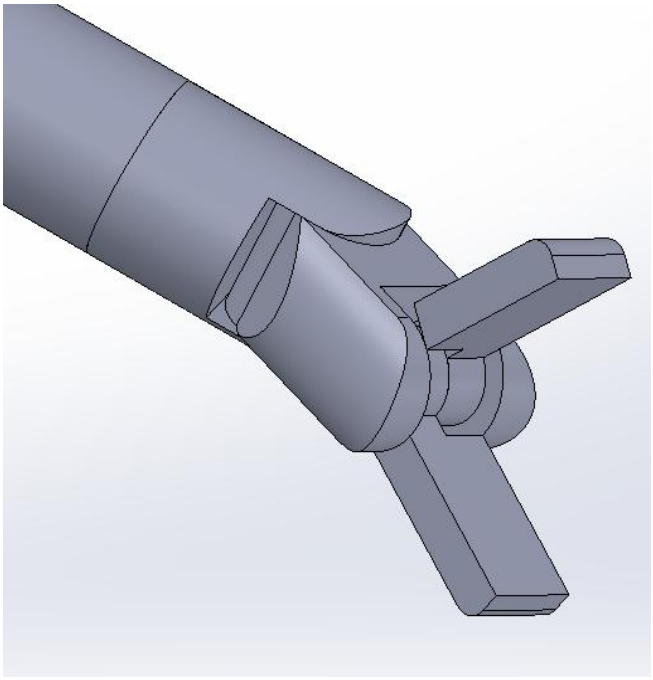
Tension Control:

- By adjusting the tension in specific cables, the pulleys work together to produce the desired movement.
- The movement is highly precise because each cable is independently controlled, allowing for a combination of motions.

Stage 3 (3D Modeling in SolidWorks)

- Create a fully detailed 3D model of the end-effector in SolidWorks.
- Incorporate functional features, ergonomic improvements, and compatibility with the robotic arm and holder interface.





Mechanical Future tasks

Stage 1 (Final Assembly and Motor Integration)

- Complete the assembly of the robotic arm and its components.
- Design and fabricate custom couplers for all motors to ensure secure and efficient connections to the mechanical system.

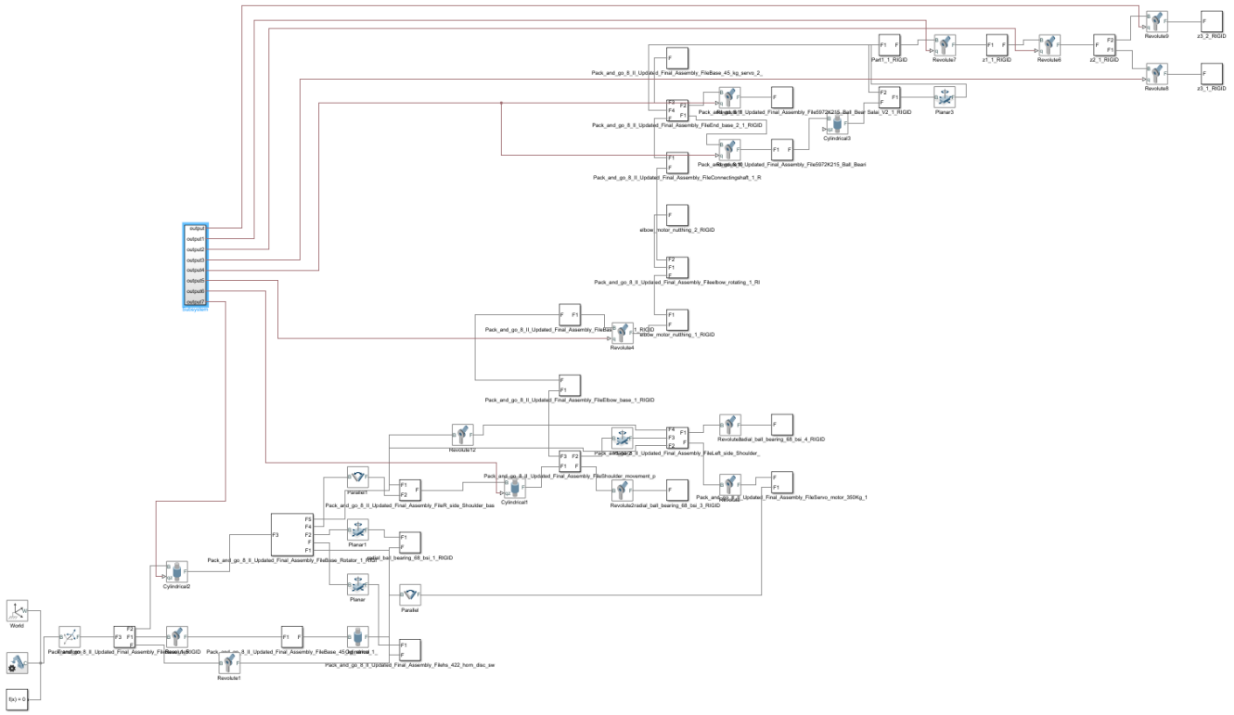
Stage 2 (Camera Station Design and Installation)

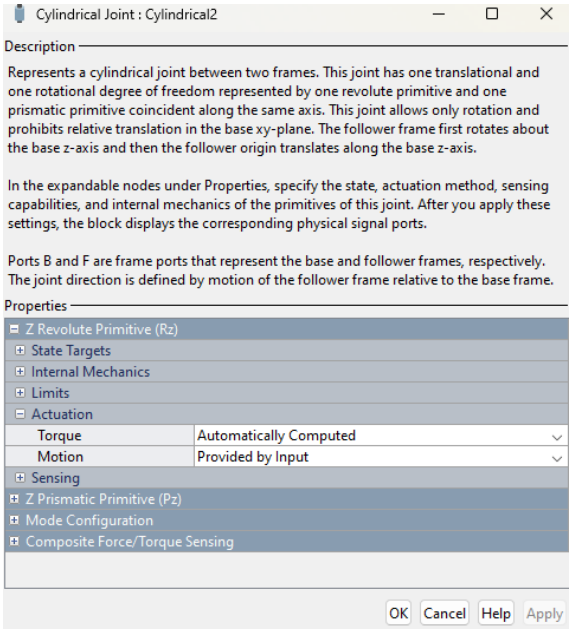
- Design and fabricate a dedicated station or mounting platform for the camera.
- Install and integrate the camera system into the overall robotic arm structure for enhanced functionality, ensuring proper positioning and stability during operation.

Pre-Stimulation

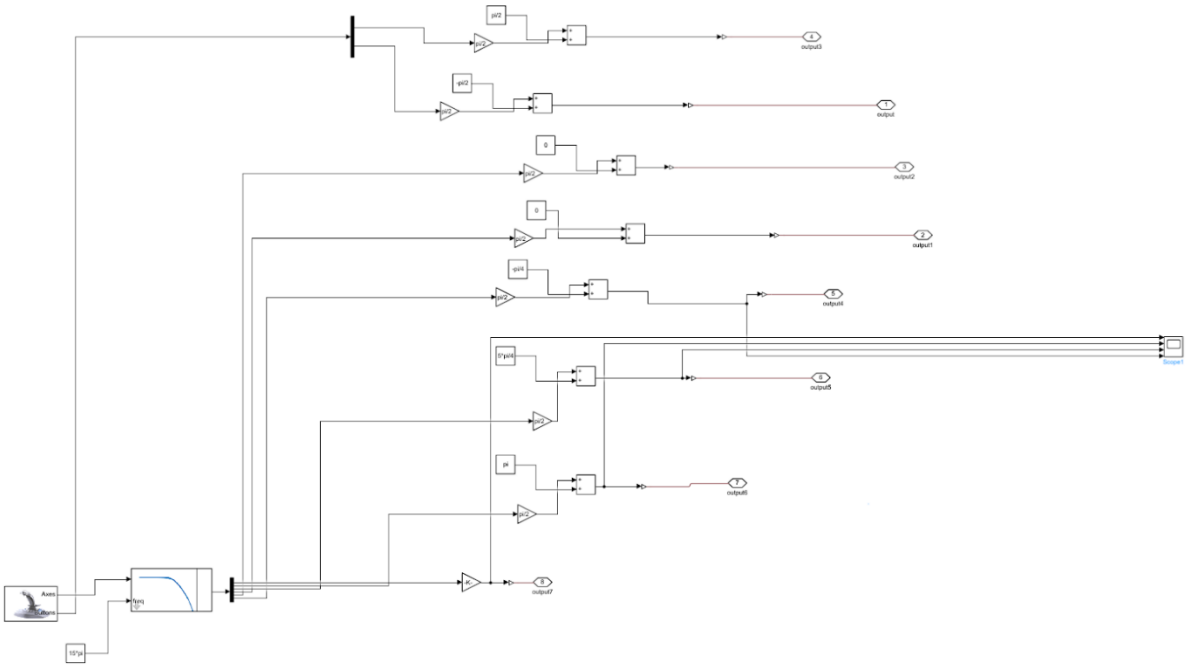
- Integration of SolidWorks and MATLAB

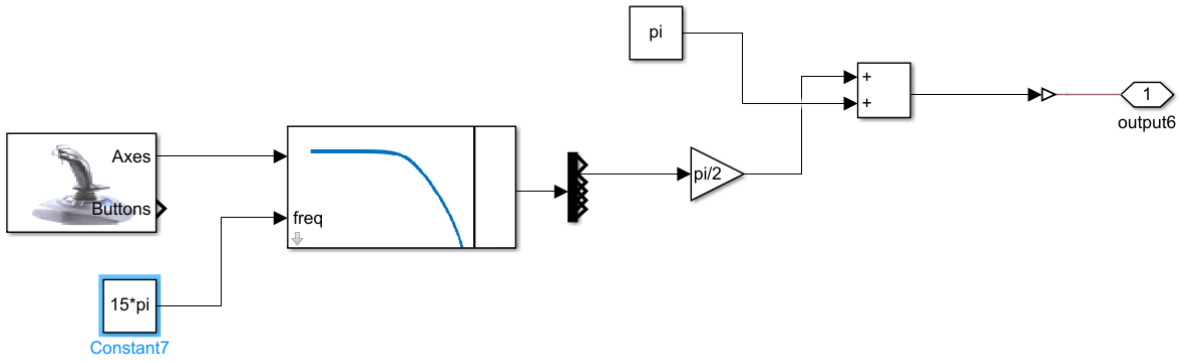
- Rigid Body Tree Modeling in MATLAB





- Direct Motor Actuation and Control





Elementary Robotics

Several themes and concepts need to be explained in some detail in order to begin our simulation process.

1-Coordinate frames:

A coordinate frame i consists of an origin, denoted Oi, and a triad of mutually orthogonal basis vectors, denoted (xi yi zi), that are all fixed within a particular body. Motion of a body is always described relative to some other body(pose of one coordinate frame relative to another).

2-Pose vector:

The position of the origin of coordinate frame i relative to coordinate frame j can be denoted by the 3x1 vector

$$ {}^{j}\boldsymbol{p}_i = \begin{pmatrix} {}^{j}p_i^x \\ {}^{j}p_i^y \\ {}^{j}p_i^z \end{pmatrix}. $$

3-Rotation Matrices:

The orientation of some coordinate frame i relative to some other frame j can be denoted as:

$$ ({}^{j}\hat{\boldsymbol{x}}_i \; {}^{j}\hat{\boldsymbol{y}}_i \; {}^{j}\hat{\boldsymbol{z}}_i) $$

Or in matrix form:

$$ {}^{j}\mathbf{R}_i = \begin{pmatrix} \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{x}}_j & \hat{\boldsymbol{y}}_i \cdot \hat{\boldsymbol{x}}_j & \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{x}}_j \\ \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{y}}_j & \hat{\boldsymbol{y}}_i \cdot \hat{\boldsymbol{y}}_j & \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{y}}_j \\ \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{z}}_j & \hat{\boldsymbol{y}}_i \cdot \hat{\boldsymbol{z}}_j & \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{z}}_j \end{pmatrix} $$

Rotation matrices can be combined by simple matrix multiplication to denote an overall transformation.

4-Forward Kinematics:

The forward kinematics problem is simply finding the pose of some part of the robot (usually the tool) given actuation angles to joints. It is straightforward and easy to solve. For example, actuation of a sample manipulator chain consisting of six joints is

$$^{0}\mathbf{T}_{6} = {}^{0}\mathbf{T}_{1}{}^{1}\mathbf{T}_{2}{}^{2}\mathbf{T}_{3}{}^{3}\mathbf{T}_{4}{}^{4}\mathbf{T}_{5}{}^{5}\mathbf{T}_{6}$$

5-Inverse Kinematics:

The inverse kinematic problem in some sense is the opposite (or rather the reverse) of the forward kinematic problem: find the values of the joints positions given the position and orientation of the end effector relative to the base.

6-Jacobian:

Differentiation with respect to time of the forward position kinematics equations yields a set of equations of the form

$$^{k}\boldsymbol{v}_{N} = \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}},$$

Alternatively written as

$$^{k}\boldsymbol{v}_{N} = [\mathbf{J}_1 \;\; \mathbf{J}_2 \;\; \cdots \;\; \mathbf{J}_N]\; \dot{\boldsymbol{q}},$$

where N is the number of joints and Ji provides thecol umn(s) of J.q/ which correspond(s) to P qi. Solving for the DH parameters.

6-DH parameters:

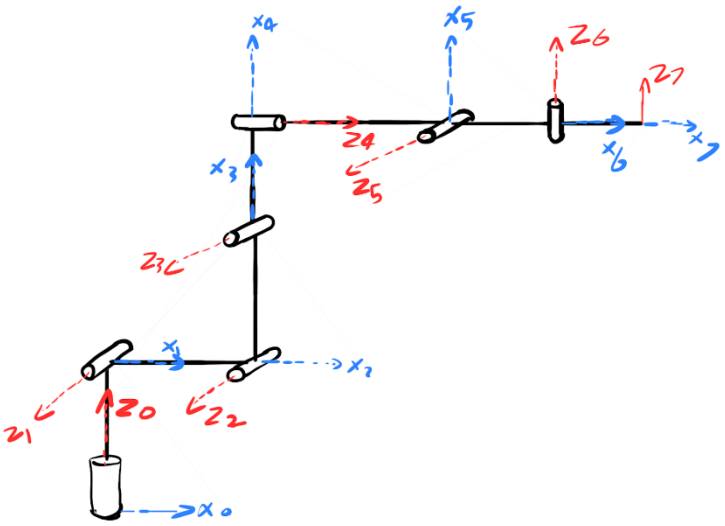
The DH parameters are a convention to generalize transformation matrices of serial manipulators given the parameters (a, alpha,d theta) for each link.

$$^{n-1}T_n = \left[\begin{array}{cccc} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

- d: offset along previous z to the common normal
- θ: angle about previous z from old x to new x
- r: length of the common normal (aka a, but if using this notation, do not confuse with α). Assuming a revolute joint, this is the radius about previous z.
- α: angle about common normal, from old z axis to new z axis

Robot's DH Paramters

By studying the robot, we acquire the following parameters



link number	a	alpha	d	theta
1	0	90	147	0
2	218	0	0	0
3	196	0	0	0
4	36	90	0	0
5	0	-90	425	0
6	7.5000	-90	0	0
7	11	0	0	0

Stimulation

Back to our model, we now explore how we implement the previous concepts relatively.

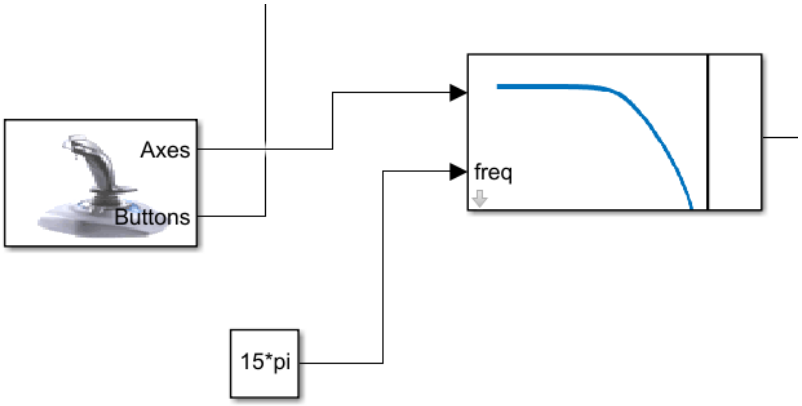
Experiments:

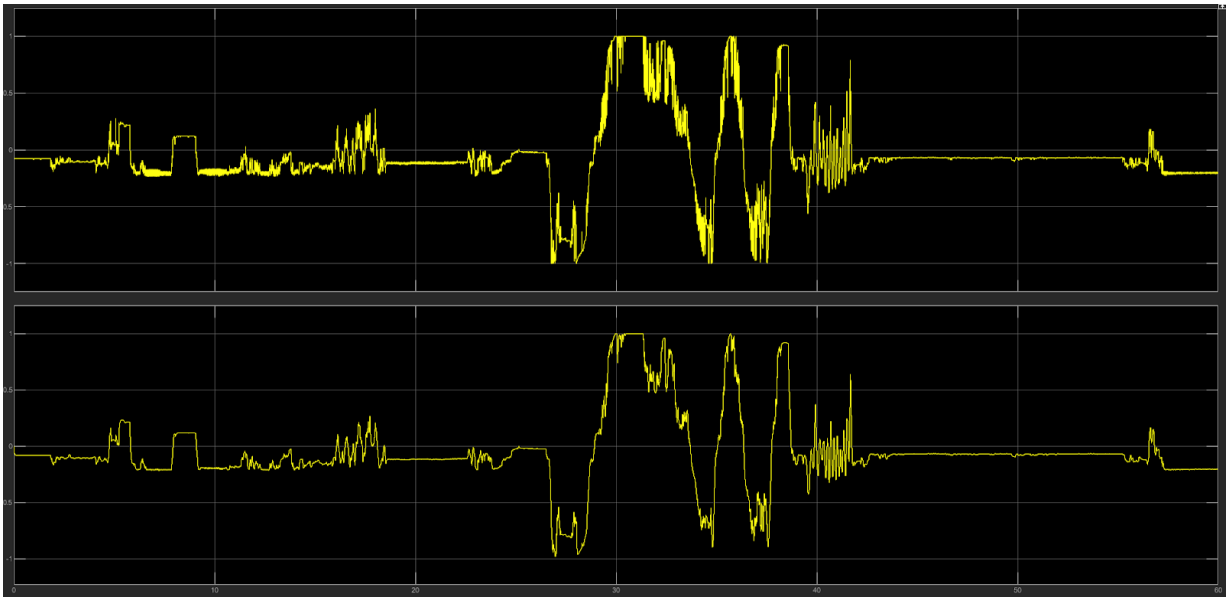
1. Interface PS4 controller with MATLAB and direct joint actuation

The first stage of simulation requires actively taking input from the analog PS4 sensor and scaling the reading directly to the robot's joints

2. Noise filtration

From the previous experiment, we find that the PS4 controller has some noise and we a assume a trial and error methodology to eliminate said noise.





Filtered output (bottom) against unfiltered output(top)

Building a custom Kinematic Solver

In order to get a proper insight in the workings of a kinematic solver and in case we needed to implement a custom solver (to prevent issues of latency or incompatibility for example), we decided to build a custom solver capable solving both forward and inverse kinematics.

Forward Kinematics

The forward kinematics is simply a series of matrix multiplication.

```
function result_transform = fk_for_gui(n, theta, alpha, a, d)


theta=theta*(pi/180)
alpha=alpha'*(pi/180)
a=a'
d=d'

% n is the number of links in the manipulator

trans_mat = []; % Reserved multidimensional matrix to hold the various transformation matrices resulting from DH implementation.

result_transform = eye(4); % Initialize with identity matrix

for i = 1:n
    trans_mat(:,:,i) = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));
                        sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));
                        0, sin(alpha(i)), cos(alpha(i)), d(i);
                        0, 0, 0, 1];
    result_transform = result_transform * trans_mat(:,:,i); % Multiply matrices (recursive function)
end
end
```

Inverse Kinematics

The inverse kinematics is a lot harder and needs to be solved numerically. Two optimization techniques are used: gradient descent and Levenberg.

"finding the Jacobian"

$$J(\theta) = \frac{\partial f}{\partial \theta}$$

linear approximation → to be solved by iterative methods

$$\Delta P = J(\theta)\,\Delta\theta$$

$$\Delta\theta = J^{-1}(\theta)\,\Delta P$$

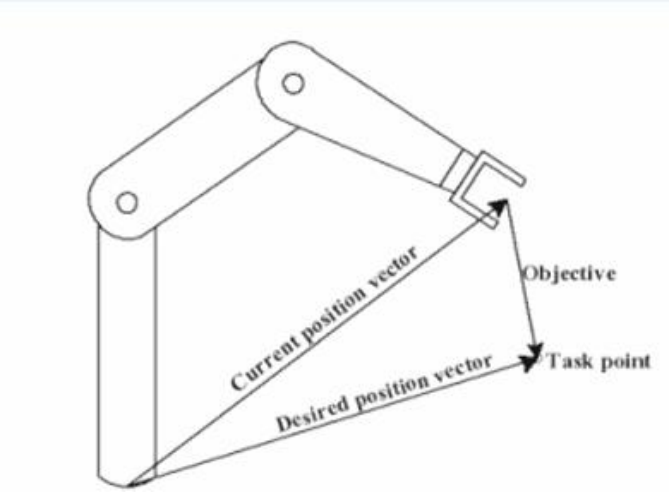
$\Delta\theta$: link angles; J^{-1}: Inverse of Jacobian; ΔP: change in Position of End effector

ΔP is the difference between target position and current end effector position

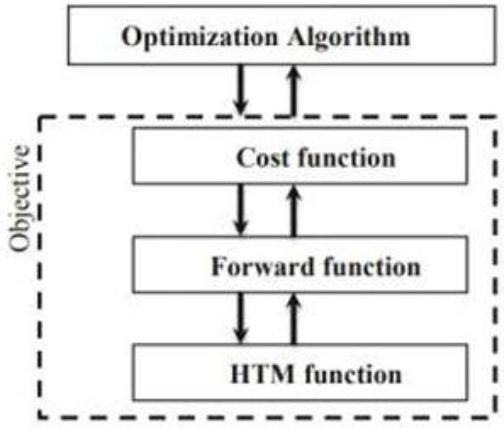
$\Delta P = \vec{e} = t - P$

$\vec{e}$: error vector; t: desired pose; P: current pose

We want to minimize the error function so that we obtain valid estimates.



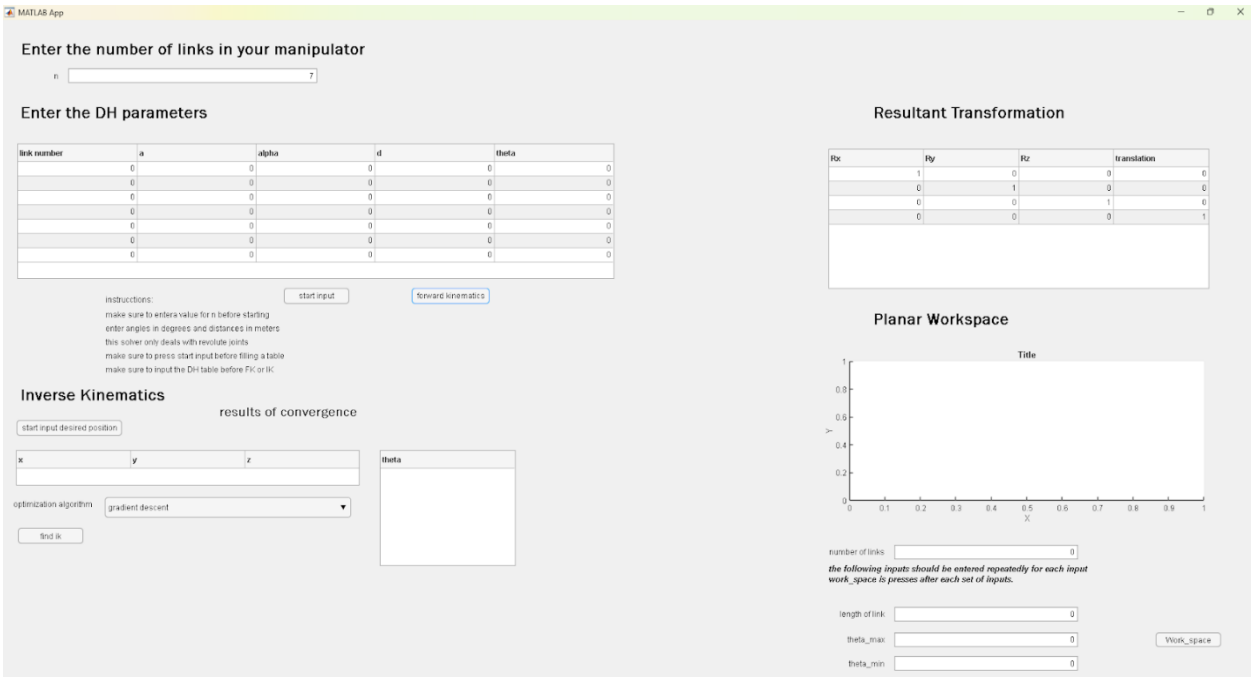
Optimization Algorithms for Inverse Kinematics of Robots



Code snippets for gradient descent optimization

```
function joint_angles = inverse_kinematics_with_prismatic_for_gui(desired_pose, n, alpha, a, d)


    alpha=alpha'
    a=a'
    d=d'
    alpha=alpha*pi/180;




    % Function parameters
    joint_angles = zeros(n,1);
    max_iterations = 300000;
    tolerance = 1e-6;
    learning_rate = 0.001;                %preset value
    epsilon = 1e-6;                       % Small value for numerical differentiation
    J = zeros(3, n);
    error_history = [];
    count=0;
    stagnation_window = 10000;      % Number of iterations to consider for stagnation



    current_pose = fk(n, joint_angles, alpha, a, d);   %forward kinematics

    for iter = 1:max_iterations
        error = norm(desired_pose - current_pose(1:3,4));
        error_history(iter) = error;

        if iter >= 2 && round(error_history(iter),6) == round(error_history(iter-1),6)

         count=count+1;

        end

        % Check convergence or stagnation
        if (error <= tolerance) || (count== stagnation_window);
            disp('Inverse kinematics converged');
            break;
        end
```

```
        % Update Jacobian matrix
        for i = 1:n

            joint_angles_incremented = joint_angles;
            joint_angles_incremented(i) = joint_angles_incremented(i) + epsilon;
            new_pose = fk(n, joint_angles_incremented, alpha, a, d);
            delta_pose = new_pose(1:3, 4) - current_pose(1:3, 4);
            J(:, i) = delta_pose / epsilon;
        end

        % Gradient descent update
        gradient = J' * (desired_pose' - current_pose(1:3,4));
        joint_angles = joint_angles + learning_rate * gradient;


        % Update current pose
        current_pose = fk(n, joint_angles, alpha, a, d);
    end

    if iter == max_iterations
        disp('Inverse kinematics did not converge within max iterations.');
    end
end
```



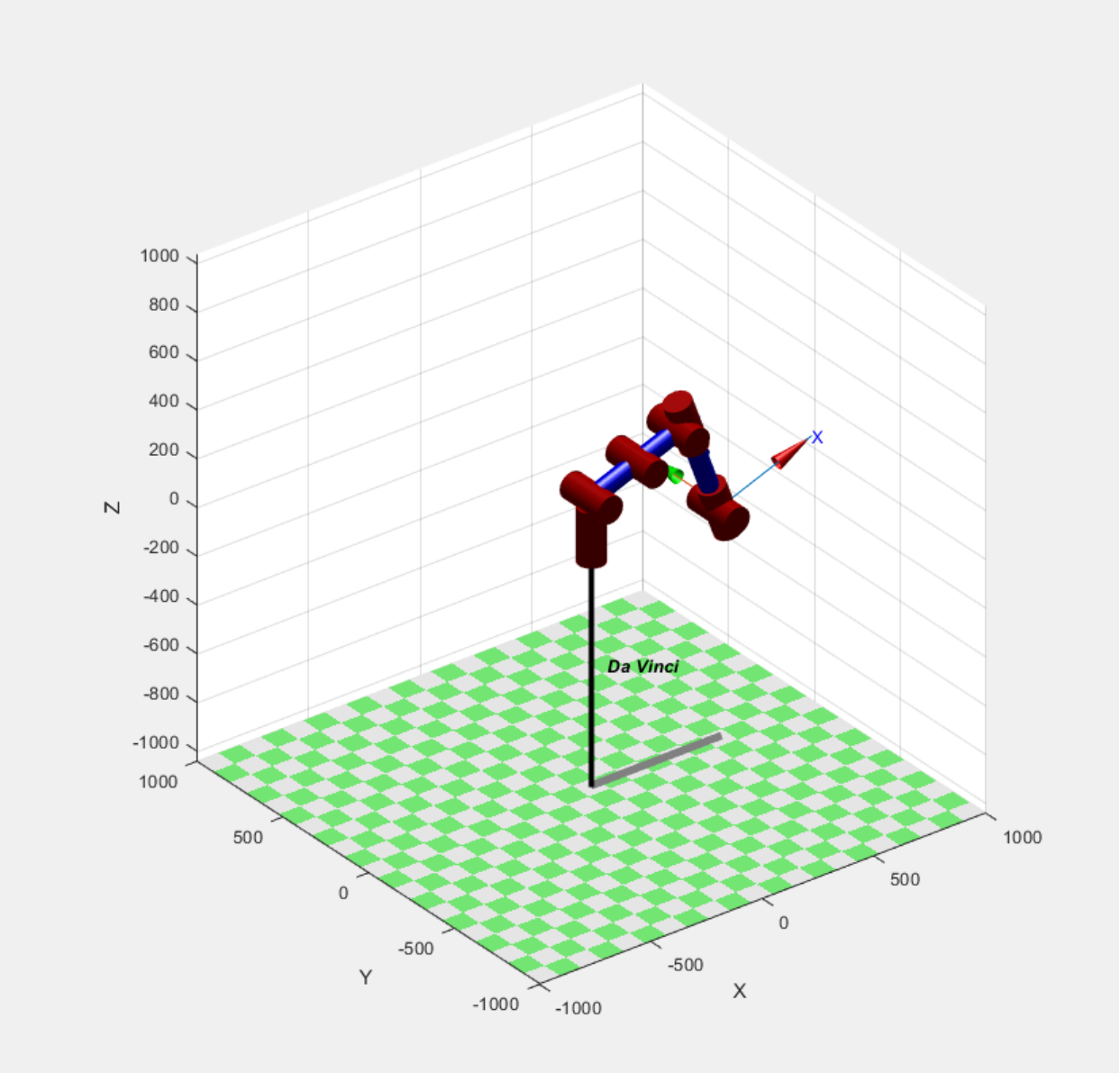
Resultant Transformation

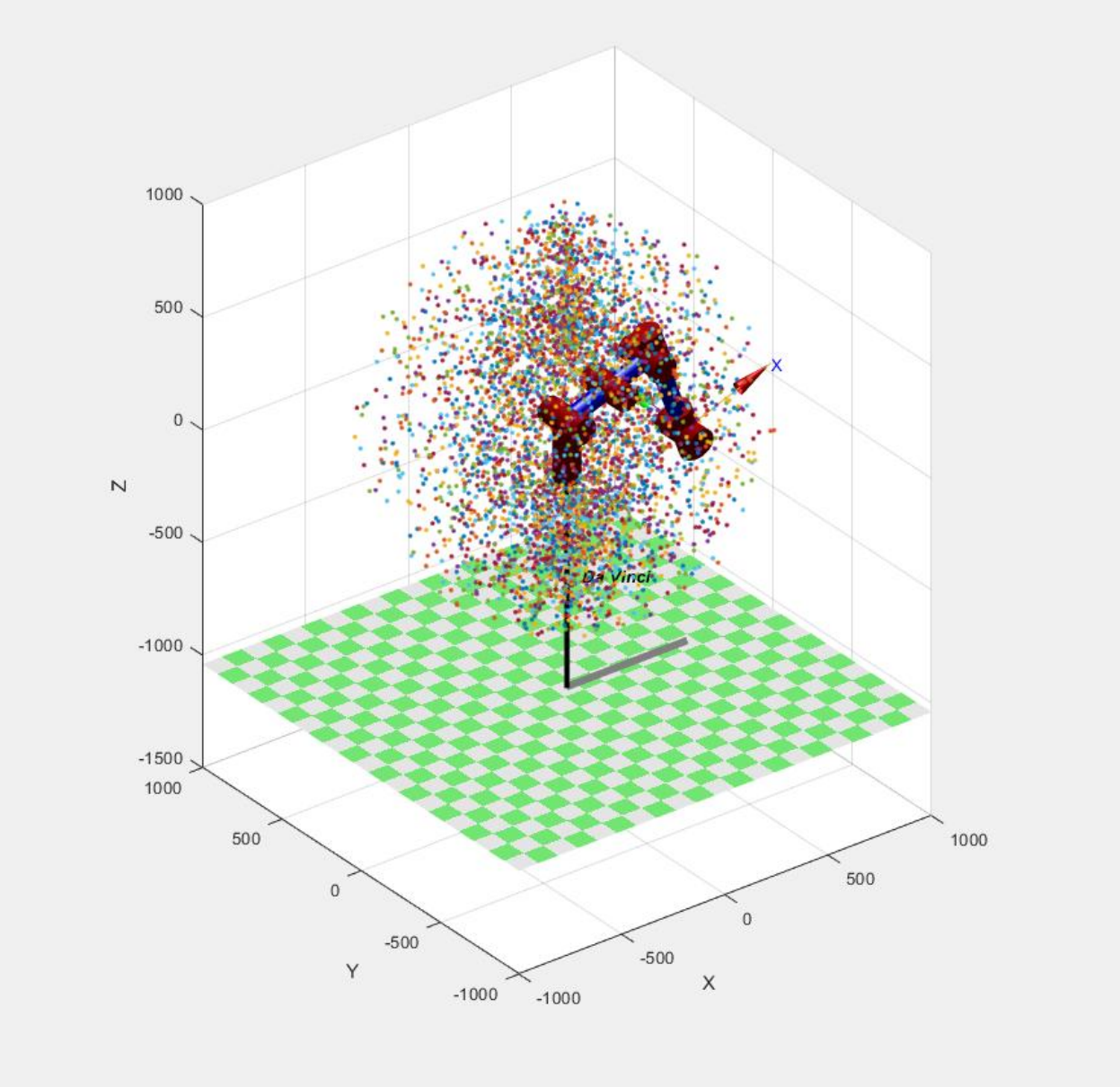
Rx	Ry	Rz	translation
1.0000	0	0	468.5000
0	1.0000	0	-0.0000
0	0	1.0000	-278.0000
0	0	0	1.0000

Peter Corke's Toolbox

We used peter's toolbox to plot the robot's workspace.

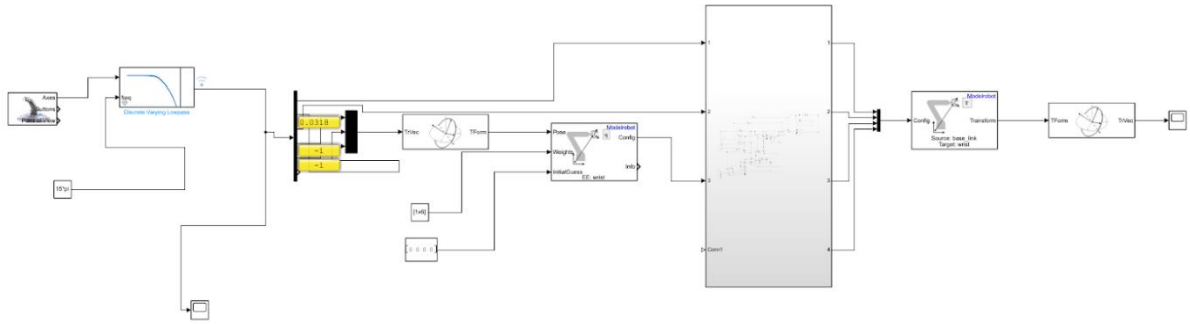
```
clear,clc,clf
clear L

L1= 147;
L2 = 218;
L3 = 196 ;
L4 = 36 ;
L5 = 425;
L6 = 7.5;
L7 = 11;

 % th d a alpha
L(1) = Link([0 L1 0 pi/2]);
L(2) = Link([0 0 218 0]);
L(3) = Link([0 0 196 0]);
L(4) = Link([0 0 36 pi/2]);
L(5) = Link([0 425 0 -pi/2]);
L(6) = Link([0 0 7.5 -pi/2]);
L(7) = Link([0 0 11 0]);

r = SerialLink(L)
r.name = ' Da Vinci'

for i =1:5000

    q1 = (rand()*270)* pi/180 ;
    q2 = (rand()*360)* pi/180 ;
    q3 = (rand()*270)* pi/180 ;
    q4 = (rand()*180)* pi/180 ;
    q5 = (rand()*180)* pi/180 ;
    q6 = (rand()*180)* pi/180 ;
    q7 = (rand()*180)* pi/180 ;

    T = r.fkine([q1 q2 q3 q4 q5 q6 q7]);

    v =transl(T);
    plot3(v(1),v(2),v(3),'.')
    grid on
    hold on

end
r.plot([0 20*(pi/180) 0 0 0 0 0 ])
```


1000
800
600
400
200
0
-200
-400
-600
-800
-1000
Z
1000
500
0
-500
-1000
Y
-1000
-500
0
500
1000
X
Da Vinci
X


1000
500
0
-500
-1000
-1500
Z
Da Vinci
X
1000
500
0
-500
-1000
Y
1000
500
0
-500
-1000
X

Simple simulink solver model

For comparison purposes, we built a simple MATLAB Simulink model to compare the results of the solver and Corke's toolbox



Network Design options (putting it all together)

ROS and ROS2 both offer good options for building the whole robot ecosystem. However, each has its pros and cons.

Aspect	ROS 1	ROS 2
Architecture	Pros: Simple and well-understood architecture.	Pros: Decentralized, modular, and scalable.
	Cons: Centralized ROS Master creates a single point of failure.	Cons: More complex due to DDS middleware.
Communication	Pros: Easy to use with TCPROS/UDPROS.	Pros: DDS-based communication with advanced QoS settings.
	Cons: Limited QoS, no real-time support, and poor scalability.	Cons: Steeper learning curve due to DDS configuration.

Real-Time Capabilities	**Pros: None.**	**Pros: Designed for real-time systems, integrates with RTOS.**
	Cons: Not suitable for real-time applications.	**Cons: Requires additional setup for real-time performance.**
Platform Support	**Pros: Strong support for Linux (Ubuntu).**	**Pros: Cross-platform support (Linux, Windows, macOS, RTOS).**
	Cons: Limited support for other operating systems.	**Cons: Some platforms may require additional configuration.**
Networking	**Pros: Works well for small-scale, single-machine setups.**	**Pros: Designed for distributed systems and complex network topologies.**
	Cons: Poor performance in distributed and multi-machine setups.	**Cons: More complex to configure for advanced networking.**
Ecosystem	**Pros: Mature ecosystem with thousands of packages and tools.**	**Pros: Growing ecosystem with active development and industry adoption.**
	Cons: Limited support for modern robotics needs.	**Cons: Fewer packages compared to ROS 1 (as of now).**
Backward Compatibility	**Pros: N/A (original framework).**	**Pros: Provides tools like ros1_bridge for interoperability.**
	Cons: Not compatible with ROS 2.	**Cons: Not fully backward compatible with ROS 1.**

Performance	**Pros: Adequate for small-scale and non-real-time applications.**	**Pros: Improved performance, scalability, and lower latency.**
	Cons: Performance degrades in large-scale or real-time systems.	**Cons: Requires more resources due to DDS middleware.**
Security	**Pros: N/A (no built-in security features).**	**Pros: Built-in security features (authentication, encryption).**
	Cons: Vulnerable to attacks in networked environments.	**Cons: Security setup can be complex.**
Use Cases	**Pros: Ideal for research, education, and prototyping.**	**Pros: Suitable for industrial, real-time, and large-scale systems.**
	Cons: Not suitable for industrial or real-time applications.	**Cons: Overkill for small-scale or simple projects.**

Sample ROS system layout

ROS (Robotic Operating System) is the core of the whole operation. *Figure 3.1* shows an illustration of the centralized Robotic Surgeon system.

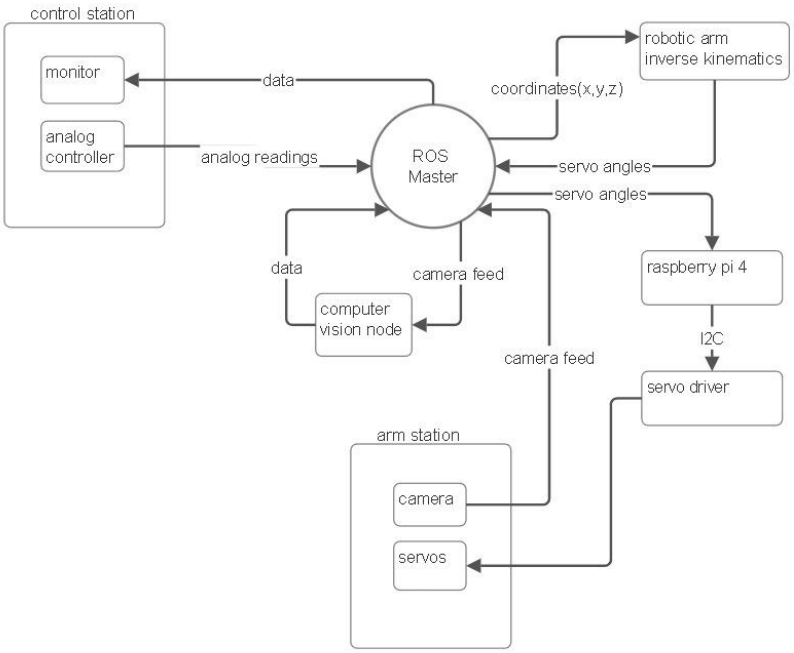


Figure 3.1(ros nodes)

Due to the centralized computing nature of ROS1 all nodes must be registered with the ROS Master. We will discuss each node in some details.

For the purposes of testing and debugging a simple publisher/subscriber model is implemented.

Interfacing the raspberry with ROS Master

To use ROS noetic on the raspberry pi, an image of Ubuntu noetic was burned and ROS installed.

For convenience, the master machine and the raspberry pi are connected via ethernet and given permanent static IPs for remote access.

```
mohamed@mohamed-G3-3500:~$ printenv | gerp ROS

Command 'gerp' not found, did you mean:

  command 'grep' from deb grep (3.4-1)

Try: sudo apt install <deb name>

mohamed@mohamed-G3-3500:~$ printenv | grep ROS
ROS_VERSION=1
ROS_PYTHON_VERSION=3
ROS_PACKAGE_PATH=/opt/ros/noetic/share
ROSLISP_PACKAGE_DIRECTORIES=
ROS_IP=192.168.2.1
ROS_ETC_DIR=/opt/ros/noetic/etc/ros
ROS_MASTER_URI=http://192.168.2.1:11311
ROS_ROOT=/opt/ros/noetic/share/ros
ROS_DISTRO=noetic
mohamed@mohamed-G3-3500:~$
```

The master is given 192.168.2.1 and the raspberry is given 192.168.2.2.

```
mohamed@mohamed-G3-3500:~$ ssh ubuntu@192.168.2.2
ubuntu@192.168.2.2's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-1115-raspi armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sat Sep 28 22:12:40 UTC 2024

  System load:  0.08               Temperature:           51.1 C
  Usage of /:   17.3% of 28.12GB   Processes:             152
  Memory usage: 3%                 Users logged in:       0
  Swap usage:   0%                 IPv4 address for eth0: 192.168.2.2
```

Controller Node

The controller node is on the master machine. It sends a message of type “joy ” containing all data from the controller with a frequency of 10Hz.

```
mohamed@mohamed-G3-3500:~/Documents/grad_proj$ rostopic echo /joy
header:
  seq: 1249
  stamp:
    secs: 1727906046
    nsecs: 864936113
  frame_id: ''
axes: [-0.019622802734375, 0.011749267578125, -1.0, 0.011749267578125, 0.00390625, -1.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
header:
  seq: 1250
  stamp:
    secs: 1727906046
    nsecs: 964903116
  frame_id: ''
axes: [-0.019622802734375, 0.011749267578125, -1.0, 0.011749267578125, 0.00390625, -1.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
```

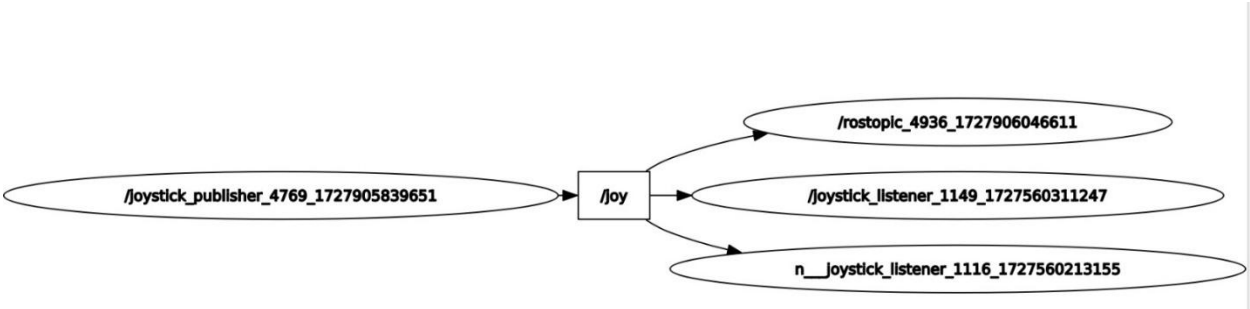
This message contains all data from the controller which will later be filter in a subscriber node on the raspberry pi.

Subscriber on pi

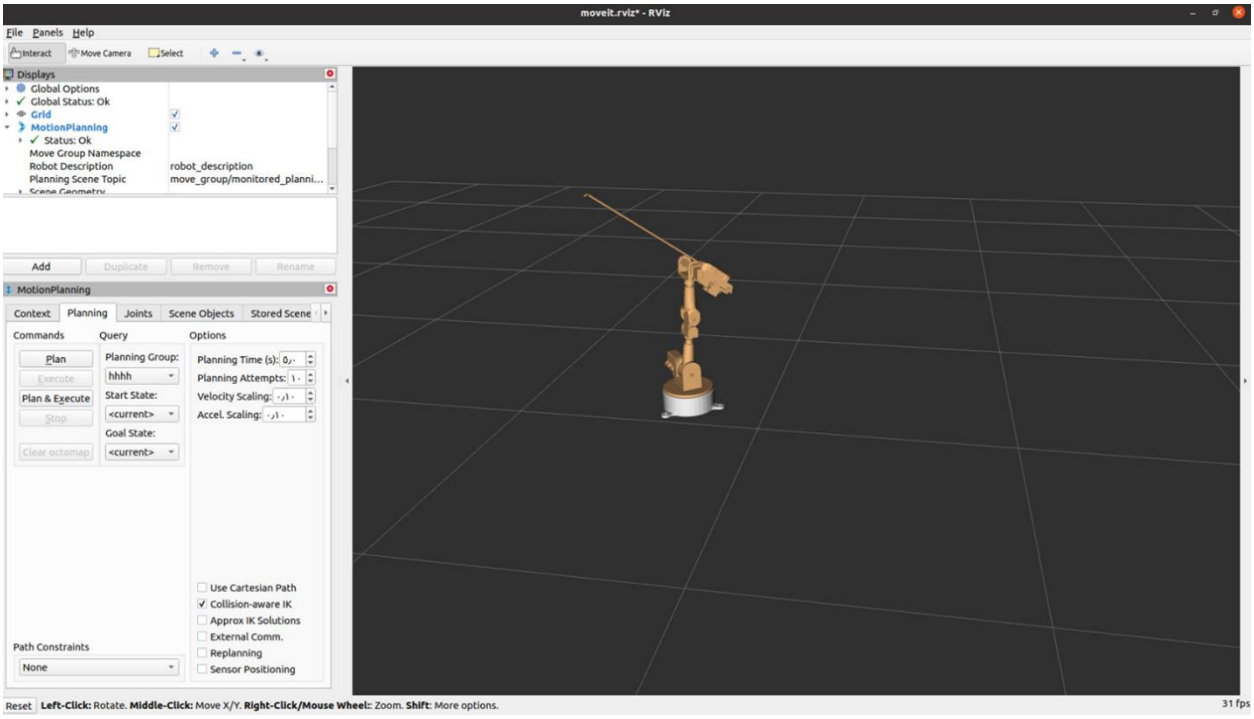
```
ubuntu@ubuntu:~/Documents/grad_proj$ rosrun from_master from_master.py
[INFO] [1727560311.487899]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560311.587634]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560311.687576]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560311.787581]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560311.887545]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560311.987613]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560312.087580]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560312.187441]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560312.287506]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560312.387552]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560312.487577]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560312.587686]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
[INFO] [1727560312.687633]: Axis 1: -0.019622802734375, Axis 2: 0.011749267578125, Axis 3: -1.0, Axis 4: 0.011749267578125
```

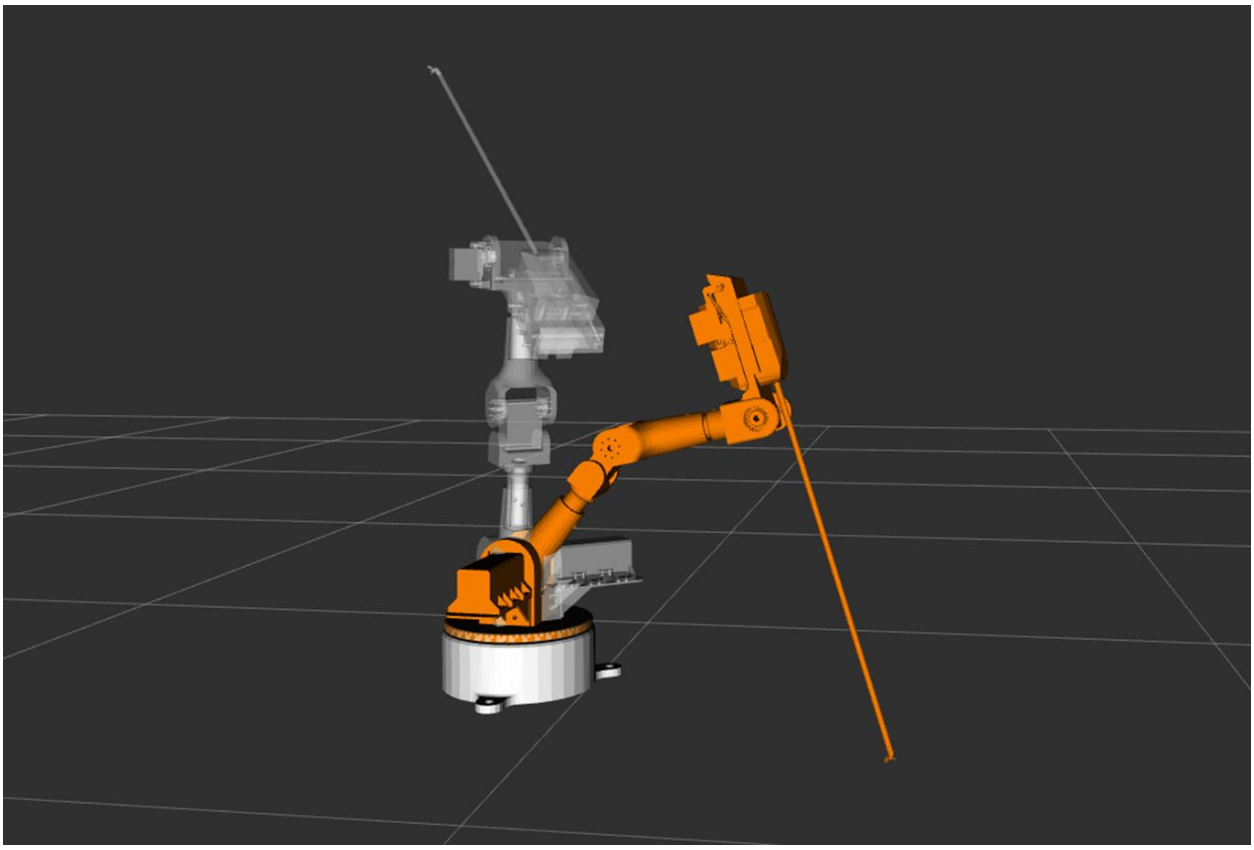
The pi subscriber logs only four axes values. These values will later be scaled in range 0-180 for servo control.

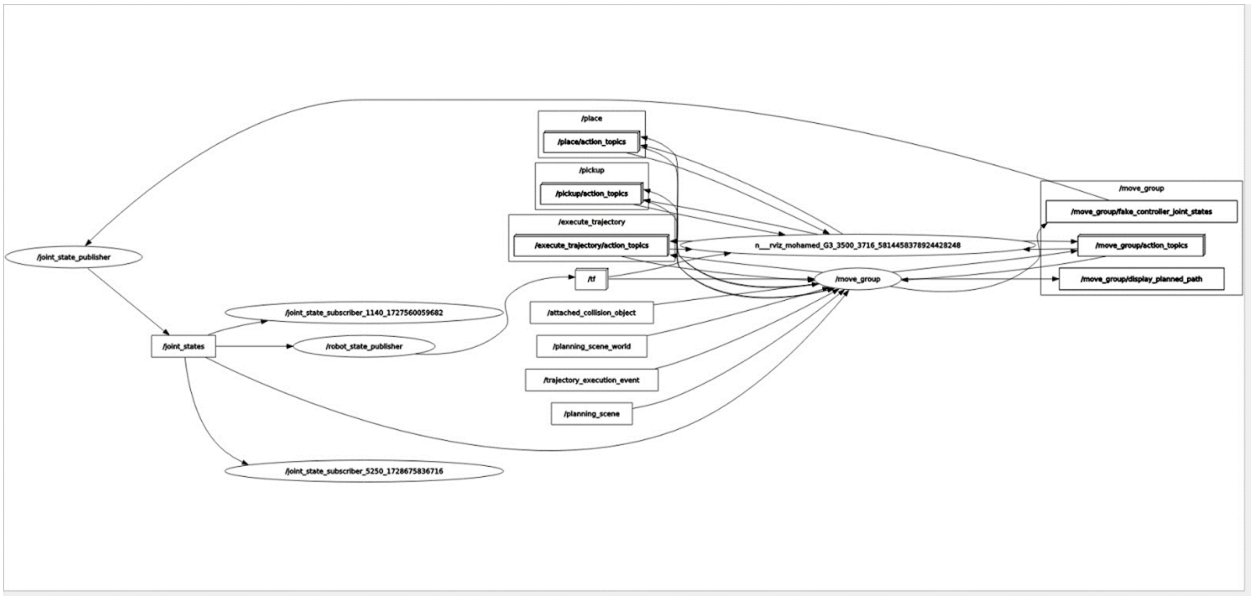
The corresponding rqt graph is shown



This configuration implements an inverse kinematics solver using an API from “Moveit”. A package is created using the Moveit Setup Assistant.






/place
/place/action_topics
/pickup
/pickup/action_topics
/execute_trajectory
/execute_trajectory/action_topics
/move_group
/move_group/fake_controller_joint_states
/move_group/action_topics
/move_group/display_planned_path
n__rviz_mohamed_G3_3500_3716_581445837892442824 8
/joint_state_publisher
/tf
/move_group
/joint_state_subscriber_1140_1727560059682
/attached_collision_object
/joint_states
/robot_state_publisher
/planning_scene_world
/trajectory_execution_event
/planning_scene
/joint_state_subscriber_5250_1728675836716

6.3. Computer Vision:

6.3.1. OAK-D Lite camera:

- What is Spatial AI?

Spatial AI is the ability of a visual AI system to make decisions based on two things.

1. **Visual Perception**: This is the ability of an AI to "see" and "interpret" its surrounding visually. For example, the system of a camera connected to a processor running a neural network for object detection can detect a cat, a person, or a car in the scene the camera is looking at.
2. **Depth Perception:** This is the ability of the AI to understand how far objects are. In computer vision jargon, "depth" simply means "how far."

The idea of Spatial AI is inspired by human vision. We use our eyes to interpret our surroundings. In addition, we use our two eyes (i.e. stereo vision) to perceive how far things are from us.

- What is OAK-D Lite?

- The OAK-D Lite is a small, affordable, and powerful smart camera.
- Combines depth and AI processing in one compact device.
- Perfect for developers, hobbyists, and professionals.

Figure 5.1

Key Highlights:

- **Three cameras:** Two for depth (stereo) and one for RGB vision.
- **Powered by Myriad X VPU:** Handles AI tasks and 3D depth processing.
- **USB 3.1 Gen1 connectivity:** Quick data transfer to your computer.

OAK-D and OAK-D-Lite are Spatial AI Cameras

At a high-level OAK-D and OAK-D-Lite consist of the following important components.

1. **A 4K RGB Camera:** The RGB camera placed at the center can capture very high-resolution 4k footage. Typically, this camera is used for visual perception.
2. **A Stereo pair**: This is a system of two cameras (the word "stereo" means two) used for depth perception.
3. **Intel® Myriad™ X Visual Processing Unit (VPU:** This is the **"brain" of the OAKs.** It is a powerful processor capable of running modern neural networks for visual perception and simultaneously creating a depth map from the stereo pair of images in real-time.

▪ OAK-D vs OAK-D Lite

In terms of features, both devices are nearly identical.

With an OAK-D Lite, you can do almost everything that an OAK-D is capable of. As a beginner, you should not notice much difference. However, there are a few specs that separate the two devices. OAK-D Lite is sleek, lighter, cheaper, and uses less power.

On the other hand, OAK-D is slightly more power-user-oriented.

Features/Specs	OAK-D	OAK-D Lite
RGB camera	12 Megapixel, 4k, up to 60 fps	12 Megapixel, 4k, up to 60 fps
Mono cameras	1280x800p, 120 fps, Global Shutter	640x480p, 120 fps, Global Shutter
Inertial Measurement Unit	Yes	No
USB-C, Power Jack	Yes, Yes	Yes, No
VPU	Myriad-X, 4 trillion ops/s	Myriad-X, 4 trillion ops/s

- Installation

The best part of using an **OAK-D** or **OAK-D Lite** is that there are no external hardware or software dependencies. It has integrated hardware, firmware, and software resulting in a seamless experience.

Depth-AI is the API (Application Programming Interface) through which we program the OAK-D. It's cross-platform, so you don't need to worry about what OS you have.

So, let's go ahead and install the API by firing up a terminal or PowerShell. It should only take about 30 seconds, given that you have a good internet connection.

```
git clone https://github.com/luxonis/depthai.git
cd depthai
python3 install_requirements.py
python3 depthai_demo.py
```

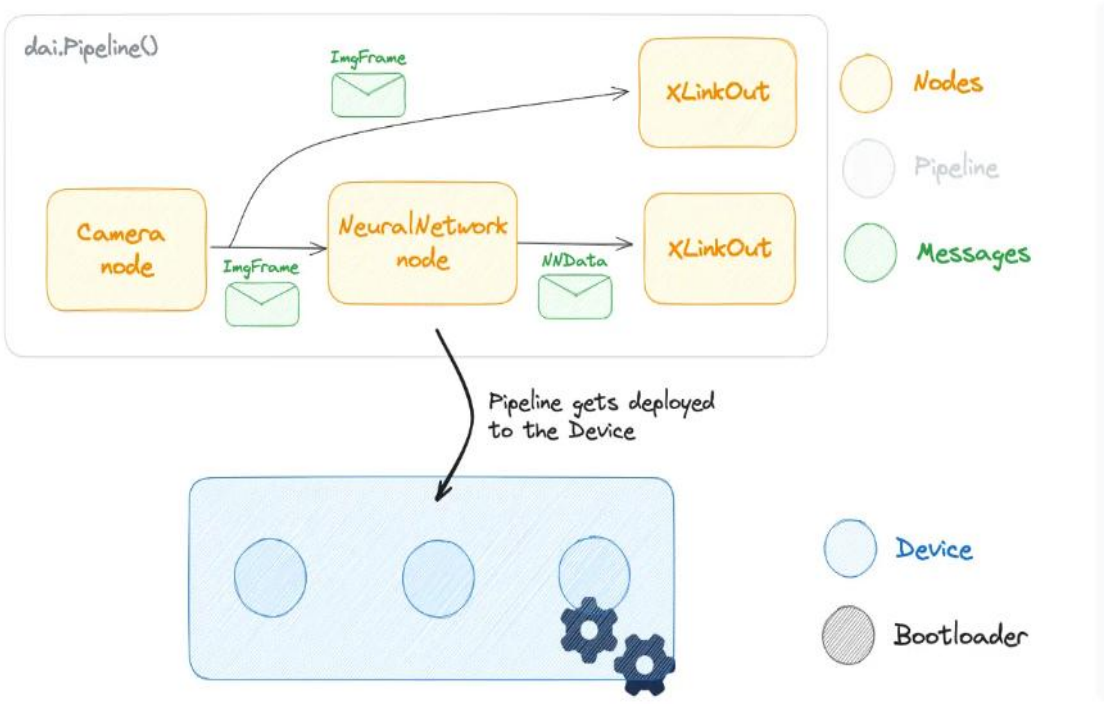
- How OAK-D Lite works:

  - Nodes are building blocks that handle specific functions like capturing camera data, running AI models, and outputting results
  - messages act as the medium for passing data between nodes.
  - The cameras work as part of a pipeline, where data is processed by linked nodes and deployed to the device."

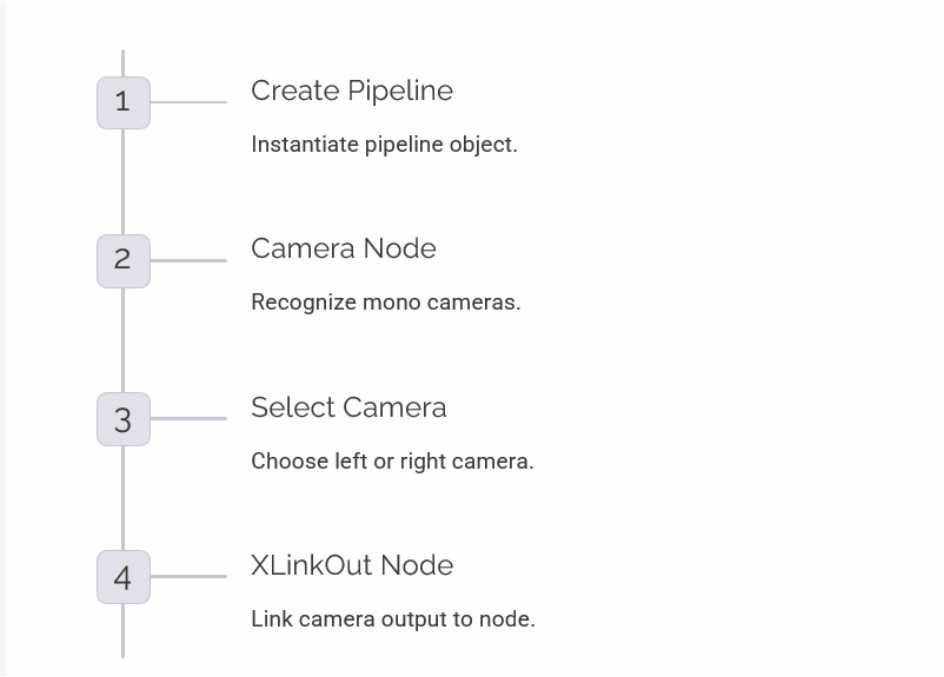
1) Camera Node: Captures data from the camera and produces an "ImgFrame" message.

2) Neural Network Node: Processes the camera data (e.g., runs AI models) and generates "NNData" messages.

3) XLinkOut Node: Outputs processed data (messages) to the host computer.



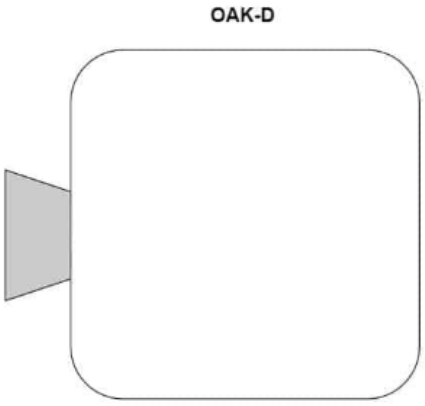
Steps:



1. **Create a pipeline**

A pipeline is a collection of nodes, and a node is a unit with some inputs and outputs. To understand the Depth-AI pipeline, let us go through the following illustrations. In the figures, we are showing what is happening inside the camera when the given command is executed. It's a very simple pipeline through which we capture the frames from the left mono camera.

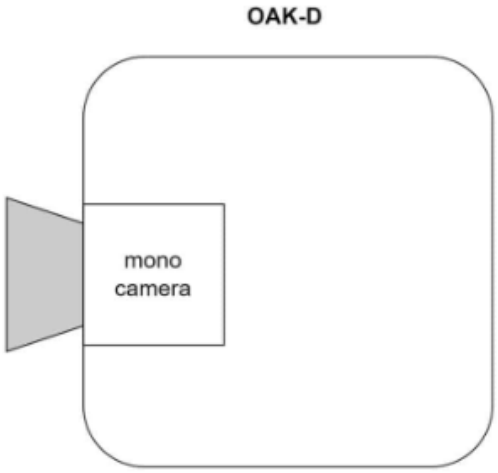
```
import depthai as dai
pipeline = dai.Pipeline()
```



2. Create the camera node

With the instruction below, we create the mono camera node. It does not do anything visually. It just recognizes the mono cameras.

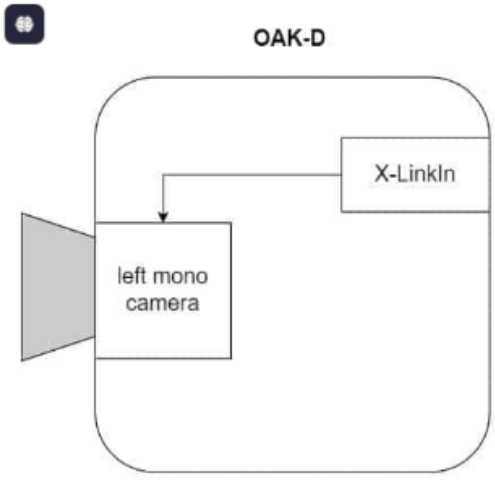
```
mono = pipeline.createMonoCamera()
```



3. Select the camera

To access a camera, we need to select it – in our case, the left camera. This is done by the setBoardSocket method. Internally it also creates an input node, X-LinkIn. X-Link is a mechanism using which the camera communicates with the host (computer).

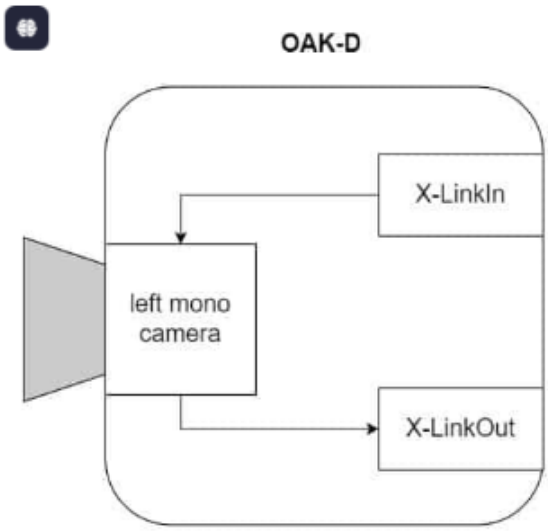
```
mono.setBoardSocket(dai.CameraBoardSocket.LEFT)
```



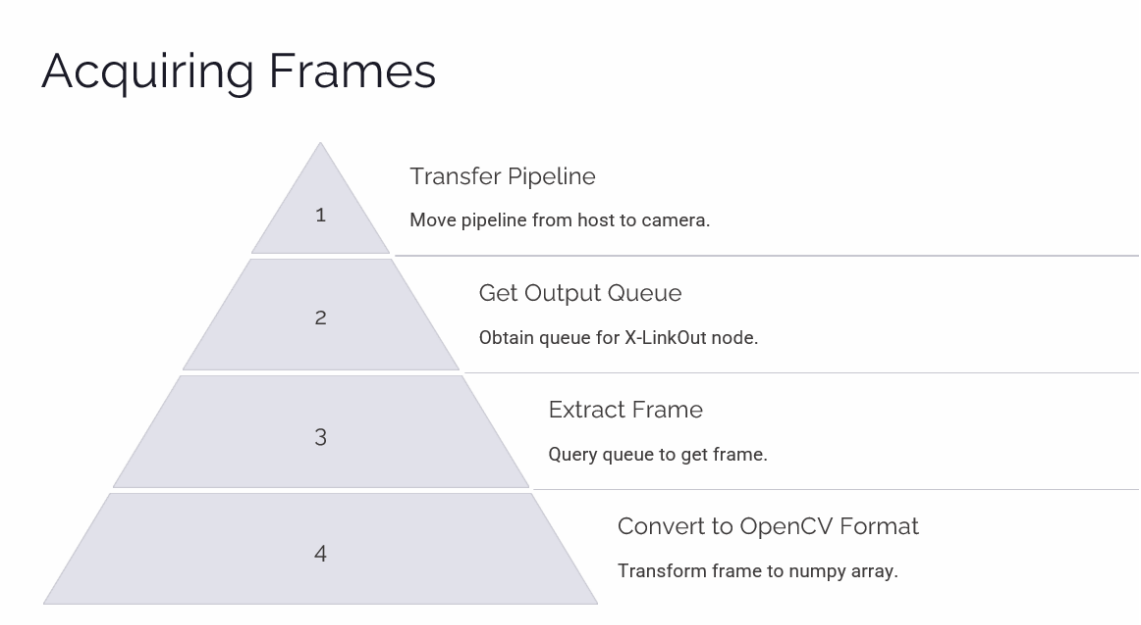
4. Create XLinkOut node and acquire frames

To get the output, we need to create the X-Link output node. The camera can have several other outputs, say another stream from the right mono camera or from the RGB camera or some other output that we don't need to worry about as of now. Hence, it has been named "left" so it does not conflict with others. Finally, we link the output of the mono camera by putting it as an input to the X-LinkOut node.

```
xout = pipeline.createXLinkOut()
xout.setStreamName("left")
mono.out.link(xout.input)
```



Acquiring Frames:



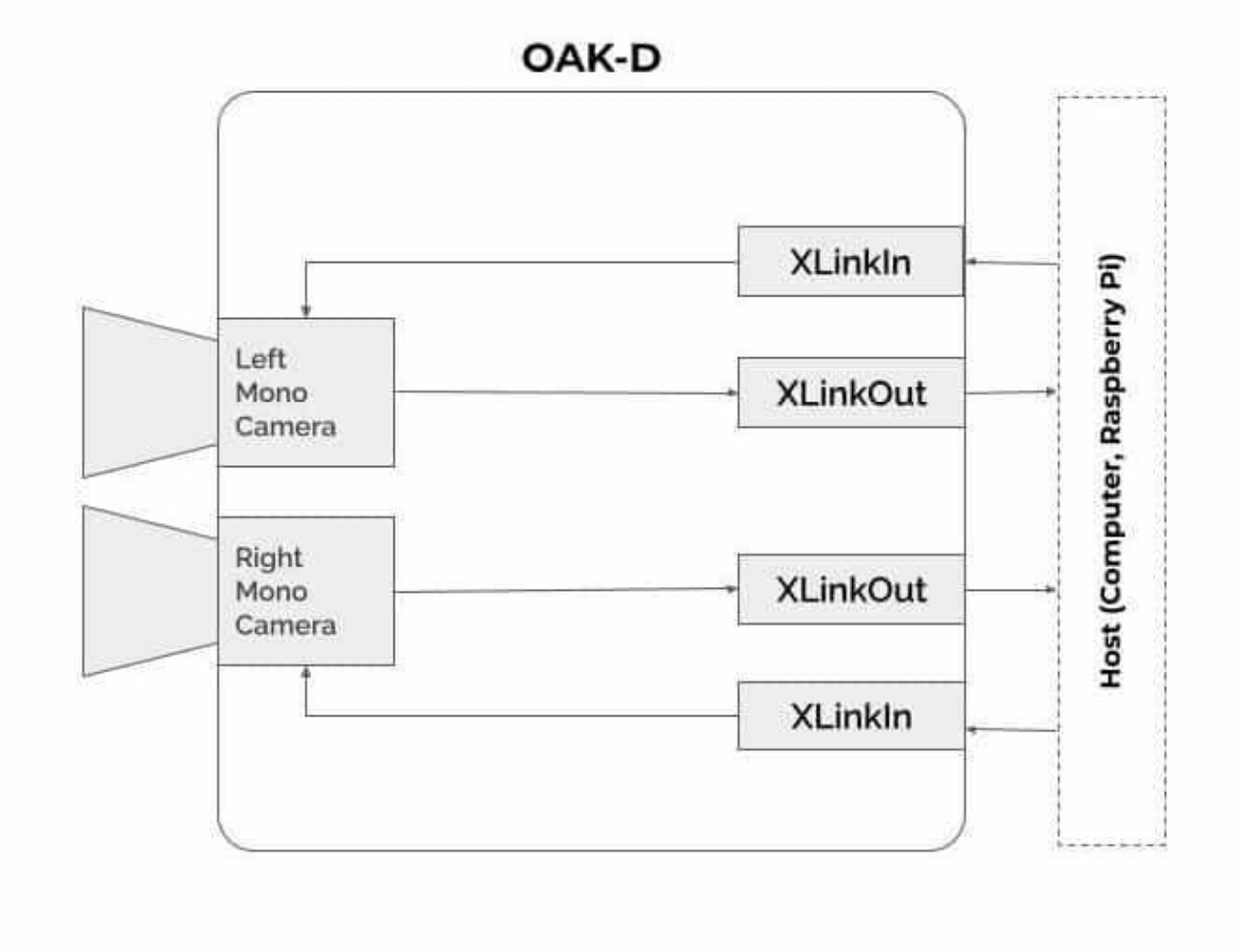
Although we showed that the previous instructions were readying the device, it was not processing anything. All the commands were running inside the host computer. You can think of it as a pre-processing step. We transfer the pipeline from the host computer to the camera with the following code snippet.

Now, we can acquire the output frame from the X-LinkOut node. Note that the output of the X-LinkOut node is not a single frame. In fact, it creates a queue that can store several frames. Which may be useful for certain applications requiring multiple frames. Video encoding is such an example. But in our case, we do not need multiple frames. So, let's keep it as the default, single frame for now. As you can see, the queue is obtained using the getOutputQueue method, where we specify the stream.

Next, we query the queue to extract the frame. At this point, the frame is transferred from the device to the host computer. The frame is of the class depthai.ImgFrame, which can have several types. To make it OpenCV friendly, we use the function getCvFrame, which returns the image as a numpy array. This concludes our basic pipeline.

```
with dai.Device(pipeline) as device:
  queue = device.getOutputQueue(name="left")
  frame = queue.get()
  imOut = frame.getCvFrame()
```

complete pipeline for both left and right cameras. We will use the outputs from the mono cameras to display the separate and merged views.



1. Import Libraries

```
import cv2
import depthai as dai
import numpy as np
```

2. Function to extract frame

It queries the frame from the queue, transfers it to the host, and converts it to a NumPy array.

```
def getFrame(queue):
  # Get frame from queue
  frame = queue.get()
  # Convert frame to OpenCV format and return
  return frame.getCvFrame()
```

3. Function to select mono camera

Here, a camera node is created for the pipeline. Then we set the camera resolution using the **setResolution** method. The **sensorResolution** class has the following attributes to choose from.

- THE_700_P (1280x720p)
- THE_800_P (1280x800p)
- THE_400_P (640x400p)
- THE_480_P (640x480p)

In our case, we are setting the resolution to 640x400p. The board socket is set to the left or right mono camera using the **isLeft** boolean.

```
def getMonoCamera(pipeline, isLeft):
  # Configure mono camera
  mono = pipeline.createMonoCamera()

  # Set Camera Resolution
    mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
  if isLeft:
      # Get left camera
      mono.setBoardSocket(dai.CameraBoardSocket.LEFT)
  else :
      # Get right camera
      mono.setBoardSocket(dai.CameraBoardSocket.RIGHT)
  return mono
```

Main Function

1. Pipeline and camera setup

We start by creating the pipeline and setting up the left and right mono cameras. The predefined function getMonoCamera creates the X-LinkIn node internally and returns mono camera output. The outputs from the cameras are then tied to the X-LinkOut nodes.

```
if __name__ == '__main__':
    pipeline = dai.Pipeline()

    # Set up left and right cameras
    monoLeft = getMonoCamera(pipeline, isLeft = True)
    monoRight = getMonoCamera(pipeline, isLeft = False)

    # Set output Xlink for left camera
    xoutLeft = pipeline.createXLinkOut()
    xoutLeft.setStreamName("left")

    # Set output Xlink for right camera
    xoutRight = pipeline.createXLinkOut()
    xoutRight.setStreamName("right")

    # Attach cameras to output Xlink
    monoLeft.out.link(xoutLeft.input)
    monoRight.out.link(xoutRight.input)
```

2. Transfer pipeline to the device

Once setup is ready, we transfer the pipeline into the device (camera). The queues for left and right camera outputs are defined with their respective names. The frame holding capacity is set with the maxSize argument, which is just a single frame in our case. A named window is also created later to display the output. The sideBySide variableis a boolean for toggling camera views (side by side or front view)

```
with dai.Device(pipeline) as device:
    # Get output queues.
    leftQueue = device.getOutputQueue(name="left", maxSize=1)
    rightQueue = device.getOutputQueue(name="right", maxSize=1)

    # Set display window name
    cv2.namedWindow("Stereo Pair")
    # Variable used to toggle between side by side view and one
    frame view.
    sideBySide = True
```

Main loop

Until now, we have created a pipeline, linked camera outputs to X-LinkOut nodes and obtained the queues. Now, it's time to query the frames and convert them to OpenCV-friendly numpy array format with the help of our predefined **getFrame** function.

For side by side view, the frames are concatenated horizontally with the help of the numpy **hstack** (horizontal stack)function. For overlapping output, we simply add the frames by reducing the intensity by a factor of 2.

The keyboard input **q** breaks the loop, and **t** toggles the display (Side-by-side and front views)

```
while True:
    # Get left frame
    leftFrame = getFrame(leftQueue)
    # Get right frame
    rightFrame = getFrame(rightQueue)

    if sideBySide:
        # Show side by side view
        imOut = np.hstack((leftFrame, rightFrame))
    else :
        # Show overlapping frames
        imOut = np.uint8(leftFrame/2 + rightFrame/2)

    # Display output image
    cv2.imshow("Stereo Pair", imOut)

    # Check for keyboard input
    key = cv2.waitKey(1)
    if key == ord('q'):
        # Quit when q is pressed
        break
    elif key == ord('t'):
        # Toggle display when t is pressed
        sideBySide = not sideBySide
```

Code:

1. Stereo vision:

```python
import cv2
import depthai as dai
import numpy as np

def getFrame(queue):
    # Get frame from queue
    frame = queue.get()
    # Convert frame to OpenCV format and return
    return frame.getCvFrame()

def getMonoCamera(pipeline, isLeft):
    # Configure mono camera
    mono = pipeline.createMonoCamera()

    # Set Camera Resolution
    mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)

    if isLeft:
        # Get left camera
        mono.setBoardSocket(dai.CameraBoardSocket.LEFT)
    else :
        # Get right camera
        mono.setBoardSocket(dai.CameraBoardSocket.RIGHT)
    return mono

if __name__ == '__main__':

    # Define a pipeline
    pipeline = dai.Pipeline()

    # Set up left and right cameras
    monoLeft = getMonoCamera(pipeline, isLeft = True)
    monoRight = getMonoCamera(pipeline, isLeft = False)

    # Set output Xlink for left camera
    xoutLeft = pipeline.createXLinkOut()
    xoutLeft.setStreamName("left")

    # Set output Xlink for right camera
    xoutRight = pipeline.createXLinkOut()
    xoutRight.setStreamName("right")

    # Attach cameras to output Xlink
    monoLeft.out.link(xoutLeft.input)
    monoRight.out.link(xoutRight.input)

    # Pipeline is defined, now we can connect to the device
    with dai.Device(pipeline) as device:

        # Get output queues.
        leftQueue = device.getOutputQueue(name="left", maxSize=1)
        rightQueue = device.getOutputQueue(name="right", maxSize=1)

        # Set display window name
        cv2.namedWindow("Stereo Pair")
        # Variable use to toggle between side by side view and one frame view.
        sideBySide = True

        while True:
            # Get left frame
            leftFrame = getFrame(leftQueue)
            # Get right frame
            rightFrame = getFrame(rightQueue)

            if sideBySide:
                # Show side by side view
                imOut = np.hstack((leftFrame, rightFrame))
            else :
                # Show overlapping frames
                imOut = np.uint8(leftFrame/2 + rightFrame/2)

            # Display output image
            cv2.imshow("Stereo Pair", imOut)

            # Check for keyboard input
            key = cv2.waitKey(1)
            if key == ord('q'):
                # Quit when q is pressed
                break
            elif key == ord('t'):
                # Toggle display when t is pressed
                sideBySide = not sideBySide
```

Output:

a) Both outputs from left and right camera:

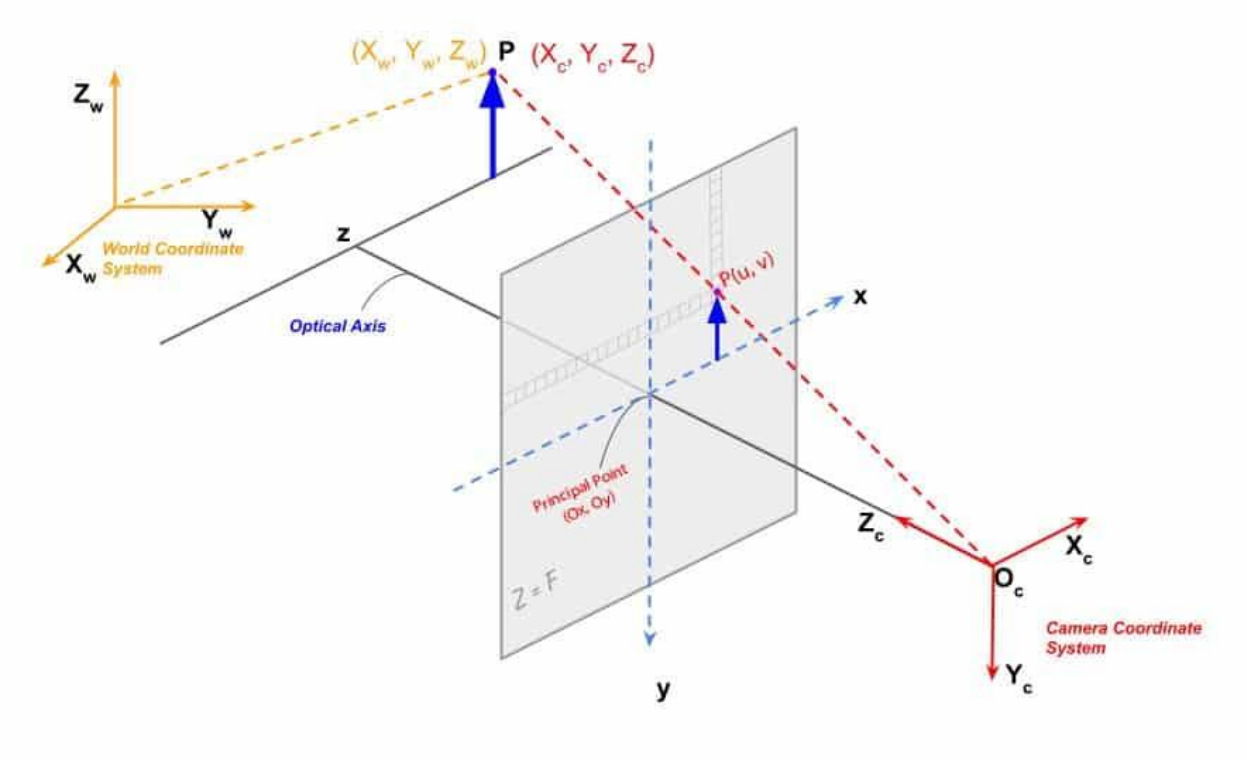
b) Output after merging the two lenses:

Depth Map generation:

- The Coordinate Systems

An image is a 2D projection of a 3D object from the real world to an image plane. We use the following coordinate systems to describe an imaging setup.

1. **World coordinate** (3D, unit: meters)
2. **Camera coordinate** (3D, unit: meters)
3. **Image plane coordinate** (2D, unit: pixels)

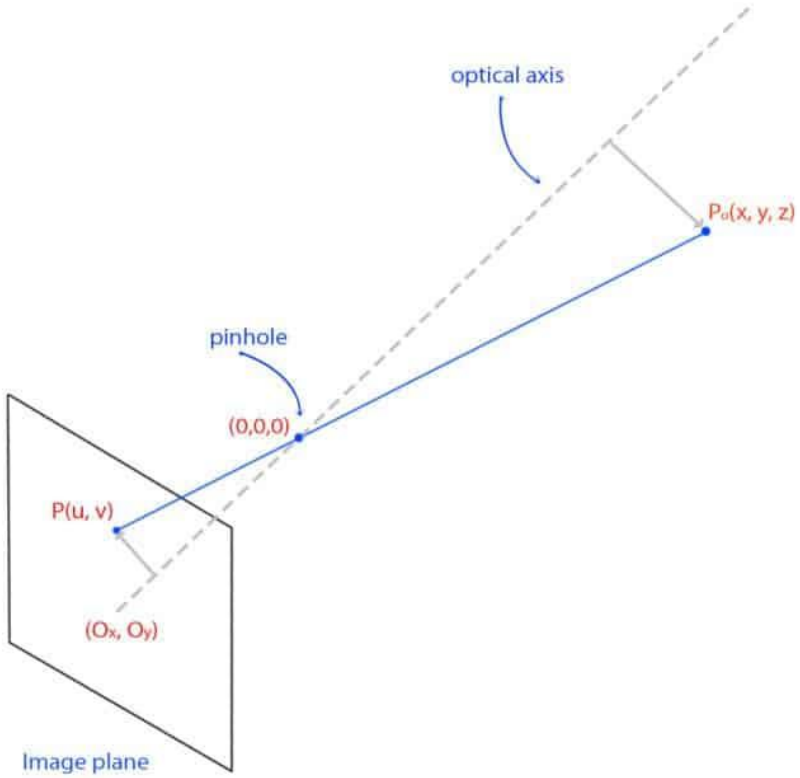


Mapping world coordinates to pixel coordinates tells us how far the object is from the camera perspective. To map or relate these coordinates, we need to know the camera's parameters (e.g., the focal length).

- How is Depth Calculated?

Let us consider that a camera captures the image of a real-world point Po.

where **P_o(x, y, z)** is the position of the point in the real world and **P(u, v)** in the image plane.



The perspective projection equations can be written as follows for a calibrated system.

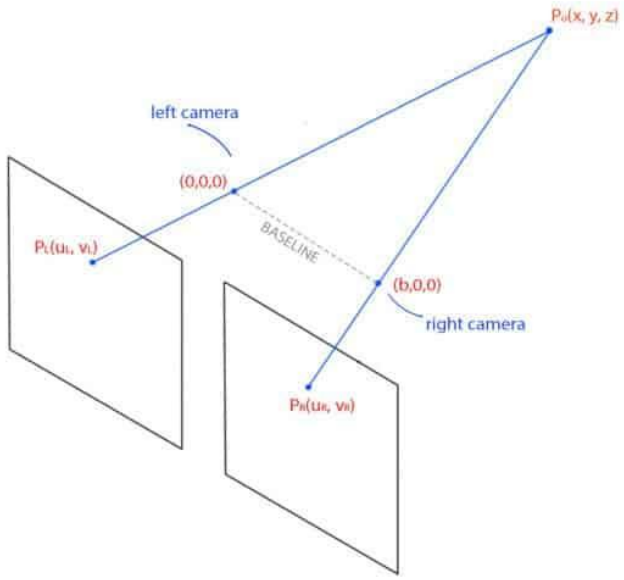
$$u = f_x \frac{x}{z} + o_x$$

$$v = f_y \frac{y}{z} + o_y$$

Where,

- f_x, f_y, **u, v,** O_x, O_y are known parameters in pixel units.
- The pixels in the image sensor may not be square, so we may have two different focal lengths f_x and f_y.
- (O_x, O_y) is the point where the optical axis intersects the image plane.

Since we have only two equations, we cannot find the three unknown variables, **x, y,** and **z**. To find them, we need two cameras. Another identical camera is positioned in a stereo system, as shown below. Both cameras are assumed to have no lens distortion



- The line between the centers of the cameras is called the **baseline.**
- $P_L(u_L, v_L)$ and $P_R(u_R, v_R)$ are projections of point P_o in the left and right image plane respectively.

$$x = \frac{b(u_l - o_x)}{u_L - u_R} \qquad y = \frac{bf_x(v_l - o_y)}{f_y(u_L - u_R)} \qquad \boxed{z = \frac{bf_x}{u_L - u_R}}$$

Therefore, Generally:

$$Depth = \frac{Bseline * Focal\ Length}{Disparity}$$

- Baseline (distance between the two stereo cameras).
- Focal length of the lenses.
- Resolution of the disparity map.
- Disparity range (minimum and maximum disparity values the stereo algorithm can compute).

Key Parameters

Let's break down the parameters for the OAK-D Lite:

1. **Baseline**:
    - The OAK-D Lite typically has a baseline of **7.5 cm (0.075 m)**.
2. **Focal Length**:
    - Using the default lens and configuration, the focal length in pixels is typically **882 pixels** for a 720p image. (This depends on the resolution used for stereo processing.)
3. **Disparity Range**:
    - Disparity values range from a **minimum of 1 pixel** (for very far objects) to a **maximum of ~96 pixels** (for very close objects).

- Calculate Minimum and Maximum Depth

Using the formula:

Maximum Depth (minimum disparity: 1 pixel):

$$Depth_{max} = \frac{0.075 * 882}{1} = 66.15\ meters$$

Minimum Depth (maximum disparity: 96 pixels):

$$Depth_{min} = \frac{0.075 * 882}{96} = 0.69\ meters$$

Practical Depth Range

1. **Minimum Depth (~0.5–0.7 m)**:
    - Objects closer than **~0.7 m** are challenging to measure due to stereo algorithm limitations (disparity saturates).
2. **Maximum Depth (~10–20 m)**:
    - In practice, the camera can reliably compute depths for objects up to **10–20 meters**, depending on lighting, texture, and stereo accuracy.
    - Beyond this range, the disparity values become too small to distinguish depth effectively, and accuracy decreases.

Therefore;

- **Depth Range**: The OAK-D Lite can compute depth reliably from approximately **0.7 m** to **10–20 m**.
- **Optimal Range**: For best accuracy, use the camera in the **0.7–4 m range**.

- Correct Interpretation of the Provided Images

1. **Close Objects (Red Areas)**:
    - In the second and third images, the red regions correspond to close objects, such as:
        - A person's face (second image).
        - Cups in front of the laptop (third image).
    - These regions have **high disparity values** and are measured as being close to the camera.
2. **Farther Objects (Blue Areas)**:
    - The background regions (walls, far-away objects) show **low disparity values** and are measured as being far from the camera.
3. **Too Close to the Camera**:
    - If an object is extremely close to the camera (closer than the stereo system's minimum range), the disparity may saturate (too high to calculate accurately) or produce artifacts.

Code

2. Disparity:

```python
import cv2
import depthai as dai
import numpy as np

def getFrame(queue):
  # Get frame from queue
  frame = queue.get()
  # Convert frame to OpenCV format and return
  return frame.getCvFrame()


def getMonoCamera(pipeline, isLeft):
  # Configure mono camera
  mono = pipeline.createMonoCamera()

  # Set Camera Resolution
  mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)

  if isLeft:
      # Get left camera
      mono.setBoardSocket(dai.CameraBoardSocket.LEFT)
  else :
      # Get right camera
      mono.setBoardSocket(dai.CameraBoardSocket.RIGHT)
  return mono


def getStereoPair(pipeline, monoLeft, monoRight):
    # Configure stereo pair for depth estimation
    stereo = pipeline.createStereoDepth()
    # Checks occluded pixels and marks them as invalid
    stereo.setLeftRightCheck(True)

    # Configure left and right cameras to work as a stereo pair
    monoLeft.out.link(stereo.left)
    monoRight.out.link(stereo.right)

    return stereo
```

```
39
40  def mouseCallback(event,x,y,flags,param):
41      global mouseX, mouseY
42      if event == cv2.EVENT_LBUTTONDOWN:
43          mouseX = x
44          mouseY = y
45
46  if __name__ == '__main__':
47
48      mouseX = 0
49      mouseY = 640
50      # Start defining a pipeline
51      pipeline = dai.Pipeline()
52
53      # Set up left and right cameras
54      monoLeft = getMonoCamera(pipeline, isLeft = True)
55      monoRight = getMonoCamera(pipeline, isLeft = False)
56
57      # Combine left and right cameras to form a stereo pair
58      stereo = getStereoPair(pipeline, monoLeft, monoRight)
59
60
61      # Set XlinkOut for disparity, rectifiedLeft, and rectifiedRight
62      xoutDisp = pipeline.createXLinkOut()
63      xoutDisp.setStreamName("disparity")
64
65      xoutRectifiedLeft = pipeline.createXLinkOut()
66      xoutRectifiedLeft.setStreamName("rectifiedLeft")
67
68      xoutRectifiedRight = pipeline.createXLinkOut()
69      xoutRectifiedRight.setStreamName("rectifiedRight")
70
71      stereo.disparity.link(xoutDisp.input)
72
73      stereo.rectifiedLeft.link(xoutRectifiedLeft.input)
74      stereo.rectifiedRight.link(xoutRectifiedRight.input)
75
76      # Pipeline is defined, now we can connect to the device
```

```
78      with dai.Device(pipeline) as device:
79
80
81          # Output queues will be used to get the rgb frames and nn data from the outputs defined above
82          disparityQueue = device.getOutputQueue(name="disparity", maxSize=1, blocking=False)
83          rectifiedLeftQueue = device.getOutputQueue(name="rectifiedLeft", maxSize=1, blocking=False)
84          rectifiedRightQueue = device.getOutputQueue(name="rectifiedRight", maxSize=1, blocking=False)
85
86
87          # Calculate a multiplier for colormapping disparity map
88          disparityMultiplier = 255 / stereo.getMaxDisparity()
89
90          cv2.namedWindow("Stereo Pair")
91          cv2.setMouseCallback("Stereo Pair", mouseCallback)
92
93          # Variable use to toggle between side by side view and one frame view.
94          sideBySide = False
95
```

```
        while True:

            # Get disparity map
            disparity = getFrame(disparityQueue)

            # Colormap disparity for display
            disparity = (disparity * disparityMultiplier).astype(np.uint8)
            disparity = cv2.applyColorMap(disparity, cv2.COLORMAP_JET)

            # Get left and right rectified frame
            leftFrame = getFrame(rectifiedLeftQueue);
            rightFrame = getFrame(rectifiedRightQueue)

            if sideBySide:
                # Show side by side view
                imOut = np.hstack((leftFrame, rightFrame))
            else :
                # Show overlapping frames
                imOut = np.uint8(leftFrame/2 + rightFrame/2)


            imOut = cv2.cvtColor(imOut,cv2.COLOR_GRAY2RGB)

            imOut = cv2.line(imOut, (mouseX, mouseY), (1280, mouseY), (0, 0, 255), 2)
            imOut = cv2.circle(imOut, (mouseX, mouseY), 2, (255, 255, 128), 2)
            cv2.imshow("Stereo Pair", imOut)
            cv2.imshow("Disparity", disparity)

            # Check for keyboard input
            key = cv2.waitKey(1)
            if key == ord('q'):
                # Quit when q is pressed
                break
            elif key == ord('t'):
                # Toggle display when t is pressed
                sideBySide = not sideBySide

```

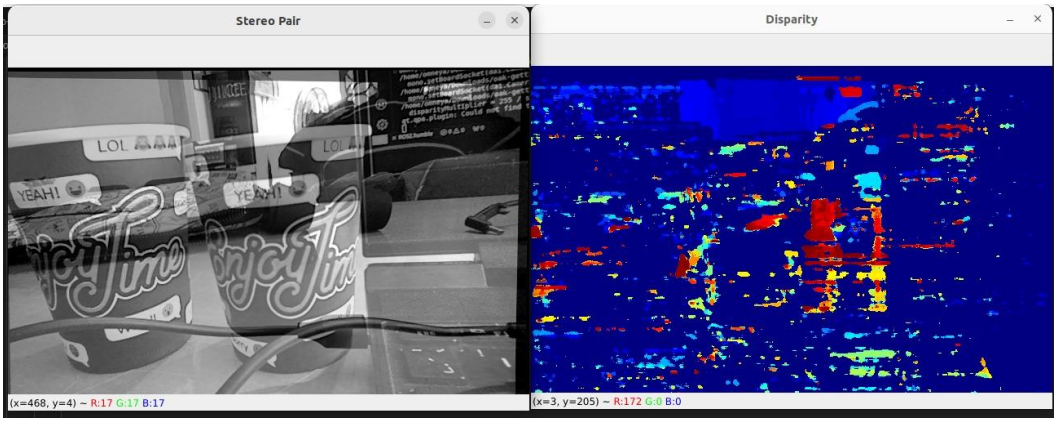
Depth:

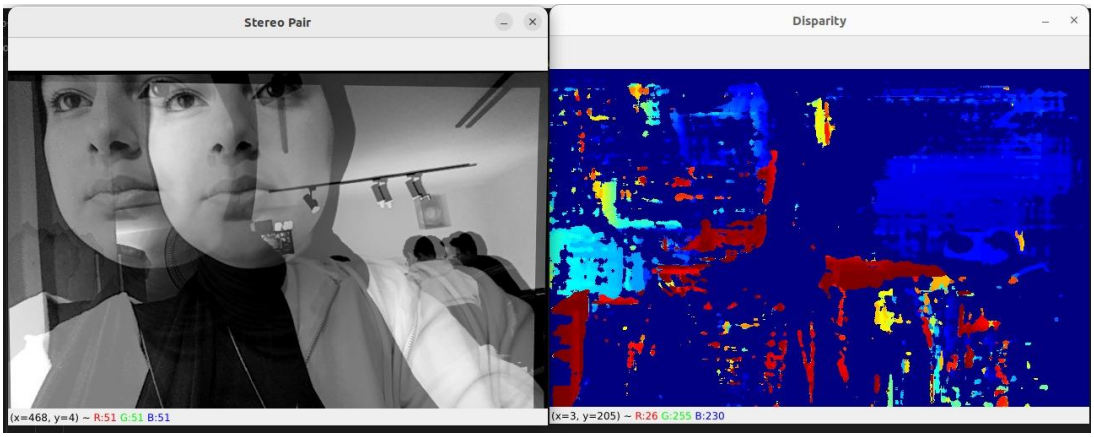
```python
import cv2
import numpy as np

def disparity_to_depth(disparity_map, baseline, focal_length):
    """
    Converts a disparity map to a depth map.

    Parameters:
        disparity_map (numpy.ndarray): The disparity image (grayscale or single-channel).
        baseline (float): The baseline distance between the stereo cameras (in meters).
        focal_length (float): The focal length of the camera (in pixels).

    Returns:
        numpy.ndarray: The depth map (in meters).
    """
    # Avoid division by zero by setting zero disparities to a small value
    disparity_map = np.where(disparity_map > 0, disparity_map, 1e-6)

    # Compute depth using the formula: Depth = (Baseline * Focal Length) / Disparity
    depth_map = (baseline * focal_length) / disparity_map

    return depth_map

# Example usage
if __name__ == "__main__":
    # Load the disparity map (example: 8-bit or 16-bit image)
    disparity_map = cv2.imread("disparity_image.png", cv2.IMREAD_UNCHANGED)

    # Camera parameters (specific to OAK-D Lite or your setup)
    BASELINE = 0.075  # Baseline in meters (7.5 cm)
    FOCAL_LENGTH = 882  # Focal length in pixels (adjust for your resolution)

    # Convert disparity to depth
    depth_map = disparity_to_depth(disparity_map, BASELINE, FOCAL_LENGTH)

    # Normalize depth map for visualization (optional)
    normalized_depth = cv2.normalize(depth_map, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

    # Display and save depth map
    cv2.imshow("Depth Map", normalized_depth)
    cv2.imwrite("depth_map.png", normalized_depth)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Output:

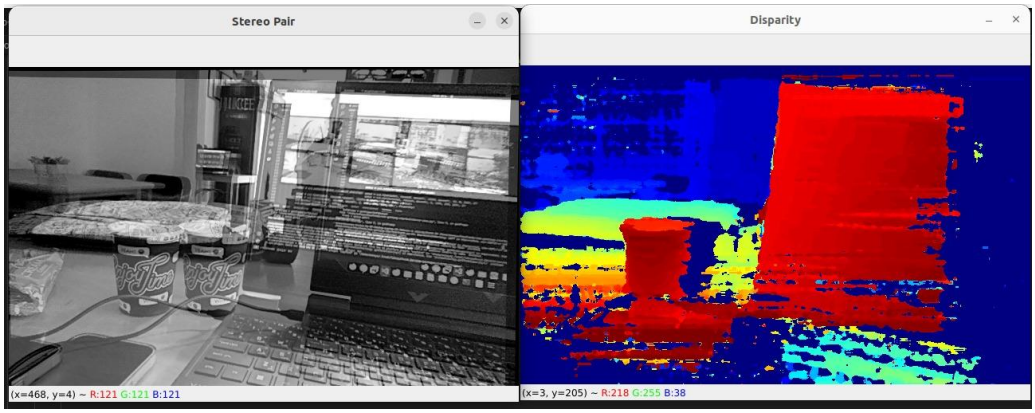
If we are too close to camera the depth is inaccurate as there is a specific range the camera can get the depth starting from that distance

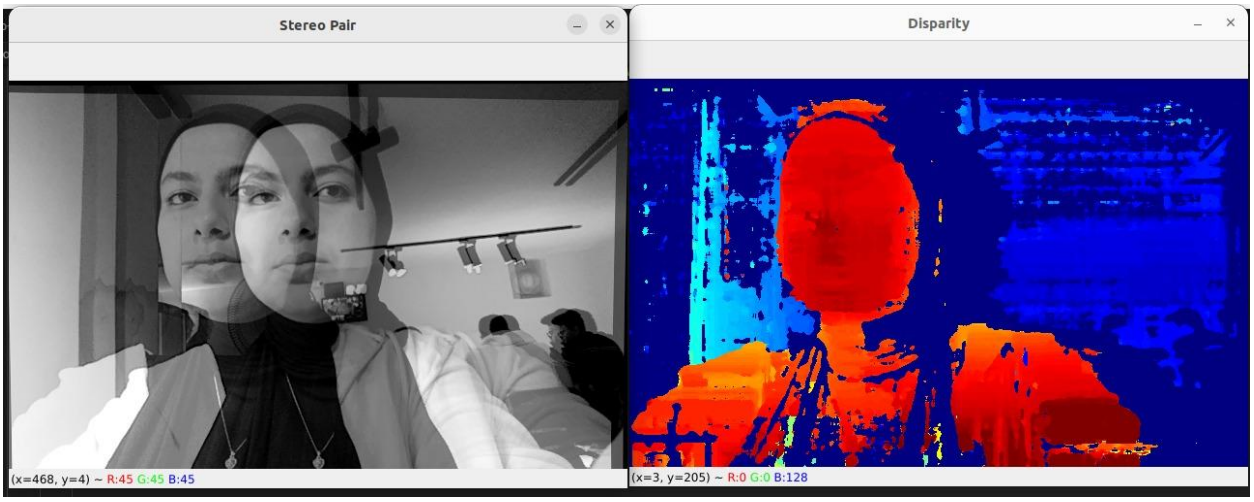
a) If too close:





b) If far with a suitable range:





Stereo Pair (Left Panel):

- These images represent the grayscale stereo pair captured by the OAK-D Lite camera.
- The stereo image overlap highlights how the left and right cameras perceive the scene differently, with disparities most noticeable for closer objects.

Disparity Map (Right Panel):

- The right panel shows the computed disparity maps using the stereo pair.
- The disparity map uses a color-coded scheme to indicate the disparity (and indirectly depth):
    - ✓ **Red regions** indicate higher disparity values, meaning objects closer to the camera.
    - ✓ **Blue regions** indicate lower disparity values, corresponding to objects further away from the camera.
    - ✓ Areas without disparity information (e.g., very smooth surfaces or occluded regions) appear black or may be noisy.

- Camera interaction with ROS

Install the pre-requisites.

```
sudo wget -qO- https://raw.githubusercontent.com/luxonis/depthai-ros/main/install_dependencies.sh | sudo bash

sudo apt install libopencv-dev

# if you don't have rosdep installed and not initialized please execute the following steps:

sudo apt install python-rosdep(melodic) or sudo apt install python3-rosdep

sudo rosdep init

rosdep update

# you will need this to import external repos

sudo apt install python3-vcstool
```

Install the main repos and build the code

```
# The following setup procedure assumes you have cmake version >= 3.10.2 and OpenCV version >= 4.0.0

mkdir -p luxonis_depthai_ws/src

cd luxonis_depthai_ws

wget https://raw.githubusercontent.com/luxonis/depthai-ros/main/underlay.repos

vcs import src < underlay.repos

rosdep install --from-paths src --ignore-src -r -y

source /opt/ros/foxy/setup.bash

colon build

source install/setup.bash
```

Launch an example node.

```
ros2 launch depthai_examples stereo.launch.py
```

```
ros2 topic list

# this will show output like this
/camera/left/image_raw
/camera/right/image_raw
/clicked_point
/initialpose
/left/camera_info
/move_base_simple/goal
/parameter_events
/right/camera_info
/right/image_rect
/robot_description
/rosout
/stereo/points
/tf
/tf_static
```

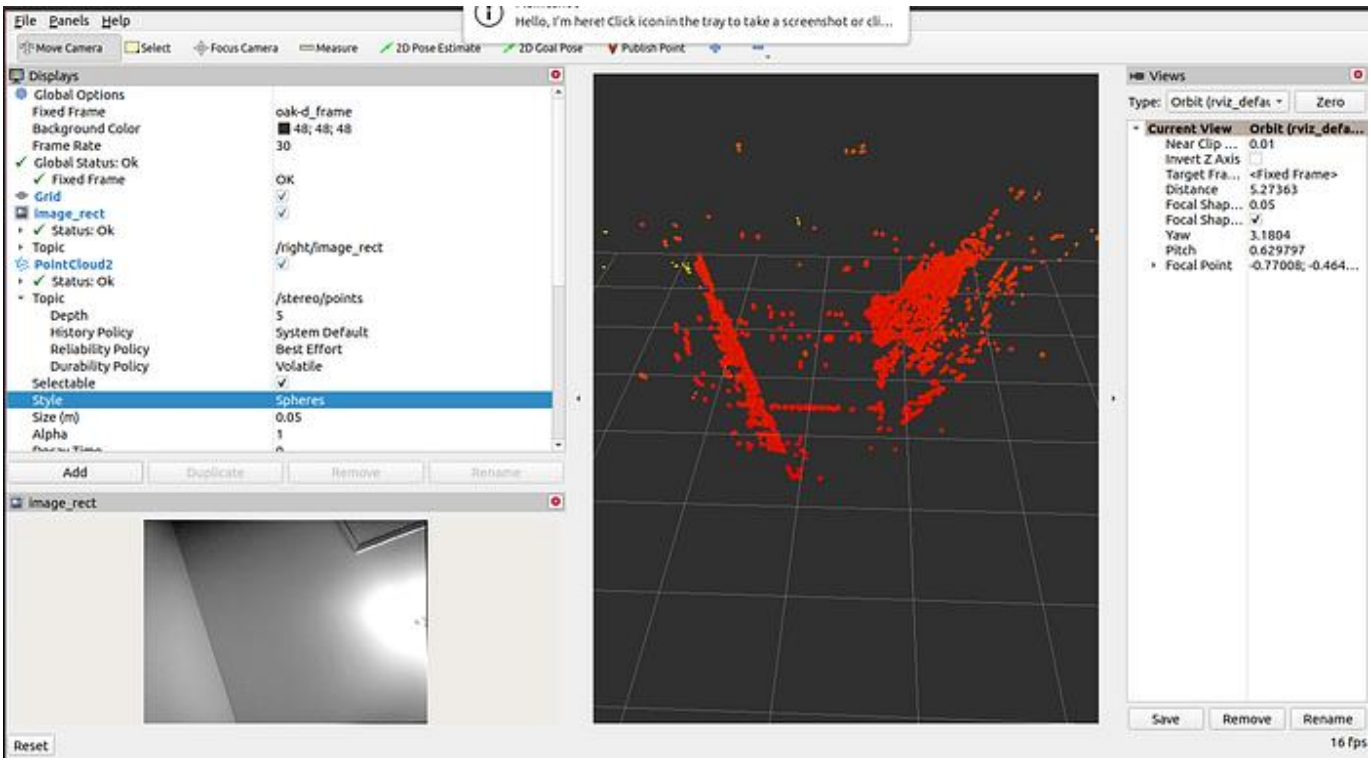
You can echo the topic to see if the data is actually being published.

```
ros2 topic echo /stereo/points
```

Host Machine

Once everything is confirmed on RPi, you can go to your host machine [laptop/desktop] where you have ROS2 GUI tools installed and launch the RViz2. You can either clone the github repo again there or open the rviz config file from RPi if you have network access to browse files on RPi So you can navigate to the src/luxonis/depthai-ros-examples/depthai_examples/rviz/stereoPointCloud.rviz folder and open that file in the RViz2.

now should something like this on your RViz2 screen with camera image stream and Pointcloud data showing up. The Pointcloud data may be very small, you can adjust the size of points from 0.01 to 0.05 so it can be seen better.



6.3.2. Wound and Stitched Detection:

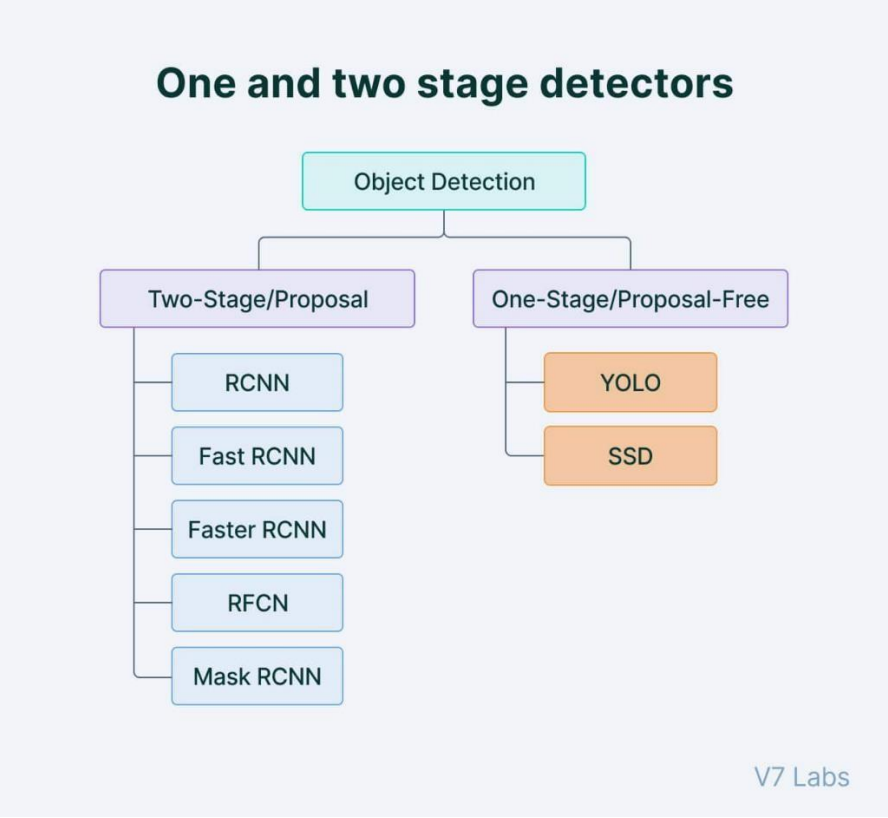
- What is object detection?

Object detection is a computer vision task that involves identifying and locating objects in images or videos. It is an important part of many applications, such as surveillance, self-driving cars, or robotics. Object detection algorithms can be divided into two main categories: single-shot detectors and two-stage detectors.

One of the earliest successful attempts to address the object detection problem using deep learning was the R-CNN (Regions with CNN features) model.

This model used a combination of region proposal algorithms and convolutional neural networks (CNNs) to detect and localize objects in images.

Object detection algorithms are broadly classified into two categories based on how many times the same input image is passed through a network.



a) Single-shot object detection

Single-shot object detection uses a single pass of the input image to make predictions about the presence and location of objects in the image. It processes an entire image in a single pass, making them computationally efficient.

However, single-shot object detection is generally less accurate than other methods, and it's less effective in detecting small objects. Such algorithms can be used to detect objects in real time in resource-constrained environments.

YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process an image. We will dive deeper into the YOLO model in the next section.

b) Two-shot object detection

Two-shot object detection uses two passes of the input image to make predictions about the presence and location of objects. The first pass is used to generate a set of proposals or potential object locations, and the second pass is used to refine these proposals and make final predictions.

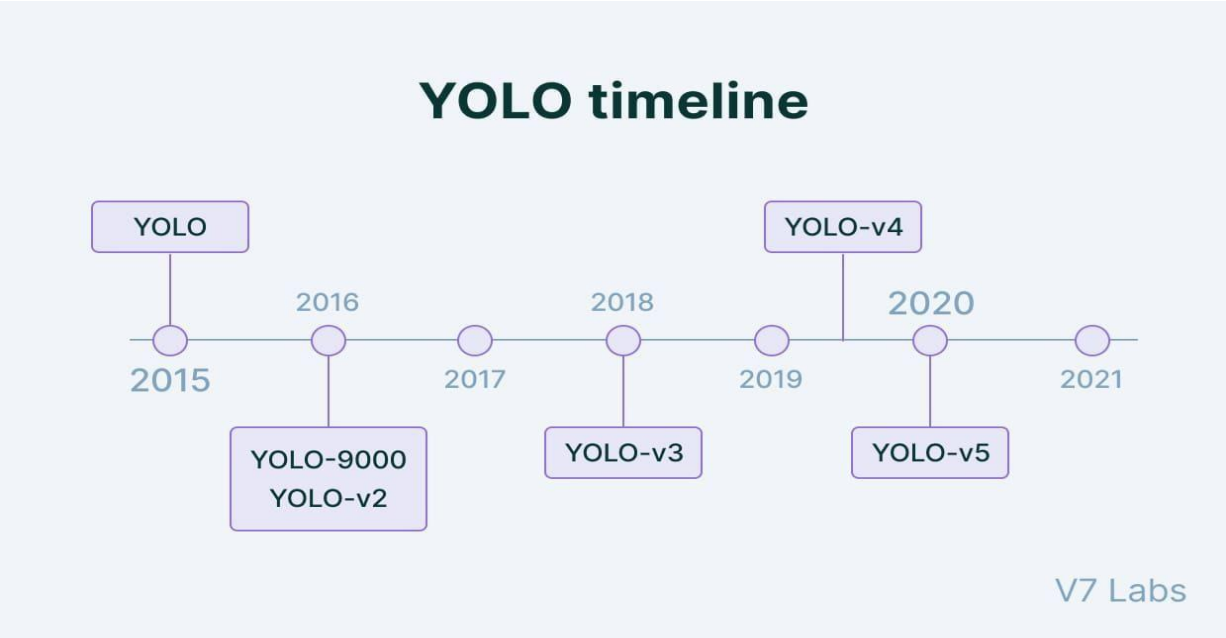
This approach is more accurate than single-shot object detection but is also more computationally expensive.

Overall, the choice between single-shot and two-shot object detection depends on the specific requirements and constraints of the application.

Generally, single-shot object detection is better suited for real-time applications, while two-shot object detection is better for applications where accuracy is more important.

- What is YOLO?

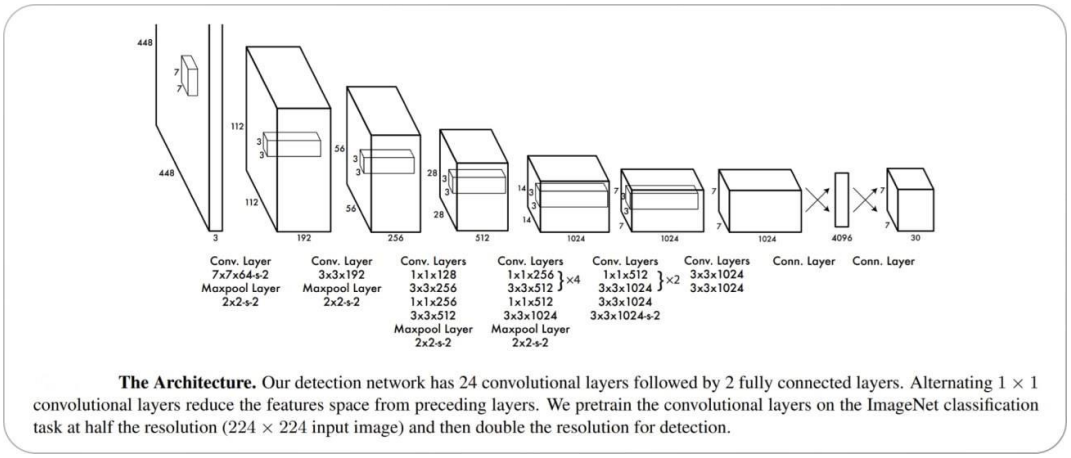
You Only Look Once (YOLO) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once.



- How does YOLO work? YOLO Architecture

The YOLO algorithm takes an image as input and then uses a simple deep convolutional neural network to detect objects in the image. The architecture of the

CNN model that forms the backbone of YOLO is shown below.



The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

The first 20 convolution layers of the model are pre-trained using ImageNet by plugging in a temporary average pooling and fully connected layer. Then, this pretrained model is converted to perform detection since previous research showcased that adding convolution and connected layers to a pre-trained network improves performance. YOLO's final fully connected layer predicts both class probabilities and bounding box coordinates.

YOLO divides an input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the predicted box is.

YOLO predicts multiple bounding boxes per grid cell. At training time, we only want one bounding box predictor to be responsible for each object. YOLO assigns one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at forecasting certain sizes, aspect ratios, or classes of objects, improving the overall recall score.

In our project we used Yolov8:

Why yolov8?

- User-friendly API (Command Line + Python).
- Faster and More Accurate.
- Supports
  - ✓ Object Detection,
  - ✓ Instance Segmentation,
  - ✓ Image Classification.
- Extensible to all previous versions.
- New Backbone network.
- New Anchor-Free head.
- New Loss Function.

YOLOv8 is also highly efficient and flexible supporting numerous export formats, and the model can run on CPUs & GPUs.

- Models Available in YOLOv8

There are five models in each category of YOLOv8 models for detection, segmentation, and classification. YOLOv8 Nano is the fastest and smallest, while YOLOv8 Extra Large (YOLOv8x) is the most accurate yet the slowest among them.

YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	YOLOv8x

- What is YOLOv8?

YOLOv8 is the latest family of YOLO based Object Detection models from Ultralytics providing state-of-the-art performance.

Leveraging the previous YOLO versions, the YOLOv8 model is faster and more accurate while providing a unified framework for training models for performing

- Object detection
- Segmentation
- Classification

As of writing this, a lot of features are yet to be added to the Ultralytics YOLOv8 repository. This includes the complete set of export features for the trained models. Also, Ultralytics will release a paper on Arxiv comparing YOLOv8 with other stateof-the-art vision models.

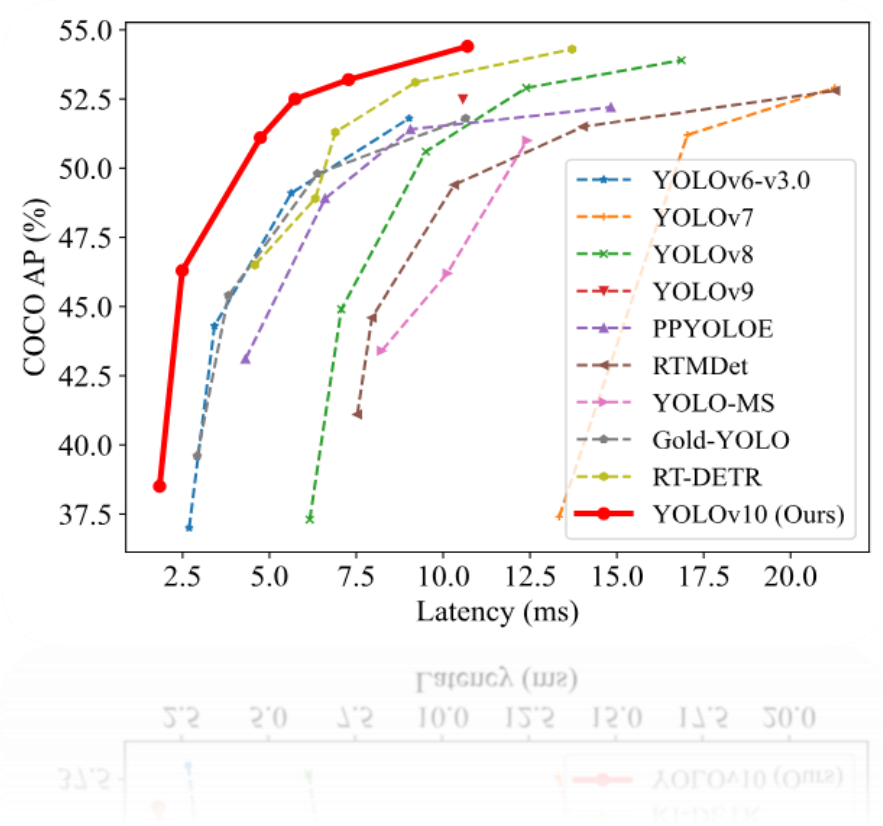
Ultralytics have released a completely new repository for YOLO Models.

It is built as a unified framework for training Object Detection, Instance Segmentation, and Image Classification models.

New Backbone network.

- New Anchor-Free head.
- New Loss Function.

YOLOv8 is also highly efficient and flexible supporting numerous export formats, and the model can run on CPUs & GPUs.



Steps:

1. Data Collection
2. Annotation
3. Configuration:
   Choose the YOLO version
4. Model Training
5. Testing and Validation

Training on collab:

```
!nvidia-smi
```

```
Sun Jan 19 18:19:28 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   42C    P8              9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

```
# Pip install method (recommended)

!pip install ultralytics
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.63-py3-none-any.whl.metadata (35 kB)
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (1.26.4)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (3.10.0)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (4.10.0.84)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (11.1.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.32.3)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (1.13.1)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.5.1+cu121)
```

```
[ ] from ultralytics import YOLO
    from IPython.display import display, Image
    import os
    from IPython import display
    display.clear_output()

    import ultralytics
    ultralytics.checks()
```

```
Ultralytics 8.3.63 🚀 Python-3.11.11 torch-2.5.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 30.8/112.6 GB disk)
```

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="HT93rWRTrBLyjJk94NHI")
project = rf.workspace("omneya-haytham-we3yj").project("wound_stitched")
version = project.version(1)
dataset = version.download("yolov8")
```

```
Collecting roboflow
  Downloading roboflow-1.1.51-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from roboflow) (2024.12.14)
Collecting idna==3.7 (from roboflow)
  Downloading idna-3.7-py3-none-any.whl.metadata (9.9 kB)
Requirement already satisfied: cycler in /usr/local/lib/python3.11/dist-packages (from roboflow) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.4.8)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from roboflow) (3.10.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.26.4)
Requirement already satisfied: opencv-python-headless==4.10.0.84 in /usr/local/lib/python3.11/dist-packages (from roboflow) (4.10.0.84)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.11/dist-packages (from roboflow) (11.1.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.8.2)
Collecting python-dotenv (from roboflow)
  Downloading python_dotenv-1.0.1-py3-none-any.whl.metadata (23 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.32.3)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.17.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.3.0)
```

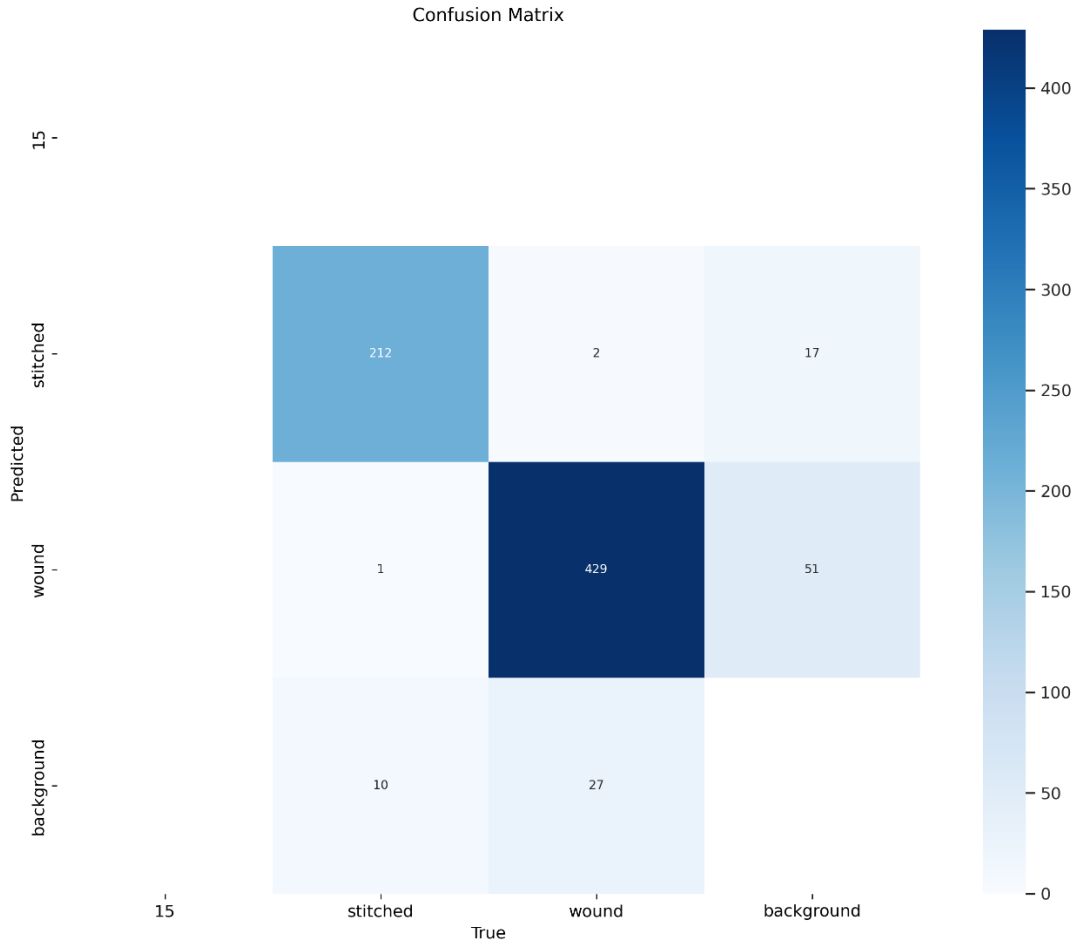
```
]  from ultralytics import YOLO

   from IPython.display import display, Image
```

```
!yolo task=detect mode=train model=yolov8m.pt data={dataset.location}/data.yaml epochs=100 imgsz=800
```

```
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 20/20 [00:13<00:00,  1.53it/s]
                 all        622        681      0.941      0.912      0.954      0.594

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     93/100      10.4G     0.9619     0.4885      1.345         11        800: 100% 136/136 [01:57<00:00,  1.16it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 20/20 [00:13<00:00,  1.47it/s]
                 all        622        681      0.935       0.92      0.954      0.597

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     94/100      10.4G     0.9489     0.4815      1.336         13        800: 100% 136/136 [01:57<00:00,  1.16it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 20/20 [00:12<00:00,  1.56it/s]
                 all        622        681      0.926      0.917      0.947      0.596

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     95/100      10.4G     0.9323     0.4784       1.33         15        800: 100% 136/136 [01:56<00:00,  1.16it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 20/20 [00:12<00:00,  1.55it/s]
                 all        622        681      0.938       0.92       0.95      0.596

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     96/100      10.4G     0.9274     0.4705      1.324         14        800: 100% 136/136 [01:57<00:00,  1.16it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 20/20 [00:13<00:00,  1.51it/s]
                 all        622        681      0.924      0.926      0.953      0.597

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     97/100      10.4G      0.911     0.4614      1.308         13        800: 100% 136/136 [01:57<00:00,  1.16it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 20/20 [00:13<00:00,  1.48it/s]
                 all        622        681      0.929      0.917      0.952      0.599

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100      10.4G     0.8961     0.4598      1.301         11        800: 100% 136/136 [01:56<00:00,  1.16it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 20/20 [00:13<00:00,  1.54it/s]
                 all        622        681      0.932      0.922      0.956      0.604
```

Output:

Confusion matrix:

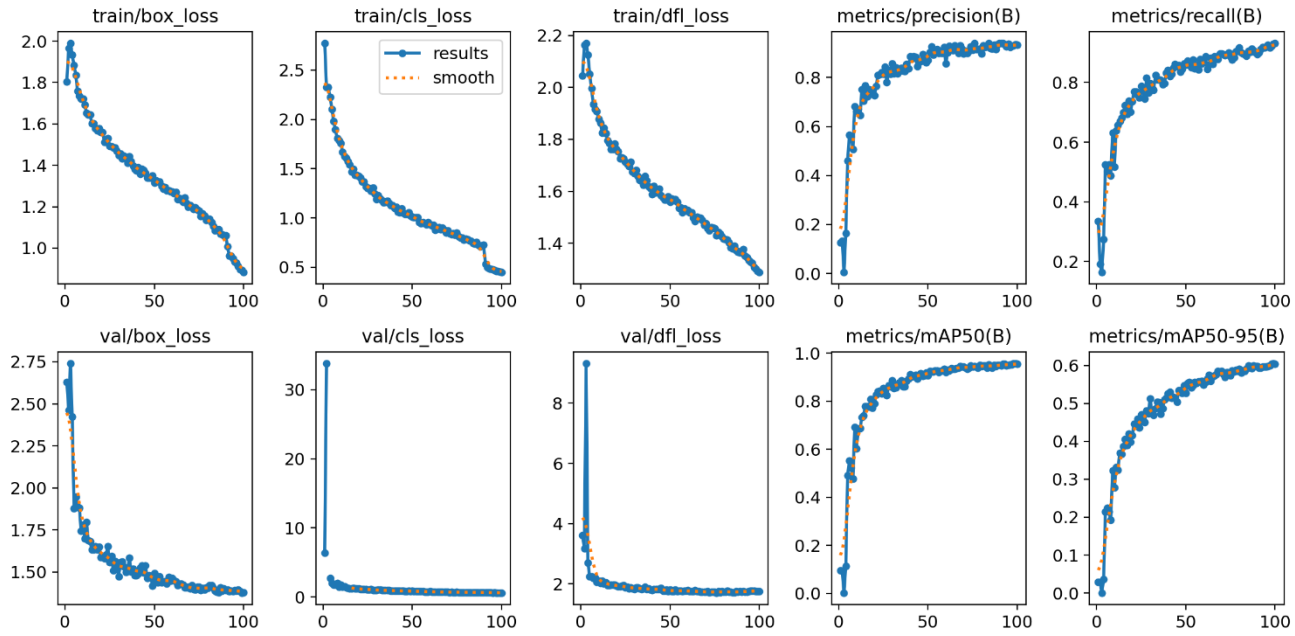


This figure visualizes the model's classification performance across different classes (stitched, wound, and background).

1. **Structure**:
    - Rows represent the predicted classes (stitched, wound, background).
    - Columns represent the actual (true) classes.

2. **Key Observations**:
    - **Stitched Class**:
        - 212 correct predictions, 2 misclassified as wounds, and 17 as background.
        - **Accuracy**: High, with minor confusion with background.
    - **Wound Class**:
        - 429 correct predictions, 1 misclassified as stitched, and 51 as background.
        - **Accuracy**: Excellent, though some overlap with background.
    - **Background Class**:
        - 27 correct predictions, 10 misclassified as stitched, and minimal as wounds.
        - **Accuracy**: Lower, likely due to imbalance or subtle features.
3. **Precision and Recall for Each Class**:
    - Precision and recall for wounds and stitches are high, indicating reliable detection for these critical categories.
    - Background class shows room for improvement, suggesting potential enhancements like more diverse training samples.

Training and Validation Metrics



This figure includes multiple subplots that monitor the training process and evaluate model performance during training and validation.

1. **Training Box Loss (Top Left)**:
    - Represents the model's performance in learning the correct placement of bounding boxes.
    - **Trend**: A consistent decrease over epochs indicates the model is improving its localization capabilities.
    - **Observation**: Loss reduced from approximately 2.0 to below 1.0, demonstrating effective bounding box optimization.
2. **Training Classification Loss (Top Center)**:
    - Measures the model's accuracy in classifying detected objects as wounds, stitches, or background.
    - **Trend**: A steady decline indicates the model is progressively better at distinguishing between classes.
    - **Observation**: Loss starts at around 2.5 and falls below 1.0, showing significant improvement.

3. **Training DFL Loss (Top Right)**:
    - Represents the distribution-focused localization loss, refining bounding box precision.
    - **Trend**: Consistent downward trend shows improved precision in bounding box boundaries.
    - **Observation**: Loss reduced from over 2.0 to about 1.4.
4. **Precision Metric (Top Row, Second from Right)**:
    - Precision measures the proportion of true positive detections among all positive predictions.
    - **Trend**: Rapid improvement in the initial epochs, stabilizing above 0.8.
    - **Observation**: A precision of ~0.85 indicates the model makes accurate positive predictions.
5. **Recall Metric (Top Right)**:
    - Recall measures the proportion of actual positives correctly identified by the model.
    - **Trend**: Initial fluctuations, followed by stabilization around 0.85.
    - **Observation**: High recall means the model successfully detects most true positives.
6. **Validation Box Loss (Bottom Left)**:
    - Similar to training box loss but evaluates performance on unseen data.
    - **Trend**: Decline from ~2.75 to ~1.5, indicating effective generalization.
    - **Observation**: Consistent with training loss, showing no signs of overfitting.
7. **Validation Classification Loss (Bottom Center)**:
    - Tracks classification performance on validation data.
    - **Trend**: Sharp decline, stabilizing below 1.0.
    - **Observation**: Confirms robust classification capabilities.
8. **Validation DFL Loss (Bottom Right)**:
    - Tracks bounding box refinement on validation data.

- **Trend**: Reduction from ~8.0 to below 2.0, indicating better box refinement.
- **Observation**: Drastic improvement in bounding box quality.

9. **mAP@0.5 Metric (Bottom Row, Second from Right)**:
    - Mean Average Precision at 50% IoU threshold, measuring detection accuracy.
    - **Trend**: Rapid increase to ~0.9, indicating high object detection accuracy.
    - **Observation**: A strong metric showing excellent detection capabilities.
10. **mAP@0.5-0.95 Metric (Bottom Right)**:
    - Mean Average Precision averaged over IoU thresholds from 50% to 95%.
    - **Trend**: Steady improvement, stabilizing near 0.6.
    - **Observation**: Indicates moderate performance across stricter IoU thresholds.

Model Evaluation

- **Accuracy**:
    - High precision (~0.85) and recall (~0.85) ensure reliable detection for critical classes.
- **Generalization**:
    - Validation losses closely align with training losses, indicating good generalization without overfitting.
- **mAP**:
    - mAP@0.5 of ~0.9 reflects strong detection accuracy, while mAP@0.5-0.95 of ~0.6 suggests room for improvement in stricter evaluations.

Real time detection code:

```python
from ultralytics import YOLO
import cvzone
import cv2

# Detection on images
model = YOLO('WoundModel.pt')

# Live webcam
cap = cv2.VideoCapture(0)

# Define the focal length (you need to calibrate your camera)
focal_length = 1000  # Replace with your calibrated focal length

while True:
    ret, image = cap.read()
    results = model(image)
    for info in results:
        parameters = info.boxes
        for box in parameters:
            # Move tensors to CPU
            x1, y1, x2, y2 = box.xyxy[0].cpu().numpy().astype('int')
            confidence = box.conf[0].cpu().numpy().astype('int') * 100
            class_detected_number = box.cls[0]
            class_detected_number = int(class_detected_number)
            class_detected_name = results[0].names[class_detected_number]

            # Calculate the width of the object in pixels
            width_pixels = x2 - x1

            # Assuming a known object size (e.g., 10 cm)
            object_width_cm = 10  # Replace with the actual object width

            # Calculate the distance using the formula: distance = (focal_length * object_width_cm) / width_pixels
            distance_cm = (focal_length * object_width_cm) / width_pixels

            # Display the bounding box and text information
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 0, 255), 3)
            # Calculate the position for the text
            text_position = [x1 + 8, y1 - 12]
            cvzone.putTextRect(image, f'{class_detected_name} {distance_cm:.2f} cm', text_position, thickness=2, scale=1.5)

    cv2.imshow('frame', image)
    cv2.waitKey(1)
```

Image Processing

Image processing is a field of computer science and engineering that focuses on manipulating, analysing, and enhancing digital images. It involves various techniques to process images to extract useful information, improve their quality, or prepare them for further analysis. Image processing is widely used in medical imaging, computer vision, robotics, remote sensing, and many other applications.

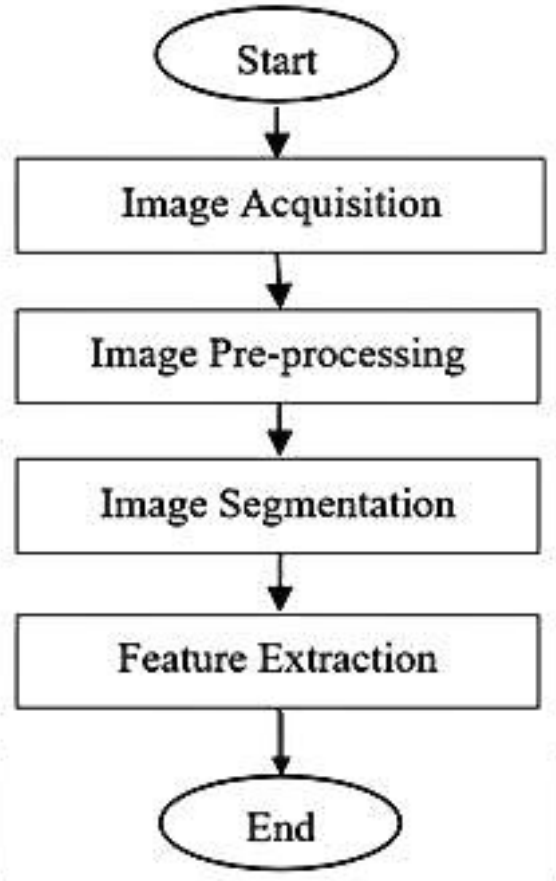
Importance:

- Enhances image quality for better visualization.
- Critical in fields like medical imaging, remote sensing and autonomous vehicles.
- Supports applications such as object detection and image segmentation.

Key Goals:

- Improve visual appearance of images captured by stereovision-camera.
- Analyze and interpret image content to gather and extract important features.
- Facilitate automation in medical applications like in doing surgeries.

Image processing Workflow:

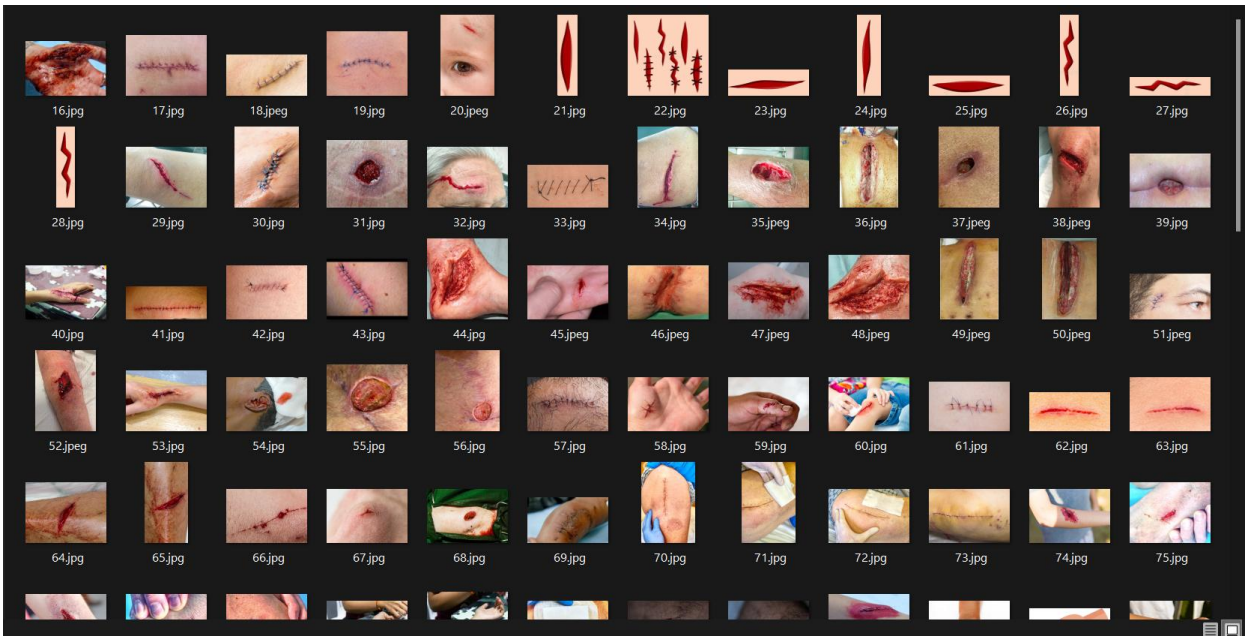


Step 1): Image Acquisition:

Image acquisition is the first step in the image processing workflow, as It involves capturing multiple images of both wounds and stitches using sensors or devices (ex: camera), so that there is a huge dataset collected to be manually annotated using LabelMe or automatically using SAM.

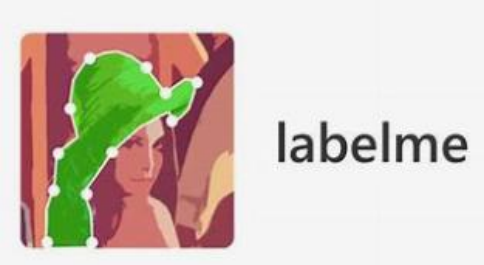
Methods of Acquisition:

- ✓ Acquiring images from trusted sources via internet.
- ✓ Phone Cameras.
- ✓ Digital Cameras



Around 5,000 images of wounds and stitched classes in total were collected from trusted sources in the internet and labelled using bounding boxes by the help of a GitHub annotating tool “LabelImg.exe”, then used image preprocessing python code so that in case of noisy images it can be fixed and clarified by using multiple modes and techniques to be explained later on. As for now there is a complete dataset of images and its corresponding labels for detection model training done previously.

Then polygon annotations of the same dataset used in object detection model, is done at first by using another annotating tool named “LabelMe.exe”, however it was very time consuming to do such thing for only 5 images, which made it impossible to repeat this for the rest of the dataset. As this software I used at first, where I have manually labelled the wound pixel by pixel to highlight the wound area and the same thing to the stitched class.

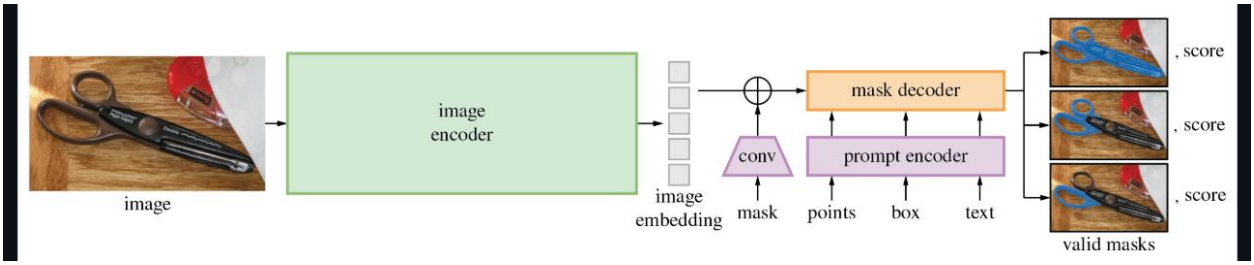


Here comes the advantage of the implementation of a code using SAM (Segment Anything Model) that is provided by Meta AI that takes these images, plus the weights of the detection model in .pt file and the weights of the large SAM model known as ‘sam_l.pt’, so that it outputs all of the images’ polygon annotations (wounds or stitched) in .txt in few minutes.

Introduction to SAM: The Segment Anything Model

The Segment Anything Model, or SAM, is a cutting-edge image segmentation model that allows for promptable segmentation, providing unparalleled versatility in image analysis tasks. SAM forms the heart of the Segment Anything initiative, a groundbreaking project that introduces a novel model, task, and dataset for image segmentation.

SAM's advanced design allows it to adapt to new image distributions and tasks without prior knowledge, a feature known as zero-shot transfer. Trained on the expansive SA-1B dataset, which contains more than 1 billion masks spread over 11 million carefully curated images, SAM has displayed impressive zero-shot performance, surpassing previous fully supervised results in many cases.



Key Features of the Segment Anything Model (SAM)

- **Promptable Segmentation Task:** SAM was designed with a promptable segmentation task in mind, allowing it to generate valid segmentation masks from any given prompt, such as spatial or text clues identifying an object.
- **Advanced Architecture:** The Segment Anything Model employs a powerful image encoder, a prompt encoder, and a lightweight mask decoder. This unique architecture enables flexible prompting, real-time mask computation, and ambiguity awareness in segmentation tasks.
- **The SA-1B Dataset:** Introduced by the Segment Anything project, the SA-1B dataset features over 1 billion masks on 11 million images. As the largest segmentation dataset to date, it provides SAM with a diverse and large-scale training data source.
- **Zero-Shot Performance:** SAM displays outstanding zero-shot performance across various segmentation tasks, making it a ready-to-use tool for diverse applications with minimal need for prompt engineering.

Available Models, Supported Tasks, and Operating Modes

This table presents the available models with their specific pre-trained weights, the tasks they support, and their compatibility with different operating modes like Inference, Validation, Training, and Export, indicated by ✅ emojis for supported modes and ❌ emojis for unsupported modes.

Model Type	Pre-trained Weights	Tasks Supported	Inference	Validation	Training	Export
SAM base	sam_b.pt	Instance Segmentation	✅	❌	❌	❌
SAM large	sam_l.pt	Instance Segmentation	✅	❌	❌	❌

As one of the best privileges using this technology is **Auto-Annotation**, which is s quick Path to Segmentation Datasets

Auto-annotation is a key feature of SAM, allowing users to generate a segmentation dataset using a pre-trained detection model. This feature enables rapid and accurate annotation of a large number of images, bypassing the need for time-consuming manual labeling.

Generate Your Segmentation Dataset Using a Detection Model

To auto-annotate your dataset with the Ultralytics framework, use the `auto_annotate` function as shown below:

Example

Python

```python
from ultralytics.data.annotator import auto_annotate

auto_annotate(data="path/to/images", det_model="yolo11x.pt", sam_model="sam_b.pt")
```

By the help of the above information taken from the ultralytics forum, concerning the usage of SAM in Segmentation of pretrained dataset that was detected using YOLO., our own custom dataset of wounds and stitched can be then annotated automatically in few minutes.

This is the code provided:

```python
# Ultralytics YOLO 🚀, AGPL-3.0 license

from pathlib import Path

from ultralytics import SAM, YOLO


def auto_annotate(data, det_model="WoundModel.pt", sam_model="sam_l.pt", device="", output_dir=None):
    """
    Automatically annotates images using a YOLO object detection model and a SAM segmentation model.

    This function processes images in a specified directory, detects objects using a YOLO model, and then generates
    segmentation masks using a SAM model. The resulting annotations are saved as text files.

    Args:
        data (str): Path to a folder containing images to be annotated.
        det_model (str): Path or name of the pre-trained YOLO detection model.
        sam_model (str): Path or name of the pre-trained SAM segmentation model.
        device (str): Device to run the models on (e.g., 'cpu', 'cuda', '0').
        output_dir (str | None): Directory to save the annotated results. If None, a default directory is created.

    Examples:
        >>> from ultralytics.data.annotator import auto_annotate
        >>> auto_annotate(data="ultralytics/assets", det_model="yolov8n.pt", sam_model="mobile_sam.pt")

    Notes:
        - The function creates a new directory for output if not specified.
        - Annotation results are saved as text files with the same names as the input images.
        - Each line in the output text file represents a detected object with its class ID and segmentation points.
    """
    det_model = YOLO(det_model)
    sam_model = SAM(sam_model)
```

```
data = Path(data)
if not output_dir:
    output_dir = data.parent / f"{data.stem}_auto_annotate_labels"
Path(output_dir).mkdir(exist_ok=True, parents=True)

det_results = det_model(data, stream=True, device=device)

for result in det_results:
    class_ids = result.boxes.cls.int().tolist()  # noqa
    if len(class_ids):
        boxes = result.boxes.xyxy  # Boxes object for bbox outputs
        sam_results = sam_model(result.orig_img, bboxes=boxes, verbose=False, save=False, device=device)
        segments = sam_results[0].masks.xyn  # noqa

        with open(f"{Path(output_dir) / Path(result.path).stem}.txt", "w") as f:
            for i in range(len(segments)):
                s = segments[i]
                if len(s) == 0:
                    continue
                segment = map(str, segments[i].reshape(-1).tolist())
                f.write(f"{class_ids[i]} " + " ".join(segment) + "\n")
```

```
from ultralytics.data.annotator import auto_annotate

auto_annotate(data="images", det_model="Last_Wound_train.pt", sam_model='sam_l.pt')
```

```
image 1/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1.jpg: 640x640 2 stitcheds, 10 wounds, 18.2ms
image 2/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/10.jpeg: 416x640 1 wound, 63.4ms
image 3/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/100.jpg: 448x640 1 wound, 60.4ms
image 4/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1000.jpg: 640x640 2 wounds, 18.9ms
image 5/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1001.jpg: 640x640 1 wound, 17.8ms
image 6/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1002.jpg: 640x640 1 wound, 17.1ms
image 7/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1003.jpg: 640x640 1 wound, 17.7ms
image 8/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1004.jpg: 640x640 1 wound, 18.0ms
image 9/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1005.jpg: 640x640 1 wound, 17.2ms
image 10/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1006.jpg: 640x640 1 wound, 17.3ms
image 11/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1007.jpg: 640x640 1 wound, 17.5ms
image 12/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1008.jpg: 640x640 1 wound, 17.2ms
image 13/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1009.jpg: 640x640 1 wound, 17.9ms
image 14/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/101.jpg: 448x640 1 wound, 13.9ms
image 15/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1010.jpg: 640x640 2 wounds, 17.6ms
image 16/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1011.jpg: 640x640 1 wound, 18.5ms
image 17/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1012.jpg: 640x640 1 wound, 17.0ms
image 18/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1013.jpg: 640x640 3 wounds, 18.2ms
image 19/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1014.jpg: 640x640 1 wound, 19.2ms
image 20/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1015.jpg: 640x640 1 wound, 17.1ms
image 21/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1016.jpg: 640x640 1 wound, 17.6ms
image 22/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1017.jpg: 640x640 1 wound, 18.2ms
image 23/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1018.jpg: 640x640 1 wound, 18.2ms
image 24/3154 /home/kholoud-waleed/Grad_Project/4-Work/SAM_annotations_(detection model)/images/1019.jpg: 640x640 1 wound, 17.0ms
```

Step 2): Image Pre-processing:

Image pre-processing is the second step for preparing all of the acquired images for further analysis by improving image clarity, sharpness, smoothness and filtering out noise from them, to make wound features more distinguishable.

Methods of Pre-processing:

- ✓ Resizing: using a standard size for consistency.
- ✓ Filtering: using Gaussian filtering and Unsharp masking for noise reduction.
- ✓ Contrast Enhancement: using CLAHE.
- ✓ Augmentation: creating variations of the original image (like rotations, flips, or colour adjustments) to increase the dataset size for training ML models.

Overview of the python code implemented using the above preprocessing techniques:

```
def preprocess_image(img, target_size):
    """
    Parameters:
    - image: Input image as a NumPy array.
    - target_size: Tuple specifying the target size for resizing (width, height).
    Returns:
    - Preprocessed image as a NumPy array.
    """
    # (1) Resize the image
    #Get original dimensions
    h, w = img.shape[:2]
    h_n, w_n = target_size
    #Calculate the scaling factor
    scale = min(w_n / w, h_n / h)
    #Calculate new dimensions
    new_w = int(w * scale)
    new_h = int(h * scale)
    #Resize
    img = cv2.resize(img, (new_w, new_h), interpolation=cv2.INTER_LINEAR)
    #Get new dimensions
    h, w = img.shape[:2]
    h_n, w_n = target_size
    #Calculate padding
    delta_w = w_n - w
    delta_h = h_n - h
    top, bottom = delta_h // 2, delta_h - (delta_h // 2)
    left, right = delta_w // 2, delta_w - (delta_w // 2)
    #Black Padding the image
    img = cv2.copyMakeBorder(img, top, bottom, left, right, cv2.BORDER_CONSTANT, value=(0, 0, 0))
```

In the above code snippet, resizing while maintaining aspect ratio is used by using a scaling factor that maintains the aspect ratio and if the resized image doesn't match the target size exactly, it is padded with black pixels. Like in the following image shown

Why Resizing?

- Ensures that all images are of a uniform size, which is required for most machine learning and deep learning models like in YOLOv8.
- Maintains the original aspect ratio to prevent distortion.
- Ensures consistency in image dimensions for neural network training.
- Padding helps preserve information by preventing cropping.

Also, black padding is used to ensure that all images have the exact target dimensions after resizing while preserving their aspect ratio. This is done by calculating the difference between the target size and the resized image, then black pixels (0,0,0) are added equally to the top, bottom, left, and right sides to fill the missing space.

Why Black Padding?

- Keeps the image cantered.
- Maintains consistency across all images.
- Prevents important details from being cropped.

```
# (2) Gaussian filter for noise filter and Sharpness enhancement
img = cv2.GaussianBlur(img, (3, 3), sigmaX=0)  #Gaussian filter
blurred = cv2.GaussianBlur(img, (0, 0), sigmaX=2)
img = cv2.addWeighted(img, 1.7, blurred, -0.7, 0) #Unsharp masking

# (3) Apply Contrast Limited Adaptive Histogram Equalization (CLAHE) to improve contrast
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
clahe = cv2.createCLAHE(clipLimit=1.25, tileGridSize=(4, 4))
img_yuv[:, :, 0] = clahe.apply(img_yuv[:, :, 0])
img = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)
```

Scrolling down into the same code, where this snippet shows that Gaussian blurring for noise reduction is used with a kernel of 3x3 kernel to smooth the image by averaging pixel intensities.

The Gaussian blur filter reduces detail and noise in an image. It "blurs" the image by applying a Gaussian function to each pixel and its surrounding pixels. This can help smooth edges and details in preparation for edge detection or other processing techniques.

Why Gaussian Blur?

- Reduces random noise, making the image cleaner.
- Helps improve feature extraction for edge-detection algorithms and neural networks.
- Reduces overfitting by eliminating unnecessary fine details

before gaussian blur

after gaussian blur

Also, unsharp masking is used for sharpness enhancement, where it enhances important details in the image by making edges sharper and that is what we aim to do to clearly detect the edges of wound to be stitched then.

How is this done? Well, a second, stronger Gaussian blur (sigmaX=2) is applied, thus the blurred image is subtracted from the original using a weighted sum: sharpened=1.7×original−0.7×blurred, where this technique enhances edges and details.

Why Unsharp Masking?

- Enhances important features, improving visibility of details.
- Helps with feature extraction for deep learning models.
- Makes edges more distinct, which can improve object detection.

After filtering out the noise from the images then another new technique is used for contrast enhancement, which is Contrast Limited Adaptive Histogram Equalization (CLAHE)

How it works?

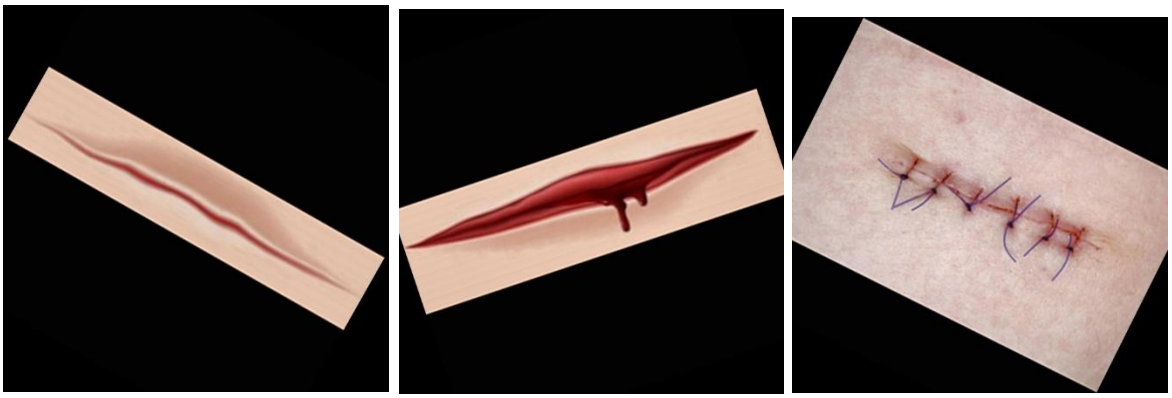
- Converts the image to **YUV colour space** (where Y represents brightness).
- Applies CLAHE only to the **Y-channel** (luminance) to improve contrast locally.
- Converts the image back to **BGR colour space**.

Why CLAHE?

- Improves contrast in images, making details more distinguishable.
- Unlike global histogram equalization, it prevents over-enhancing certain areas.
- Enhances contrast while preserving local details.
- Works well for images with poor lighting conditions.

After finishing the image preprocessing techniques to fix the problems in images and clarify its details, Image augmentation is used, in order to increase our dataset and diverse it by using many variations by flipping vertically or horizontally and rotating by random angles for better detection and segmentation to be done with higher confidence score. As shown in the code below.

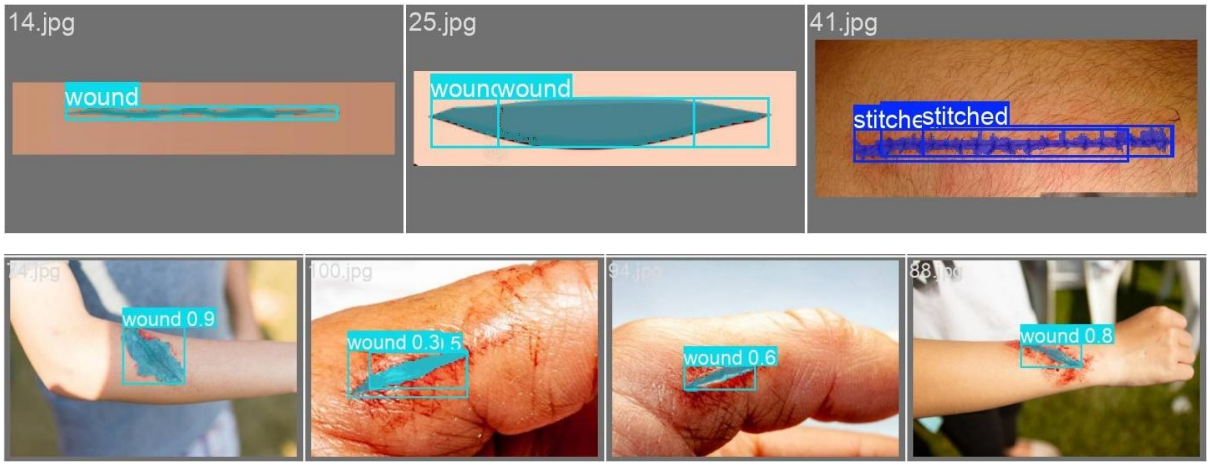
```
def augment_images_in_folder(input_folder, output_folder):
    """
    Parameters:
    - input_folder: Path to the folder containing preprocessed images.
    - output_folder: Path to the folder where augmented images will be saved.
    Returns:
    - Apply random augmentations to the image.
    """
    # (1) Create an o/p folder if it doesn't exist to save the preprocessed images
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    # (2) Iterate through all images in the i/p folder to preprocess them
    for filename in os.listdir(input_folder):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            # Specify then Load the image path
            image_path = os.path.join(input_folder, filename)
            image = cv2.imread(image_path)
            # Horizontal or Vertical flips
            flip_type = np.random.choice([1, 0, None])  # 1 for horizontal, 0 for vertical
            image = cv2.flip(image, flip_type)
            # Random rotation
            angle = np.random.uniform(90, 90)  # rotate bet the range of -90 and 90 deg
            (h, w) = image.shape[:2]
            center = (w // 2, h // 2)
            matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
            image = cv2.warpAffine(image, matrix, (w, h))
            # Save the preprocessed image
            output_path = os.path.join(output_folder, filename)
            # Convert to 0-255 scale for saving
            cv2.imwrite(output_path, image)
```



Step 3): Image Segmentation:

Image segmentation is dividing an image into specific regions “segments”, by identifying and isolating the wound region from the background for better analysis and interpretation.

This is done by using SAM to generate the polygon annotations of wounds, then YOLOv8 to provide segmentation image then binary masking.

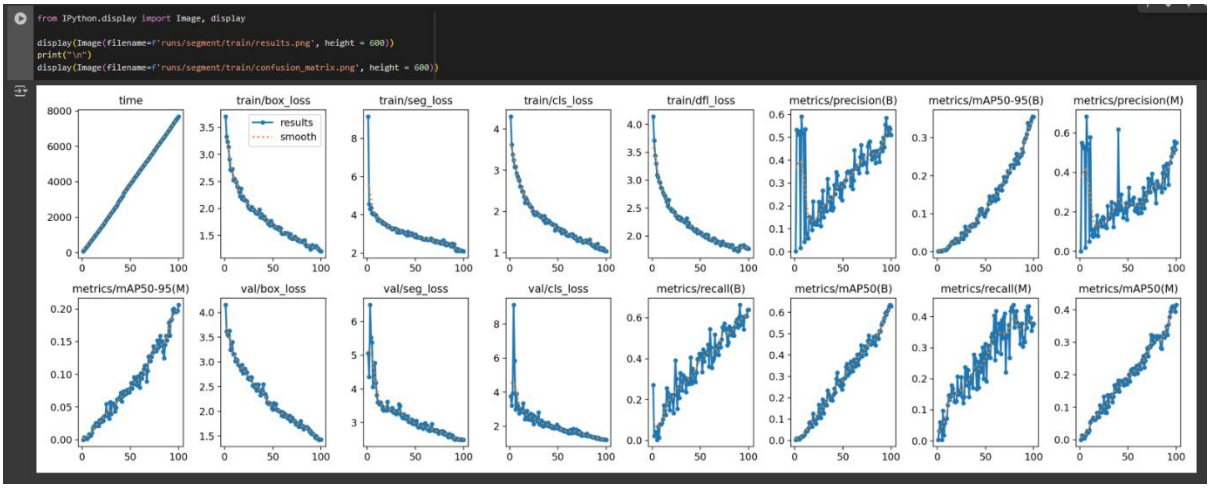


After having both the images and polygon labels ready, a YOLOv8 Image segmentation code is implemented for this custom dataset to be used as an input for training the model by using 100 epochs, 1 workers, 16 batch size.

After hours of training the model using Google Colab, the weights of segmentation model is then provided in .pt that I then tested it to do segmentation for each wound and stitched classes, so that feature extraction can be done like calculating the wound’s key-points, area, depth and other necessary information to be used in the robot planning phase.

```
from ultralytics import YOLO

# Load a model
model = YOLO("/content/drive/MyDrive/Grad_proj/proj_seg_final/yolov8x-seg_custom.yaml")  #build a new model from scratch
#model = YOLO("yolov8n-seg.pt")  #load a pretrained model (recommended for trial training)

# Use the model
results = model.train(data="/content/drive/MyDrive/Grad_proj/proj_seg_final/custom_data.yaml", epochs=100, workers=1, batch=16,imgsz=640)
```

Unfortunately, this was the first training done for the first dataset collected of around 1,000 images that is why the results are not the best, Since the dataset was then increased in size to about 5,000 images but the detection was not yet prepared to do the segmentation part, as the two stages are related to each other.



This figure visualizes the model's performance metrics as subplots that monitor the training process and evaluate its performance during training and validation.

First Row (Training Metrics)

1. **Time:**
    - The x-axis represents the number of epochs (from 0 to 100).
    - The y-axis represents the cumulative time spent in training.
    - The linear increase suggests training is progressing smoothly without significant slowdowns.

2. **train/box_loss:**
    - The x-axis is the number of epochs.
    - The y-axis is the bounding box regression loss.
    - This loss decreases over epochs, indicating the model is improving at predicting bounding box locations.

3. **train/seg_loss:**
    - Represents segmentation loss over epochs.
    - The decreasing trend suggests the model is getting better at segmenting objects.

4. **train/cls_loss:**
    - Classification loss, measuring how well the model assigns the correct class to objects.
    - It decreases over epochs, showing improvement in classification.

5. **train/dfl_loss:**
    - Distribution Focal Loss (DFL), used for refining box predictions.
    - A decreasing trend indicates better localization precision.

6. **metrics/precision(B):**
    - Precision of bounding box predictions for class "B."
    - A rising trend means the model is making fewer false positives.

7. **metrics/mAP50-95(B):**
    - Mean Average Precision (mAP) at different IoU thresholds for class "B."
    - The increase shows improved overall detection performance.

8. **metrics/precision(M):**
    - Precision of bounding box predictions for class "M."
    - Similar to (6), indicating model improvement.

Second Row (Validation Metrics)

9. **metrics/mAP50-95(M):**
    - mAP for class "M" at various IoU thresholds.
    - Higher values indicate better model performance.

10. **val/box_loss:**
    - Validation bounding box loss.
    - It follows the same decreasing trend as training, meaning the model generalizes well.

11. **val/seg_loss:**
    - Validation segmentation loss.
    - Decreasing trend confirms improved segmentation.

12. **val/cls_loss:**
    - Validation classification loss.
    - It should decrease similarly to training classification loss.

13. **metrics/recall(B):**
    - Recall for class "B," showing how many true positives are detected.
    - The increase suggests better recall performance.

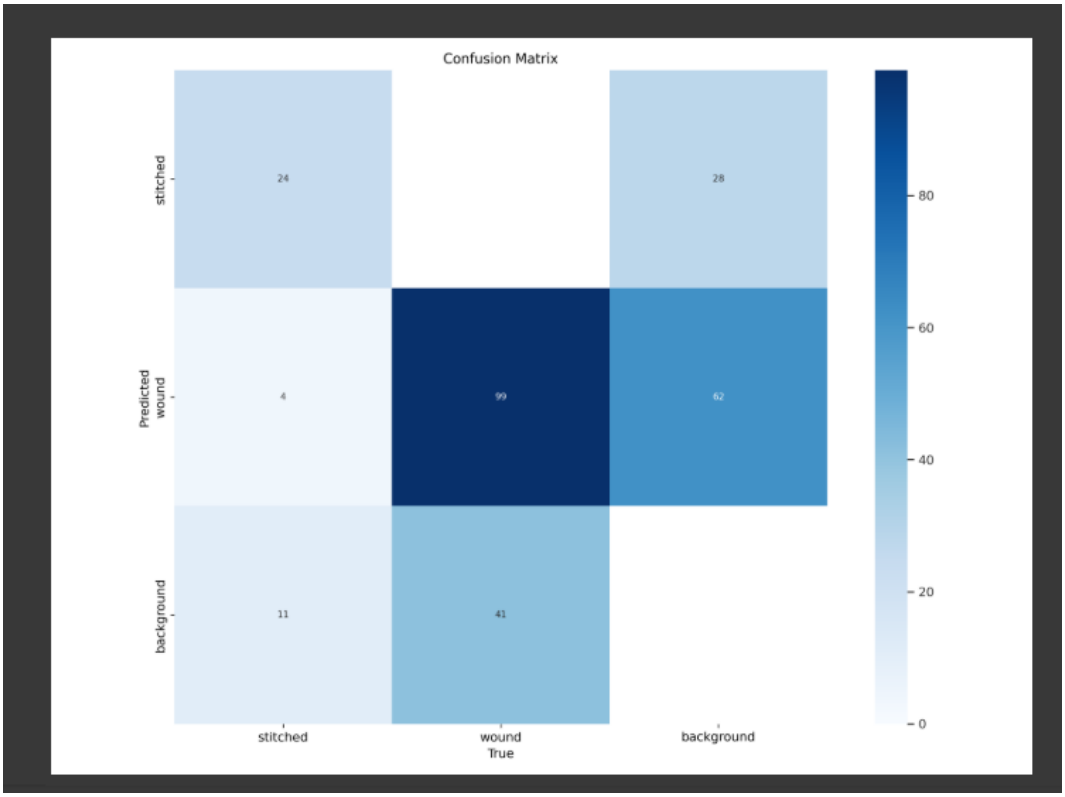
14. **metrics/mAP50(B):**
    - Mean Average Precision at IoU 0.50 for class "B."
    - A rising curve means detection accuracy is improving.

15. **metrics/recall(M):**
    - Recall for class "M."
    - Increasing recall indicates fewer false negatives.

16. **metrics/mAP50(M):**
    - Mean Average Precision at IoU 0.50 for class "M."
    - A steady increase suggests better model performance.

Insights & Observations

- Losses (box, segmentation, classification, and DFL) are all decreasing, indicating successful training.
- Precision and recall are increasing, showing fewer false positives and negatives over time.
- Validation losses follow a similar trend to training losses, meaning the model generalizes well without severe overfitting.
- mAP50 and mAP50-95 are steadily improving, confirming the model's ability to detect and classify objects accurately.



This figure visualizes the model's confusion matrix as its structure of rows representing the predicted classes (stitched, wound, background). and columns representing the actual/ true classes.

Key Observations:

- **Stitched Class**:
  - 24 correct predictions, 4 misclassified as wounds, and 11 as background.
  - **Accuracy**: High, with minor confusion with background.

- **Wound Class**:
  - 99 correct predictions, 0 misclassified as stitched, and 42 as background.
  - **Accuracy**: Excellent, though some overlap with background.

- **Background Class**:
  - 0 correct predictions, 28 misclassified as stitched, and 62 misclassified as wounds.
  - **Accuracy**: Lower, likely due to imbalance or subtle features.

Precision and Recall for Each Class:

- Precision and recall for wounds are high, indicating reliable detection for it, but poorly misclassifies stitched.
- Background class must be improved, suggesting potential enhancements like more diverse training samples.

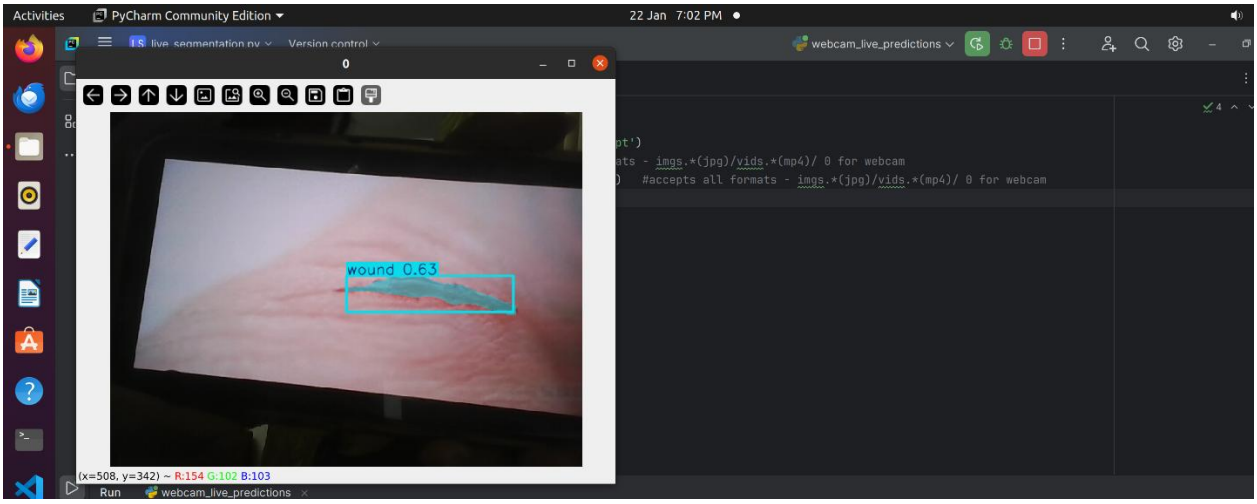
```
import glob
from IPython.display import Image, display

# Display results of predicted images
for img_path in glob.glob(f'runs/segment/predict8/*.jpg')[:9]:
    print("\n")
    display(Image(filename=img_path, height=600))
```

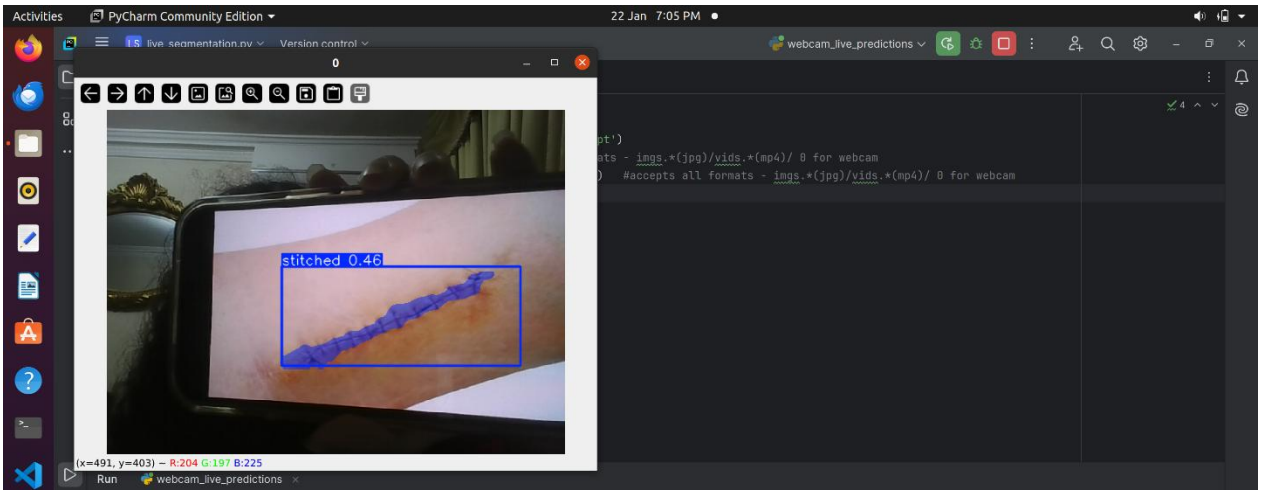
Predictions made upon the model on some test data like for example the following image which is segmented and ready to be binary masked then:

```python
from ultralytics import YOLO

model = YOLO('runs/segment/train/weights/best.pt')
#model.predict(source="0")   #accepts all formats - imgs.*(jpg)/vids.*(mp4)/ 0 for webcam
model.predict(source="0", show=True, conf=0.45)   #accepts all formats - imgs.*(jpg)/vids.*(mp4)/ 0 for webcam
```

This code shown above is used to set up real-time segmentation of wounds and stitched wound using laptop's webcam

After running the code for testing it practically, the webcam's window opened and directed sample images of wounds and stitched wounds from mobile phone and it successfully predicted them correctly.

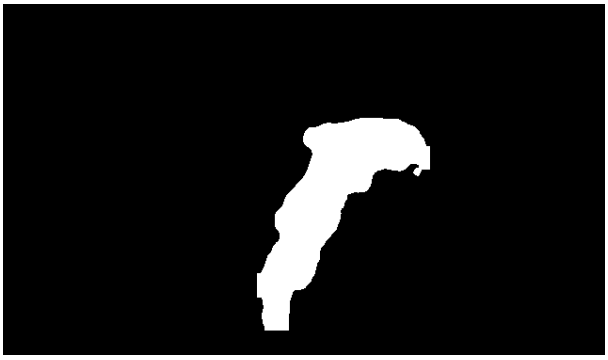


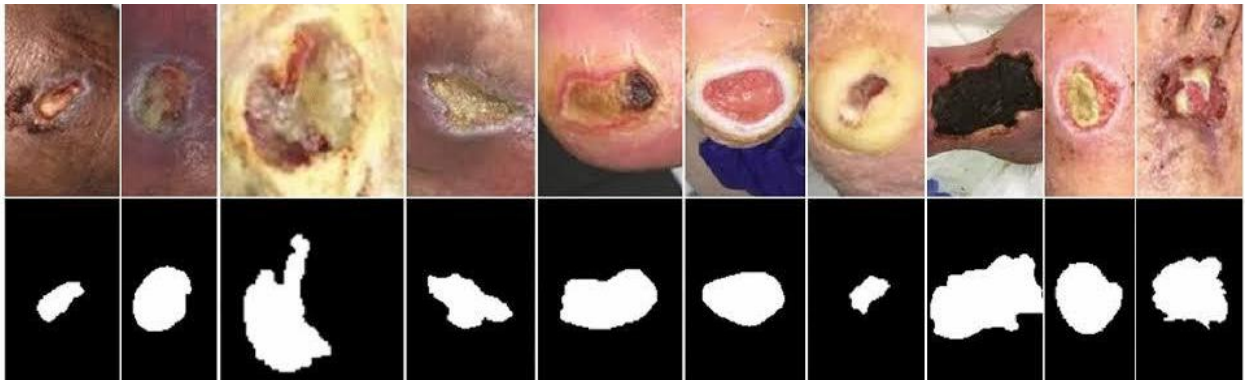


```python
# Create output folder
os.makedirs("masks1_output", exist_ok=True)

# Iterate over all images in results
for i, result in enumerate(results):
    if result.masks is not None:  # Check if masks exist
        mask_tensor = result.masks.data  # Get the mask tensor

        for j in range(mask_tensor.shape[0]):  # Iterate over detected masks
            mask_array = mask_tensor[j].cpu().numpy()  # Convert to NumPy
            mask = (mask_array * 255).astype('uint8')  # Convert to 8-bit format
            mask = mask.squeeze()  # Ensure correct shape

            # Save the mask
            filename = os.path.join("masks1_output", f"image_{i}_mask_{j}.png")
            cv2.imwrite(filename, mask)  # Save using OpenCV
            print(f"Saved: {filename}")
    else:
        print(f"No masks found for image {i}")
```

```
Saved: masks1_output/image_0_mask_0.png
```

After running the above code, the segmented image using YOLOv8 model id then binary masked and ready to be used for feature extraction and path planning phases.





Step 4): Feature Extraction (Future task to be completed):

Feature extraction is identifying and quantifying important characteristics from segmented images of wounds, which are essential for analysis, obtaining relevant information and facilitating interpretation.

Techniques:

- ✓ Shape Features: analyses contours and geometric properties such as area, perimeter, and aspect ratio.
- ✓ Colour Features: extracts colour information using colour histograms.
- ✓ Texture Features: assess the texture of surfaces using (LBP) or (GLCM) to capture patterns, smoothness and regularity of surfaces.

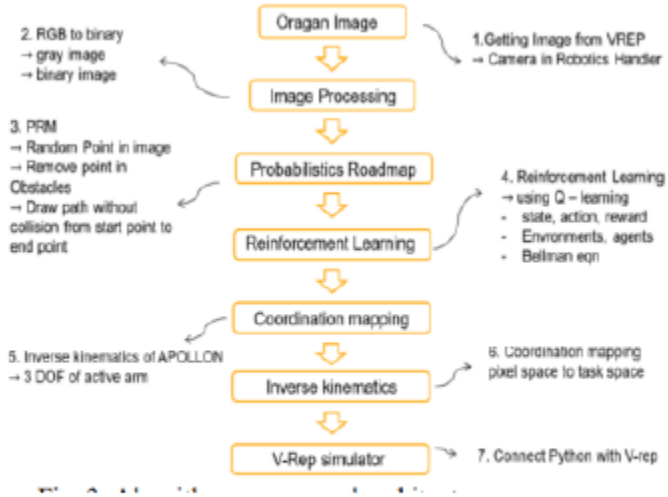
(a) Wound capturing (b) Wound segmentation (c) Wound reconstruction and area measurement

6.4. Path planning:

This subsection focuses on the various techniques and algorithms used to plan paths for robotic surgery, particularly for tasks like suturing. Path planning is essential for ensuring that the surgical robot avoids collisions with surrounding tissues, follows the optimal path, and operates in a dynamic environment.

The following sections detail the core components involved in path planning, including image processing, Probabilistic Roadmap, Reinforcement Learning, coordinate mapping, and the use of V-REP simulator.

Introduction



(a) Algorithm process and architecture

Path planning is a critical component of robotic surgery, as it ensures that the surgical instruments move safely and efficiently within the body. This involves considering factors such as the location of surrounding tissues, gripper forces, and the dynamic nature of the surgical environment. Effective path planning allows for more precise and faster surgeries, reducing the physical burden on surgeons while increasing the safety and reliability of the procedure.

Laparoscopic robotic surgery, particularly with systems like Intuitive Surgical's da Vinci robotic surgery, has gained significant popularity in recent years due to its ability to minimize incisions, reduce wound areas, and promote faster postoperative recovery. The da Vinci system has proven its safety and convenience, with its precise robotic movements ensuring more accurate surgery. Unlike conventional surgical methods, where the surgeon performs the surgery directly, the da Vinci robot's precise motion prevents hand tremors and allows for greater flexibility within the surgical area through a master-slave control system operated by the surgeon.

Despite the advantages of robotic surgery, physicians still face the challenge of performing repetitive tasks such as suturing, which can cause fatigue and stress during long procedures. As the demand for more efficient and less tiring surgical techniques grows, there has been increasing research into automation in robotic surgery, especially for tasks like automatic suturing, needle path planning, and tissue resection. The da Vinci Surgical System has already paved the way for automation in surgery, but the incorporation of fully automated suturing remains an area of active development.

A key challenge in automatic suturing is the complexity of surgical tissue manipulation, which involves precise control of gripper forces, suturing direction, and path planning, all while preventing damage to surrounding tissues. Optimal path planning is essential, particularly in avoiding collisions between the surgical tool and surrounding tissues. The dynamic nature of the surgical environment adds another layer of complexity, requiring realtime adjustments to the robot's actions. Path planning techniques, such as Probabilistic Roadmap (PRM), offer an intuitive and computationally efficient approach, allowing for quick identification of paths from a starting point to an endpoint. While this method is useful for real-time collision avoidance, it does not always provide the optimal path and can struggle with dynamic environments.

To address these challenges, Reinforcement Learning (RL) has emerged as a promising solution. RL has proven effective in dealing with uncertainties, as seen in its applications to autonomous vehicles, robotics, and even games like Google's AlphaGo. In the context of robotic surgery, RL can help the da Vinci system adapt to dynamic conditions in the surgical field, providing a more precise and efficient approach to automated suturing and tissue manipulation. Research is ongoing, with many focusing on optimizing path planning for suturing tasks to further reduce

Probabilistic Roadmap

Probabilistic Roadmap (PRM) is a widely used path planning method in robotics, especially for complex environments like the human body during surgery. PRM works by constructing a roadmap of possible paths and selecting the optimal path from a start to an endpoint, avoiding obstacles in the process. This method is computationally efficient, making it suitable for real-time applications in robotic surgery. While PRM is effective for static environments, its ability to adapt to dynamic changes during surgery is a key area of ongoing research.

In a Probabilistic Roadmap (PRM), the objective is to create a roadmap for navigating through a robotic environment. The roadmap is a graph where the nodes represent valid configurations (or milestones), and the edges represent feasible paths between these configurations. The method works especially well for high-dimensional spaces, such as robot arm movements or complex environments like surgical robots, where the motion planning problem is hard to solve directly.

Roadmap Construction and Vision Processing A roadmap is constructed by selecting a set of valid points (milestones) in the environment that are free of obstacles. These points are considered non-collision nodes. In the construction phase of PRM, Vision Processing plays an important role by identifying random configurations in the environment. Here’s how it works in detail:

Vision Processing for Milestone Selection

A predefined number of points are randomly sampled within the given environment. In a robotic system, this could be within a defined region, where the robot may move. These points are then checked against a binary image of the environment to ensure that they are not colliding with any obstacles (i.e., the points are not inside or near any obstacles). Only non-collision points (those inside free space) are selected as milestones for the roadmap.

Connection of Milestones:

After the random sampling of milestones, the system attempts to connect these milestones. This is typically done by connecting each node to k nearest neighbors, where k is a predefined number. Alternatively, nodes can be connected to all other nodes within a certain distance, creating a dense network of connections. These connections are valid as long as

there is no collision along the edge between the nodes. The system will continue this process of creating connections until the roadmap is sufficiently dense. A dense roadmap means there are many possible paths to travel from one point to another, which helps ensure that paths can be found for a variety of starting and target points. Two Phases of PRM The PRM process is divided into two main phases:

1. Construction Phase:

Goal: To build a roadmap that describes all possible configurations in the free space of the environment. Steps: Random Sampling: The system randomly generates a predefined number of points in the free space and ensures they are non-colliding with obstacles. Connecting Nodes: The milestones (non-collision nodes) are connected to their k nearest neighbors. Alternatively, if the algorithm uses distance-based connection criteria, nodes within a certain radius are connected. Graph Building: Once the connections are made, the result is a graph (roadmap) where the node are milestones, and the edges represent possible paths between them. The graph is continually expanded until the roadmap is sufficiently populated with non-collision paths.

2. Query Phase:

Goal: To find a path from a given start point to a target point using the constructed roadmap. Steps: Adding Start and Target Nodes: The system adds the start point and target point to the roadmap as new nodes, ensuring that both are within free space and do not collide with obstacles. Connecting to the Graph: The new nodes (start and target) are then connected to the existing roadmap. The connections are made based on proximity to the existing nodes in the roadmap. For example, the start node may connect to the nearest few nodes in the roadmap, and similarly for the target. Path Query: Once the start and target nodes are connected to the roadmap, the system uses a graph search algorithm (such as Dijkstra's algorithm) to find the shortest path between the start and target nodes. This search considers the edges in the graph and determines the optimal route by evaluating the distances or costs associated with each path.

Dijkstra's algorithm

It is a well-known graph search algorithm used to find the shortest path between nodes in a graph, particularly when all edges have non-negative weights. It works by incrementally finding the shortest path from the starting node to all other nodes in the graph. Here's a step-by-step explanation of how it works:

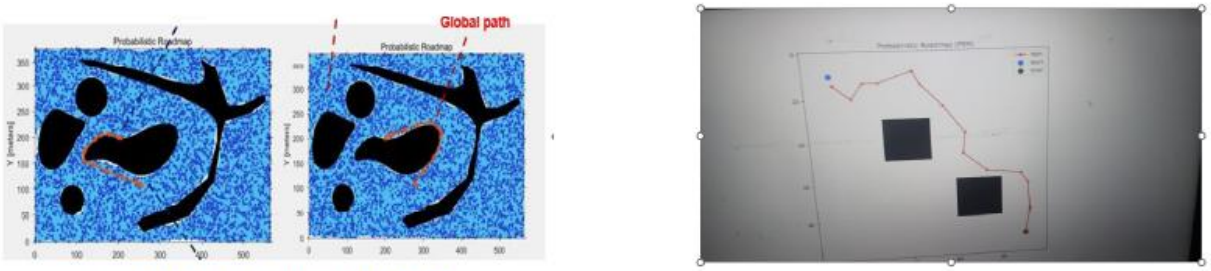
How Dijkstra's Algorithm Works Initialization:

Mark the distance to the start node as 0 (since the shortest path from the start to itself is 0). Mark the distance to all other nodes as infinity (since no path has been found yet). Set the previous node for each node to null (to track the path). Add all nodes to a priority queue (or min-heap) with their distances as priorities. The priority queue allows us to always expand the node with the smallest tentative distance.

Relaxation: While the priority queue is not empty: Remove the node with the smallest distance from the priority queue. Let's call this node current node. For each neighbor of the current node: Calculate the tentative distance to the neighbor as the sum of the current node's distance and the weight of the edge connecting them. If the tentative distance is smaller than the previously recorded distance for the neighbor, update the distance to the neighbor. Update the previous node for the neighbor to be the current node (this helps to reconstruct the shortest path later). If the distance was updated, add the neighbor back to the priority queue to revisit it.

Termination: The algorithm terminates when the priority queue is empty or when all nodes have been processed. At this point, the shortest distances from the starting node to all other nodes are found.

Path Reconstruction: After the algorithm finishes, you can reconstruct the shortest path from the start node to any other node by following the previous node pointers from the destination node back to the start node



(a) PRM ALGORITHM

(b) PRM IN PRACTICAL

Figure 3: PRM

Reinforcement Learning

Reinforcement Learning (RL) has been gaining traction in robotic surgery due to its ability to handle uncertainties and dynamic environments. In RL, the robot learns optimal actions through trial and error, receiving feedback from its environment to refine its strategy over time. For surgical tasks like suturing, RL can enable the robot to adapt to changes in tissue, obstacles, and surgical tool movements, ultimately leading to more efficient and precise actions in real-time.

Reinforcement learning (RL) is a type of machine learning where an agent learns how to behave in an environment by performing actions and receiving feedback in the form of rewards or penalties. Unlike supervised or unsupervised learning, where the learning is based on pre-labeled data or hidden patterns in the data, RL is based on learning from interactions with the environment, guided by rewards or compensations.

Q-Learning Explanation

In RL, Q-learning is one of the most popular methods for solving the decision-making problem. Its to find the optimal actions election policy, which tells the agent which action to take in each state, in order to maximize cumulative future rewards.

- State (St): Represents the current condition / situation of the environment at time t.
- Action (At): The action taken by the agent at time t.
- Reward (Rt+1): The immediate feedback the agent receives after taking action At in state St, which helps the agent evaluate its performance.
- Q-function: The Q-value Q(St,At) is a measure of the quality of the action At taken in state St. It indicates the expected cumulative future reward that can be obtained from state St by taking action At and following the optimal policy thereafter. The agent aims to learn this Q-value function, so it can always choose the best action.

Update Formula (Q-Learning)

The key idea in Q-learning is to update the Q-values using the formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$

Where:

- $Q(S_t, A_t)$: Current Q-value for the state S_t and action A_t.
- R_{t+1}: Reward the agent receives after taking action A_t in state S_t and transitioning to state S_{t+1}.
- γ: Discount factor ($0 \leq \gamma \leq 1$). It represents how much future rewards are valued. A value of 0 makes the agent short-sighted, focusing only on immediate rewards, while a value of 1 means future rewards are as important as immediate rewards.
- $\max_{a'} Q(S_{t+1}, a')$: The maximum Q-value for the next state S_{t+1} over all possible actions a'. This represents the best possible action that could be taken from state S_{t+1} (i.e., the agent's expectation of the highest reward achievable from the next state).
- α: Learning rate ($0 \leq \alpha \leq 1$). It determines how much new information (new rewards) will overwrite the old Q-value.

How It Works:

1. At each time step, the agent takes an action in a given state and receives a reward based on the action taken.
2. The Q-value for the state-action pair (S_t, A_t) is updated based on the reward received and the expected future rewards (via the next state S_{t+1}).
3. The agent updates its Q-values iteratively, gradually learning which actions lead to higher rewards.
4. The update process continues until the Q-values converge, meaning the agent has learned the optimal policy for choosing actions.

Q-Learning and PRM Integration

In traditional PRM, the roadmap is constructed offline, and a graph search algorithm is used to find paths. However, Reinforcement Learning (RL) (e.g., Q-learning) can complement PRM by enabling the agent to adapt to dynamic environments or learn optimal paths over time.

How Q-learning Works in PRM:

State Representation: The states in Q-learning could be the robot's configurations (positions) in the environment. These states are the nodes in the PRM roadmap.

Action Representation: Actions in Q-learning could correspond to moving from one configuration (node) to another by following edges in the PRM roadmap. The robot can move from one node to another, and the action taken can be represented by the edges in the roadmap.

Reward Function: The reward function in Q-learning helps the robot evaluate how good a move (action) is. In the case of PRM, the reward could be based on:

- Distance to the goal: The robot receives a higher reward if it moves closer to the goal.
- Collision avoidance: The robot could be penalized if it takes an action that leads to a collision with an obstacle.
- Efficiency of the path: The robot might receive higher rewards for taking shorter or faster paths.

Updating Q-values: The robot will explore different paths in the PRM graph and update the Q-values for state-action pairs based on the received rewards. The Q-value of a state-action pair Q(St,At) is updated using the formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$

Where:

- St and At represent the current state and action.
- Rt+1 is the reward for the action taken.
- St+1 is the new state after the action.
- γ is the discount factor, controlling how much future rewards are considered.

Learning Optimal Paths:

Through exploration and exploitation (balancing between exploring new paths and exploiting known good paths), the robot learns the optimal policy to navigate through the environment. Over time, it updates the Q-values and discovers which paths in the PRM graph lead to the highest cumulative rewards.

Advantages of Integrating PRM and Q-learning:

- Adaptation to Dynamic Environments: Q-learning allows the agent to adapt its strategy to dynamic environments where obstacles might move or change. The robot can update its path based on the environment's feedback.
- Improved Pathfinding: While PRM generates a roadmap, Q-learning can help the robot learn to choose the most efficient or safest path, even in complex environments with moving obstacles or other unpredictable factors.
- Learning Over Time: As the robot navigates the environment, it refines its decision-making process based on accumulated experience, potentially finding better paths over time than those initially generated by PRM alone.

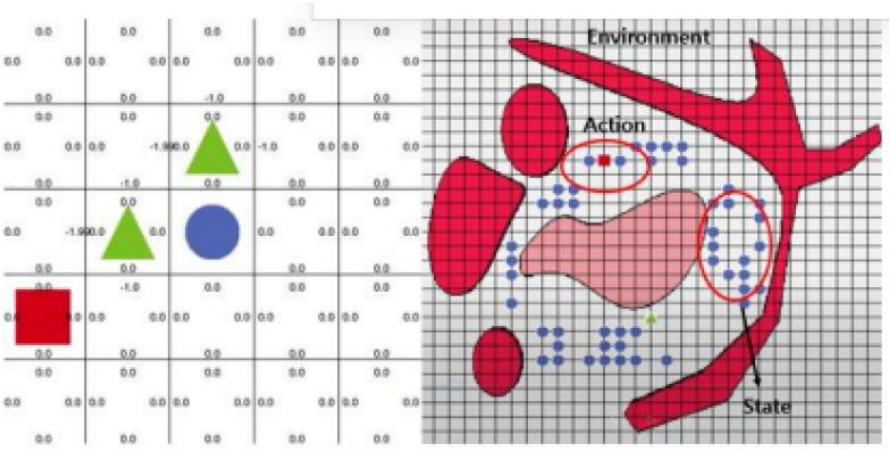


Figure 4: RL

Coordinate Mapping

Coordinate mapping is an essential part of path planning, as it translate real-world positions into a format that the robotic system can understand and navigate. By converting the surgical site's coordinates into a system that aligns with the robot's motion, the planning system can precisely control the movement of the tools within the body. This process ensures that the robot's actions are in sync with the surgeon's intentions, allowing for safe and accurate surgery. the path learned by the RL agent consists of a sequence of states or positions.

By converting each state to its corresponding (x, y, z) coordinates, we create a spatial trajectory, which is a sequence of points in real-world coordinates. he path learned by the RL agent consists of a sequence of states or positions. By converting each state to its corresponding (x, y, z) coordinates, we create a spatial trajectory, which is a sequence of points in real-world coordinates.

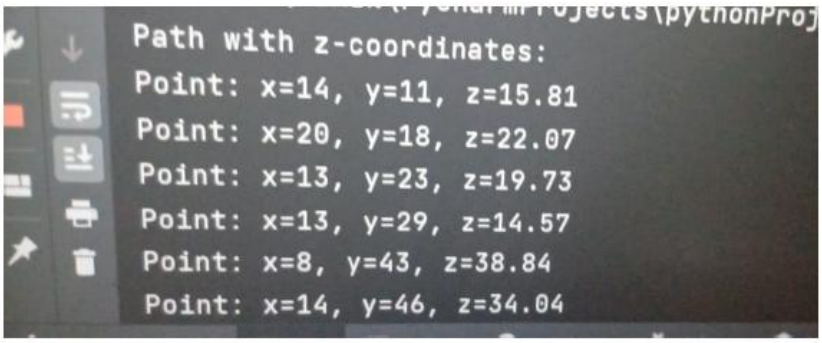


Figure 5: Coordinate Mapping

V-REP Simulator

V-REP (Virtual Robot Experimentation Platform) is a simulation software commonly used in robotic development. In the context of surgical robots, V-REP allows for the testing and visualization of path planning algorithm in a virtual environment before applying them to real-world scenarios. It provides a realistic 3D model of the surgical site and the robotic system, helping researchers test various planning strategies, refine algorithms, and simulate potential risks and errors before performing the actual surgery.

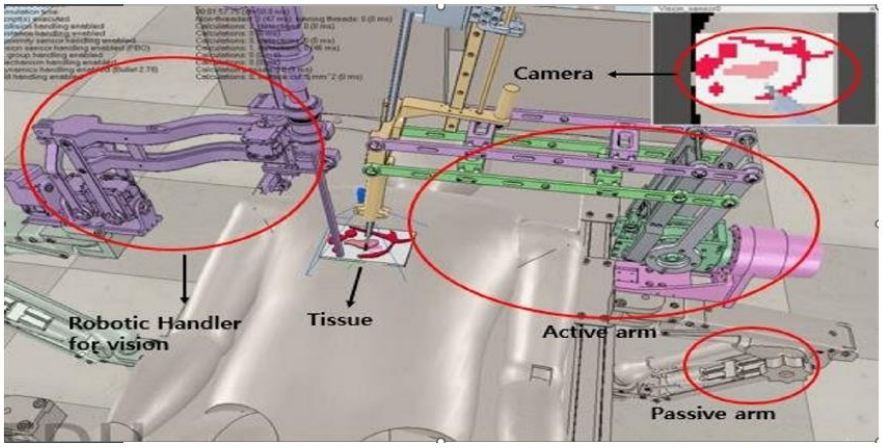
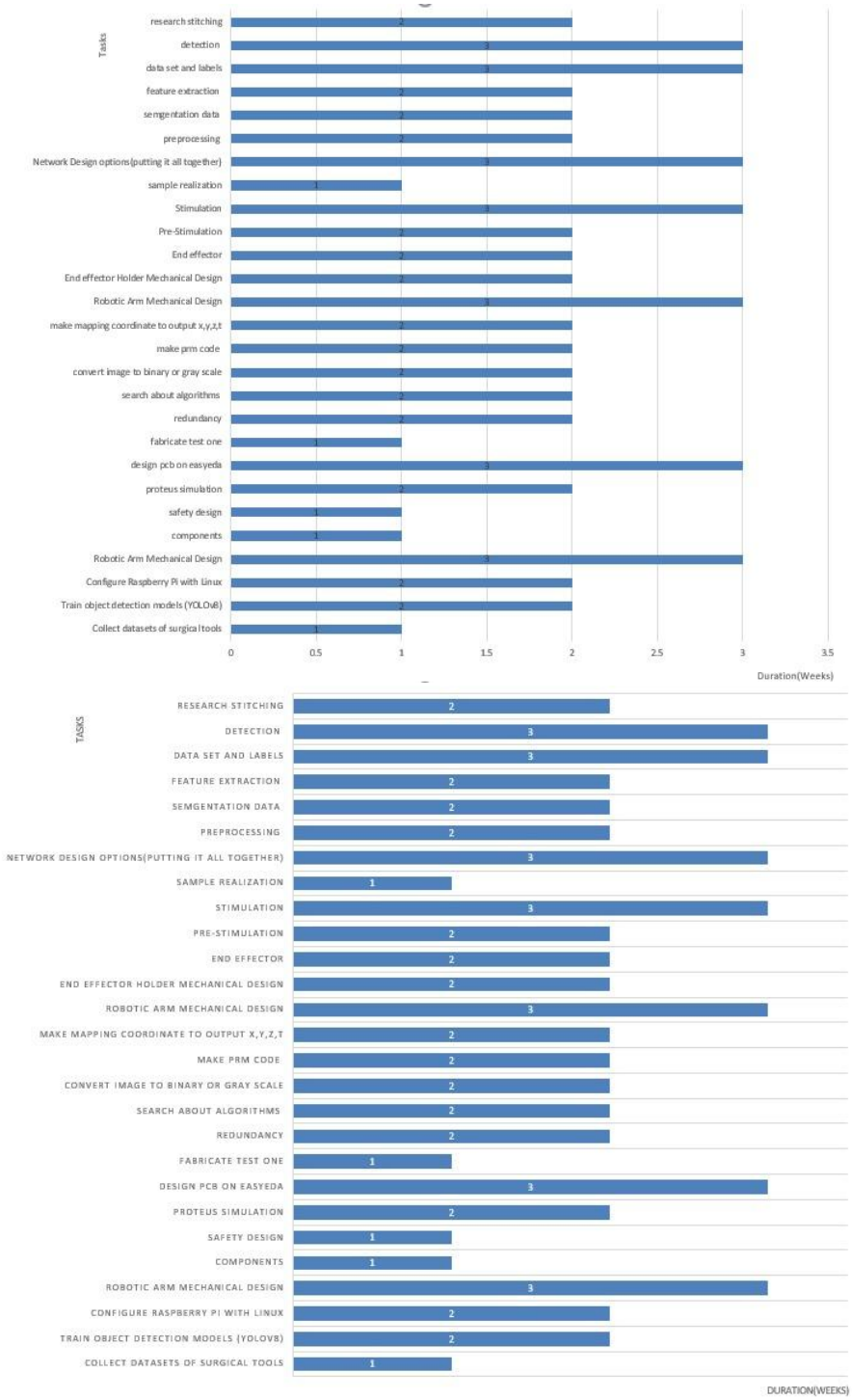


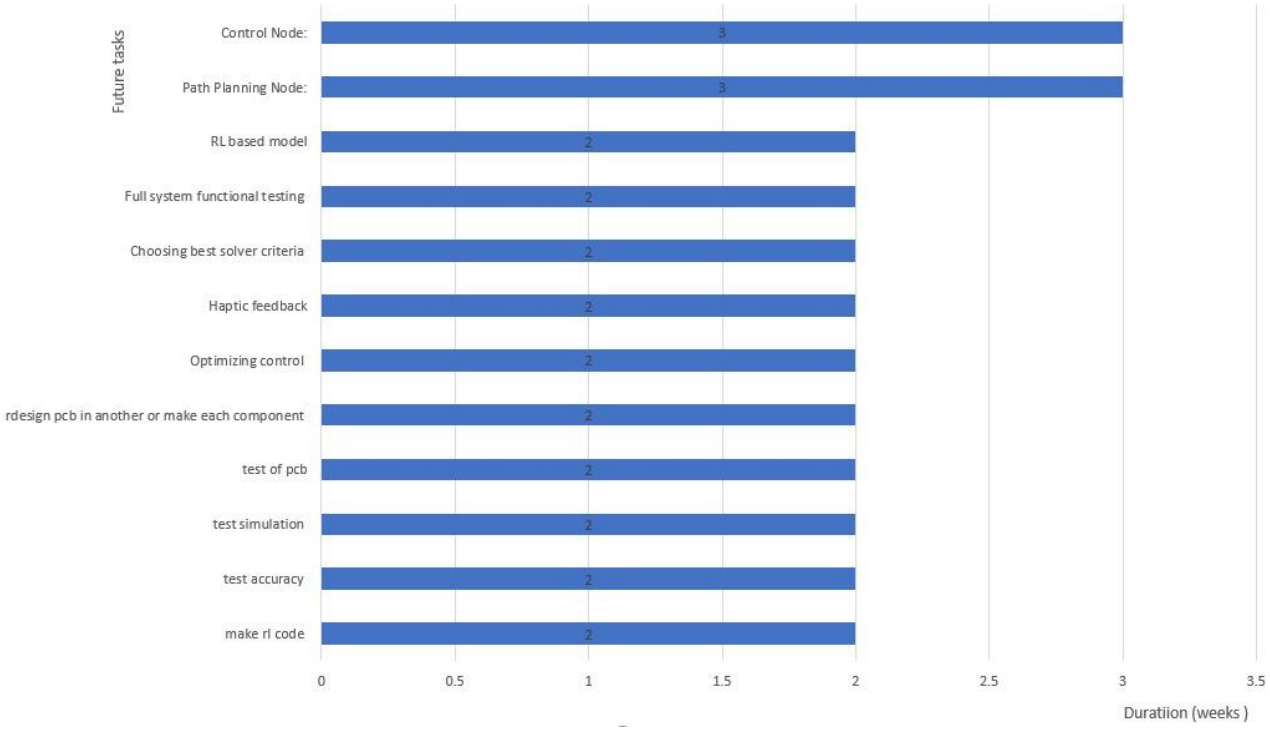
Figure 6: V-REP SimulatoR

7. Project Timeline:

Tasks done



Future tasks:



8. Cost analysis

Item	Type (Hardware/ Software/ Other)	Part in the Block Diagram	Specifications	Quantity	Price in EGP
3d printed design prototype	Hardware	Arm station	3d printed prototype	1	7000
Power supply	Hardware	Control station	Power supply 24v 30 A	1	1781
Motor Drive	Hardware	Servo driver	Cytron 20A 6V-30V DC Motor Driver	1	2600
DC-DC step down Converter	Hardware	Arm station	DC-DC step down Converter	2	350
Current sensor	Hardware	Pcb in control station	ACS758LCB	2	500
Voltage sensor	Hardware	Pcb in control station	LV 25-P	2	2000
Bearings	Hardware	Arm station	Bearing	4	800
Tiva c	Hardware	control station	TM4C1294	1	6350
Stm	Hardware	control station	Nucleo-64 STM32F446	1	1200
Raspberry Pi 4	Hardware	Raspberry Pi 4	Computer Model B – 8GB RAM	1	6000
Motor	Hardware	Arm station	150 kg.cm torque	1	2700
Motor	Hardware	Arm station	45 kg.cm torque	2	2700
Motor	Hardware	Arm station	350 kg.cm torque	1	5000
End effector	Hardware	Arm station	Intuitive Surgical Da Vinci Curved scissors 8 mm 420178	2	5300
Camera	Hardware	Arm station	OAK-D Lite Robotics Camera -Auto focus	1	7500
Joystick	Hardware	Arm station	Ps4 controller	1	1000
PCB	Hardware	Arm station	Prototype PCB	3	1800
PCB	Hardware	Arm station	Final PCB manufactured	3	4500
Grand Total					**59081**

9. Conclusion:

The implementation of the Da Vinci robot surgical system in Egypt marks a significant milestone in the country's healthcare journey; to represent a transformative step in elevating the standards of healthcare. As the first of its kind in our region, this initiative by our graduation team, not only enhancing the surgical precision and patient outcomes, but also modernizing surgical practices across various specialties.

By focusing on training, clinical integration, community outreach, quality assurance and safety, the project will improve the overall quality of care, attract medical professionals and establish Egypt as a pioneer in advanced robotic-assisted surgery. We invite collaboration and support from all companies to make this vision a reality and improve patients'' lives across the nation.

10. References:

https://learnopencv.com/introduction-to-opencv-ai-kit-and-depthai/

https://learnopencv.com/stereo-vision-and-depth-estimation-using-opencv-ai-kit/

https://core-electronics.com.au/guides/oak-d-lite-raspberry-pi/

https://learnopencv.com/object-detection-with-depth-measurement-with-oak-d/

https://robofoundry.medium.com/oak-d-lite-camera-ros2-setup-1e74ed03350d

https://www.youtube.com/playlist?list=PLfYPZalDvZDLOjzSkoHQ2_h4joHNUegbB

SAM (Segment Anything Model) - Ultralytics YOLO Docs

Introducing Segment Anything: Working toward the first foundation model for image segmentation

https://www.youtube.com/watch?v=1sz5fhO8jxA&list=LL&index=1&t=550s&pp=gAQBiAQB

https://www.youtube.com/playlist?list=PLv8Cp2NvcY8ClWpGlPJ9tmBmUhlA94Umy

Image Processing Using OpenCV - YouTube

The Complete Guide to Image Preprocessing Techniques in Python | by Maahi Patel | Medium

Image processing with python - YouTube