Computer Vision Based Pick and Place Robot

Introduction:
Computer vision based pick and place robots are very popular nowadays for both educational and professional uses. Even though the main concept of the project is identical for any field of applications, the differences affect significantly. The similarity is that in both cases the robot should look at the environment and detect certain objects and grab the objects for relocation if necessary. The differences are not always the same but most of the time they are the cost, precision, efficiency and safety between educational use and professional use. The reason being that usually educational projects are focused on research and development of the robot. This requires lots of testing and replacing the parts and algorithms with a better one. For this reason initially especially for hobby project stage the robot has to be relatively cheap and affordable. This also effects on the precision and efficiency of the robot at least initially. The safety being a subject to improve throughout the research and development is not as good as the professional ones. On the other hand the professional robots has to be precise, efficient and safe or at least with proper safety instructions for the consumer. The cost is much higher for this reason. In our case the robot is initially at research and development state until it is safe and precise enough for professional uses being also as affordable as it can be. The area of reach of the robot also known as the “work envelope” and the weight carrying capability or “payload” are also important factors here. A spherical work envelope with radius closest to the arm length and a minimum payload of 1kg is desirable in our project.

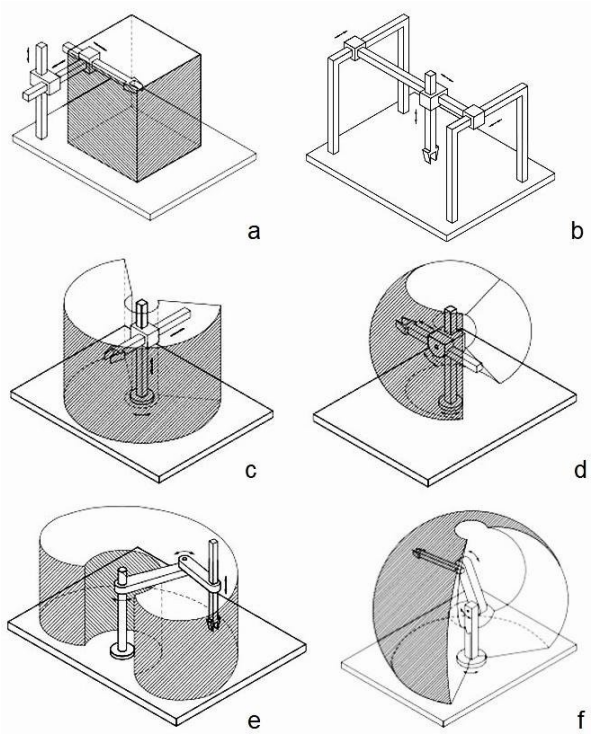


Figure 01: Different types of robotic arms and their work envelope

Requirements: For the project we are going to need some hardware components for both processing and mechanical works. For the instruction for the processing, controlling and monitoring we also need some editors, compilers and most importantly an operating system to run the programs.

Hardware:

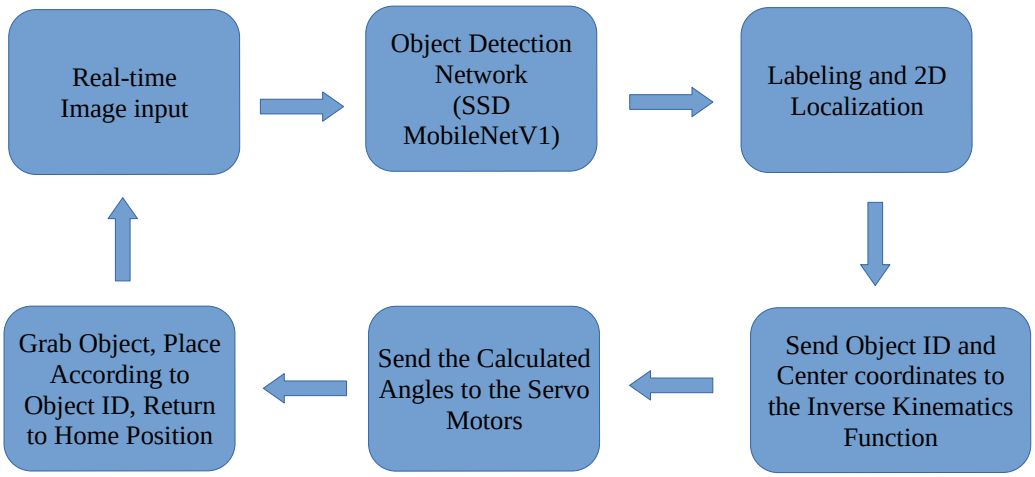
1. Jetson Nano
2. Arduino Mega
3. Webcam
4. Camera stand
5. Power supply
6. 6 Dof robotic arm
7. Servo motors
8. Servo driver
9. WiFi dongle
10. Micro SD card
11. Keyboard, Mouse, Monitor etc.

Software:

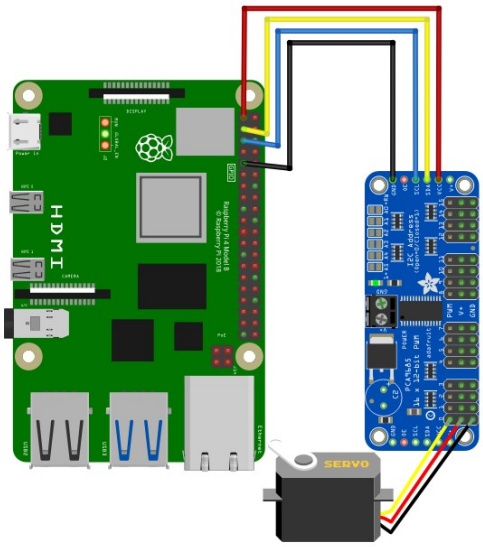
1. Balena etcher
2. Visual Studio Code
3. Arduino IDE

System Architecture:

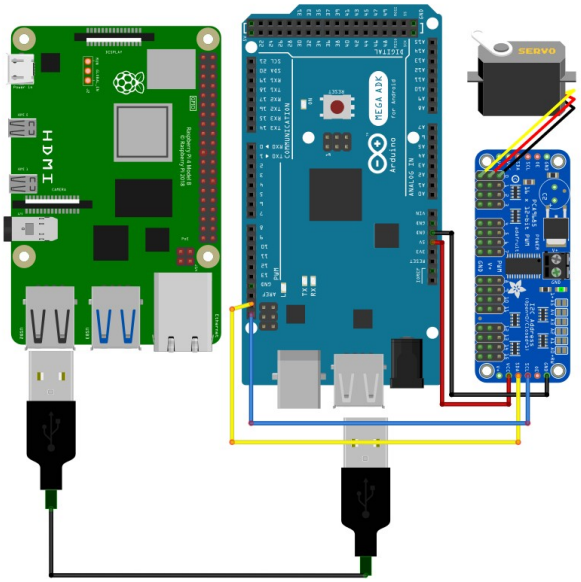
Block Diagram:



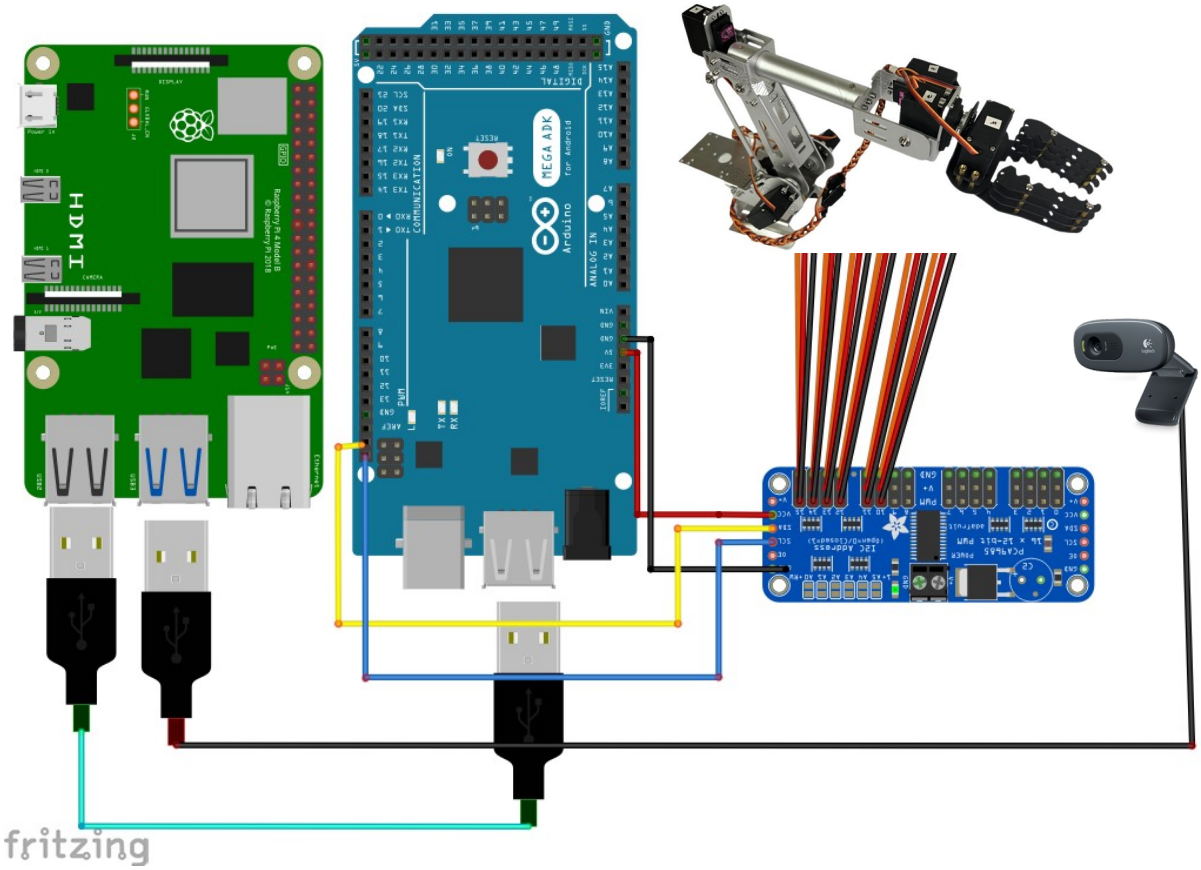
Circuit Diagram:



1. Without Arduino



2. With Arduino



3. Complete Circuit Diagram for Computer Vision based Pick and Place

To connect the driver we can choose from 1 and 2 of the above diagrams. For simple projects with few modules either of them is valid configuration. However, as we increase the complexity of task and hardware to control, the later configuration is much preferred. This will help to run the resource intensive tasks in the jetson nano and the controlling operations in the separate microcontroller unit. This will also make the design more modular and easy to troubleshoot. We may have to deal with little bit more complexity for additional communication functions with as low latency as possible. For this reason we have followed the second diagram and 3 shows the complete circuit diagram for this project excluding the power supply system.

Getting Started:

Setting Up Jetson Nano:

Note: Before you begin, make sure you have the following items:

1. Jetson Nano 4GB Developer Kit
2. MicroSD card (minimum 32GB recommended)
3. MicroSD card reader
4. Computer for flashing the SD card
5. Additional Monitor / Display with HDMI support, USB Wi-Fi adapter, USB webcam, Keyboard and Mouse
6. 5V/4A DC power supply with a barrel jack (5.5mm x 2.1mm) or a 5V USB-C power supply with a minimum of 3A current

1. Flash the MicroSD Card

1. Before setting up the Jetson Nano, you need to flash the MicroSD card with the NVIDIA JetPack SDK. This software includes the operating system and other development tools.

2. Download the latest JetPack SD Card Image for Jetson Nano from NVIDIA's website (https://developer.nvidia.com/jetson-nano-sd-card-image).

3. Insert the MicroSD card into your computer using a card reader.

4. Use a tool like Balena Etcher or dd command to flash the downloaded image onto the MicroSD card. **Make sure you select the correct device to avoid accidental data loss.**

2. Assemble the Hardware

1. Connect a display to the Jetson Nano via HDMI. Make sure your monitor or display is powered on.

2. Attach a USB keyboard and mouse to the USB ports. Attaching a USB webcam is also required but only for the detection and not for the initial setup.

3. Connect an Ethernet cable to the Gigabit Ethernet port if you have access to a wired network. However, we are going to connect to Wi-Fi using a USB Wi-Fi adapter.

4. Insert the flashed MicroSD card into the MicroSD card slot.

5. Connect the power supply to the Jetson Nano's power jack (5.5mm x 2.1mm) or to the USB-C port (ensure it can deliver at least 3A).

6. The Jetson Nano will power on automatically when you plug it in. You should see the NVIDIA logo and boot messages on your display.

3. Initial Setup

1. Follow the on-screen instructions to set your system language, keyboard layout, and other basic settings.

2. Create a user account and set a password.

3. Connect to the internet (if you're using Wi-Fi, you will need to configure it at this stage).

4. The Jetson Nano will download and install any available updates. This may take some time.

4. Getting Started

1. Once the setup is complete, you'll be presented with the Ubuntu desktop environment. You can start using your Jetson Nano.

2. To access additional resources, documentation, and support, visit the NVIDIA Jetson Nano developer page (https://developer.nvidia.com/embedded/jetson-nano-developer-kit).

3. You can also install development tools, libraries, and frameworks using the JetPack SDK Manager, which is typically pre-installed on your Jetson Nano.

4. Congratulations, the Jetson Nano is now set up and ready for the project!

Setting Up Robotic Arm:

The following items are required for this step:

1. The robotic arm parts, bearings and screws
2. 6 / 7 MG996R servo motors
3. PCA9685 16-channel I2C servo motor driver
4. Arduino Mega
5. 5V power supply (can be the same one used for powering the Jetson Nano if its 4A+)

The robotic arm used here is already setup by the supplier including the 7 servo motors of 10kg.cm torque. However, Depending on how you mount the arm you may have to upgrade one or two servo motors to a minimum 20kg.cm torque servo motors. In our case we mounted the arm to a robots sidewall attaching the base of the arm vertically. For this one servo at the shoulder had to be upgraded to support the arm weight.

There are 7 set of connecting wires for the 7 servo motors. Our servo driver has 16 channels which means we should be able to connect and control up to 16 servo motors separately with a single driver. We can connect the connectors in any sequence we want but we have to remember the channel id (0 – 15) for the motors.

Connect the driver to the Jetson Nano or to the Arduino that is connected to the jetson nano as shown in the above circuit diagrams. The later method is applied in this project but both are valid for simple tasks. It is preferred to use a separate power line of 4.8 - 6 volts to the servo driver.

Software Implementation:

To run the detection model we have to set the environment first. We have already gone through the initial setup of the Jetson Nano. Now we have to run the following commands in order to build our environment (considering python3 is installed by default):

Environment Setup:

1. Environment update:
   sudo apt-get update

2. Dialog package installation:
   sudo apt-get install dialog

3. Necessary libraries including numpy download and installation:
   sudo apt-get install git cmake libpython3-dev python3-numpy

4. Cloning into jetson inference repository:
   git clone --recursive https://github.com/dusty-nv/jetson-inference

5. Environment build, make and installation:
   cd jetson-inference
   mkdir build
   cd build
   cmake ../
   make
   sudo make install
   sudo ldconfig

Object Detection:

Data preparation: Data collected from Open Images data-set; where a list of pick-able fruits and vegetables are gathered. **12** classes are included in the detection data-set where 11 classes are gathered from the mentioned data-set and **Onion** class is processed from self processing and annotation since it was not available in the mentioned data-set. Annotation is in "**Open images**" format. Total **1126** images are used where train, validation and test ratio is:

Train: **899**
Validation: **65**
Test: **162**

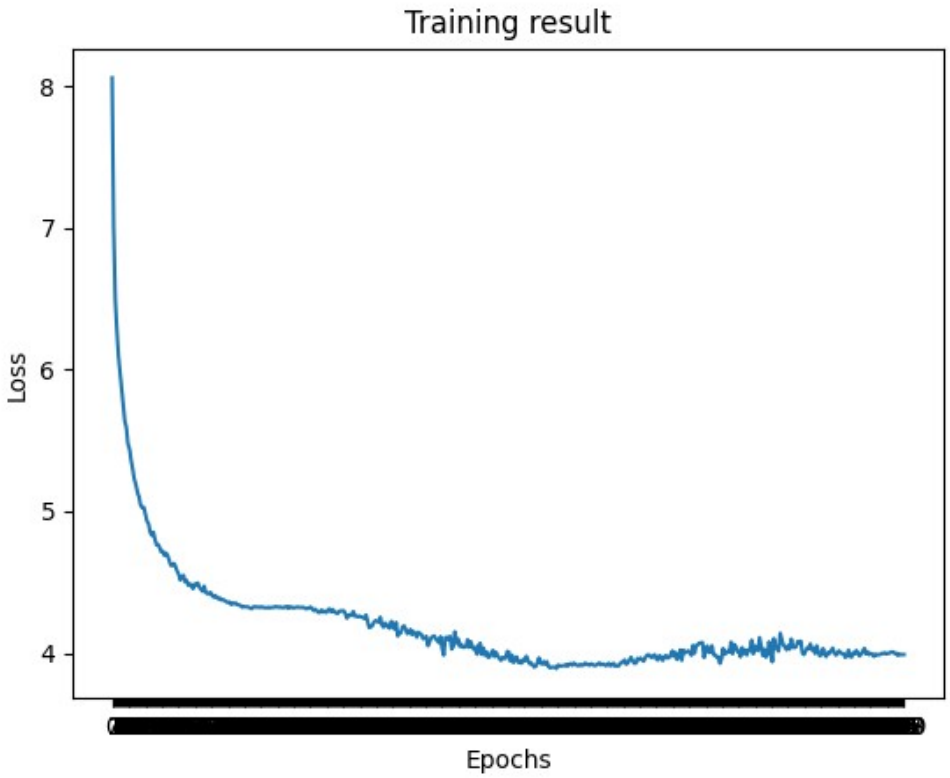
Around **94** images are used per class for data balancing. Annotation was interchanged in below steps:

Model training: Model is trained using SSD-Mobilenet-v1 detection process applying **300** epochs and **256** batches. Parameters used are:

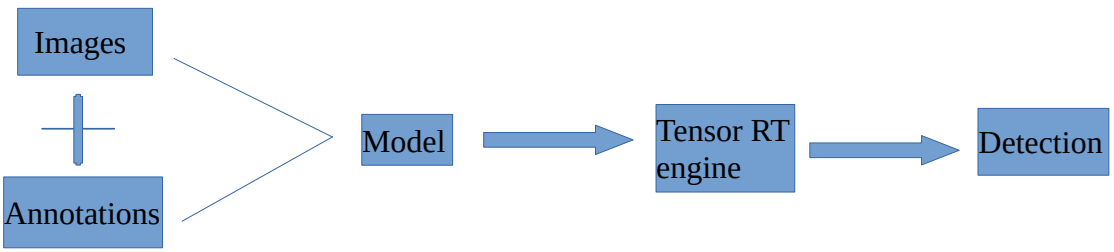
Resolution: **(300, 300)**
Learning rate: **0.01**
Momentum: **0.9**
Weight-decay: **5e-4**
Gamma: **0.1**

Detection performance: The best performance recorded is a loss amount of **3.88** after crossing **280** epochs.

Loss curve:



Block diagram:



Pick and Place:

Robotic arm: We are going to use a 6-dof (degree of freedom) robotic arm. We can initially use the same arm for 2-dof configuration simply by ignoring the joints other than the shoulder and the elbow of the arm. The reason for start by 2-dof configuration is to understand the kinematics of the arm from simpler solutions. As the number of joints increases so does the complexity of the kinematics. After we finish with the 2-dof configuration we can then move on to the 6-dof configuration.

Usually the base of a robotic arm is mounted on a horizontal plane of a desk. But it can also be mounted vertically or upside down. In our case it is mounted vertically to the stand of our robot.

Kinematic Diagram: We will use two kinematic diagram for the arm. One is for the 2-dof configuration and the other one is the actual kinematics diagram for the 6-dof robotic arm. Both of the diagrams are drawn below:

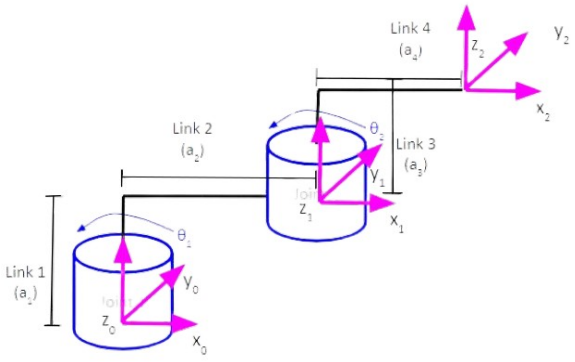


Figure: Kinematic diagram of a SCARA-type 2 DOF Robotic Arm

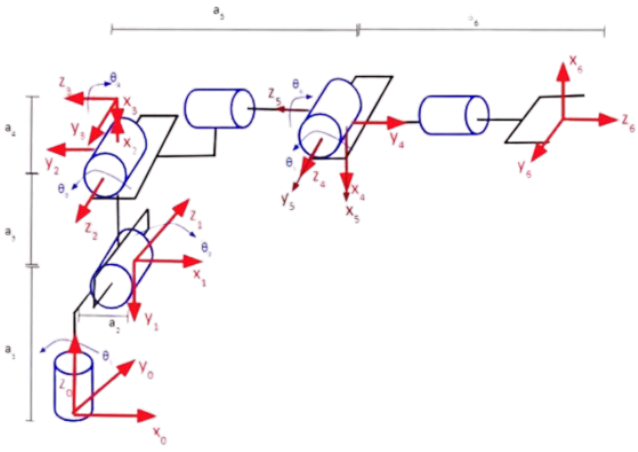


Figure: Kinematic diagram of a 6 DOF Robotic Arm

Kinematics: In the previous section we drew the kinematic diagram of our robotic arm. But what is the point of this kinematic diagram and why does the kinematics of the arm matter? If we want to get better control over our arm it is important to know that where does the gripper ends up if we rotate each joint in certain angles or what angles should we apply to the joint to get our gripper to the point we want. Kinematic equations tells us exactly that. We can apply some known variables to these equations and get the angles or coordinates. We use Forward Kinematics and Inverse Kinematics for this purpose. We will talk about both of them but in our case we will focus mainly on the Inverse Kinematics. There are some key points we need to know first.

Transformation Matrix: Each frame of a robotic arm has its own rotation in 1 or 2 or 3 axis in space. It also has a linear displacement in the 1 or 2 or 3 axes also called translation. The transform matrix of a frame of a robotic arm represents the rotation and the translation of that frame. The robotic arm we are going to use can move in 3 axes so the transformation matrices of the arm will represent its rotations and translations in 3D space.

There are more than one way to find the transform matrices of a robotic arm. We used a popular method that uses the Denavit-Hartenberg (D-H) Parameter table to find the transform matrices.

D-H Parameters: Using the Denavit-Hartenberg Parameter table is a shortcut for finding homogeneous transformation matrices and is commonly seen in documentation for industrial robots as well as in the research literature. We need the length of each link and the angle of each servo motor to calculate the transformation matrices in this method. There are three steps to find the D-H parameter table and find the homogeneous transformation matrices from that:

1. Draw the kinematic diagram according to the four Denavit-Hartenberg rules
2. Create the Denavit-Hartenberg parameter table where:
    - Number of Rows = Number of Frames – 1
    - Number of Columns = 4 (2 columns for rotation and 2 columns for displacement)
3. Find the homogeneous transformation matrices

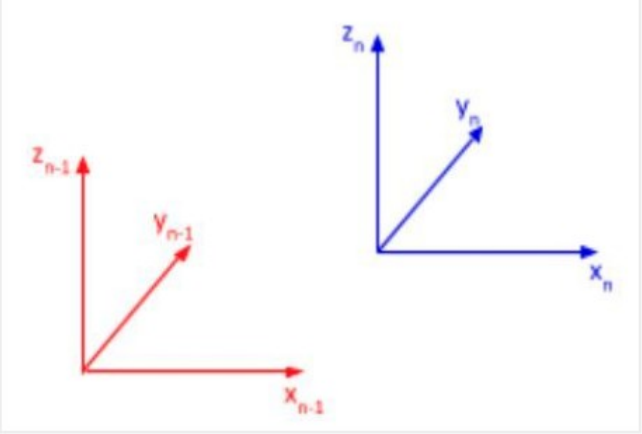
Definition of the Parameters: The Denavit-Hartenberg parameter tables consist of four variables:

- The two variables used for rotation are θ and α
- The two variables used for displacement are r and d

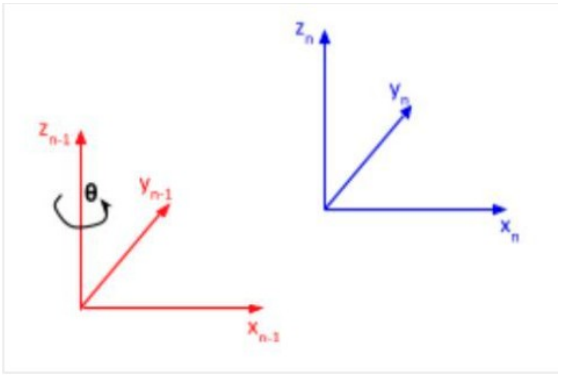
Here is the D-H parameter table template for a robotic arm with four reference frames:

Joint i	θ_i (deg)	α_i (deg)	r_i (cm)	d_i (cm)
1				
2				
3				

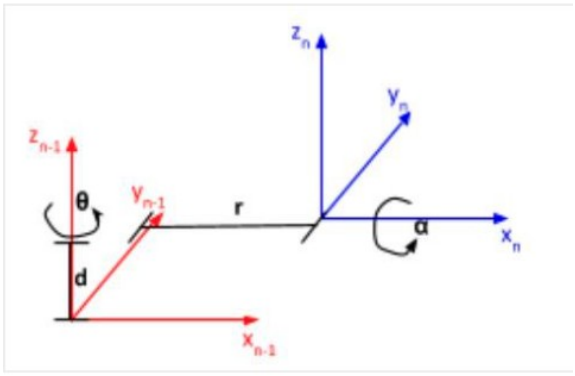
Let’s take a look at what these parameters mean by looking at two different frames. The n-1 frame is the frame before the n frame. We assume that both frames are connected by a link



θ is the angle from x_n-1 to x_n around z_n-1.

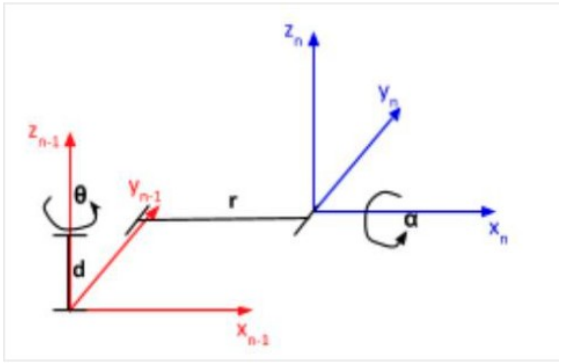


α is the angle from z_n-1 to z_n around x_n.



r / a is the distance between the origin of the n-1 frame and the origin of the n frame along the x_n direction.

d is the distance from x_n-1 to x_n along the z_n-1 direction.



Transform Matrices from The Denavit-Hartenberg parameters: After we have found all the parameters, we can find the Transform matrices for each frame from the below equation which is the homogeneous transformation matrix for joint n:

homgen_n-1_n =

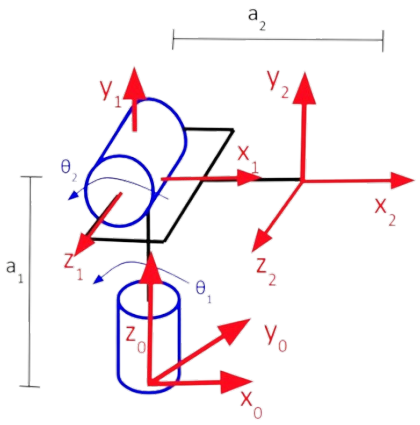
$$\begin{bmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & R & &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

R=Rotation
T=Translation

(image source: https://automaticaddison.com/how-to-find-denavit-hartenberg-parameter-tables/)

For Example, lets find the D-H parameter table for a two degree of freedom robotic arm (any valid 2-DOF configuration):

Draw the Kinematic Diagram According to the Denavit-Hartenberg Rules



According to the descriptions of the parameters, we find θ, α, r and d:

θ:
$\mathbf{x_0}$ and $\mathbf{x_1}$ both point in the same direction. The axes are therefore aligned. When the robot is in motion, $\boldsymbol{\theta_1}$ will change (which will cause **frame 1** to move relative to **frame 0**). The angle from $\mathbf{x_0}$ to $\mathbf{x_1}$ around $\mathbf{z_0}$ will be $\boldsymbol{\theta_1}$.
$\mathbf{x_1}$ and $\mathbf{x_2}$ both point in the same direction. The axes are therefore aligned. When the robot is in motion, $\boldsymbol{\theta_2}$ will change (which will cause **frame 2** to move). The angle from $\mathbf{x_1}$ to $\mathbf{x_2}$ around $\mathbf{z_1}$ will be $\boldsymbol{\theta_2}$.

α:
$\boldsymbol{\alpha_1}$ is the angle from $\mathbf{z_0}$ to $\mathbf{z_1}$ around $\mathbf{x_1}$. In the diagram above, this angle is **90** degrees.
$\boldsymbol{\alpha_2}$ is the angle from $\mathbf{z_1}$ to $\mathbf{z_2}$ around $\mathbf{x_2}$. We can see that no matter what happens to $\boldsymbol{\theta_2}$, the angle from $\mathbf{z_1}$ to $\mathbf{z_2}$ will be **0** (since both axes point in the same direction).

r:
$\mathbf{r_1}$ is the distance between the origin of **frame 0** and the **origin of frame 1** along the $\mathbf{x_1}$ direction. In the diagram that this distance is **0**.
$\mathbf{r_2}$ is the distance between the origin of **frame 1** and the **origin of frame 2** along the $\mathbf{x_2}$ direction. In the diagram that this distance is $\mathbf{a_2}$.

d:
$\mathbf{d_1}$ is the distance from $\mathbf{x_0}$ to $\mathbf{x_1}$ along the $\mathbf{z_0}$ direction. In the diagram that this distance is $\mathbf{a_1}$.
$\mathbf{d_2}$ is the distance from $\mathbf{x_1}$ to $\mathbf{x_2}$ along the $\mathbf{z_1}$ direction. In the diagram that this distance is **0**.

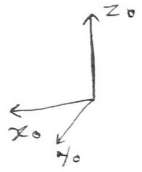
Finally, we get the Denavit-Hartenberg Parameter Table:

Joint i	θ_i (deg)	α_i (deg)	r_i (cm)	d_i (cm)
1	θ_1	90	0	a_1
2	θ_2	0	a_2	0

Now, lets find the D-H Parameters for our model. From the structure, we measured the values (in centimeters) for **r** and **d**:

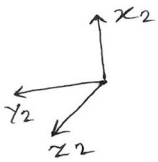
r1 = a_0 = 4, **d1 = a_1 = 8,**
r2 = a_2 = 12, **d2 = 0,**
r3 = 0, **d3 = 0,**
r4 = 0, **d4 = a_3 + a_4 = 0 + 17 = 17,**
r5 = 0, **r6 = 0,**
r6 = 0, **d6 = a_5 + a_6 = 4 + 5 = 9**

For θ and α, lets look at their respective orientations found similarly from the kinematic diagram (reduced for convenience):

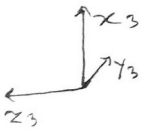


Frame 1

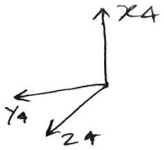
Frame 2



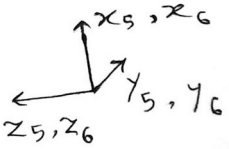
Frame 3



Frame 4



Frame 5



Frame 6

θ:

x_0 and x_1 both point in the same direction. The axes are therefore aligned. When the robot is in motion, θ_1 will change (which will cause **frame 1** to move relative to **frame 0**). The angle from x_0 to x_1 around z_0 will be θ_1.
x_1 and x_2 are perpendicular to each other. θ_2 has to change 90° clock-wise around z_1 to align x_1 and x_2 (which will cause **frame 2** to move relative to **frame 1**). So, The angle from x_1 to x_2 around z_1 will be θ_2-90°.
Rest of the frames are related similar to **frame 1** and **frame 0**. Thus the angles are θ_3, θ_4, θ_5, θ_6.

α:

α_1 is the angle from z_0 to z_1 around x_1. In the diagram above, this angle is **-90** degrees (clock-wise angle around x_1).
α_2 is the angle from z_1 to z_2 around x_2. We can see that no matter what happens to θ_2, the angle from z_1 to z_2 will be **0** (since both axes point in the same direction).
α_3 is the angle from z_2 to z_3 around x_3. In the diagram above, this angle is **-90** degrees.
α_4 is the angle from z_3 to z_4 around x_4. In the diagram above, this angle is **90** degrees (counter-clock-wise angle around x_4).
α_5 is the angle from z_4 to z_5 around x_5. In the diagram above, this angle is **-90** degrees.
α_6 is the angle from z_5 to z_6 around x_6. We can see that no matter what happens to θ_6, the angle from z_5 to z_6 will be **0** (since both axes point in the same direction).

We found all the parameters. Now we can put the values in the table with number of rows = number of frames - 1 = 6 - 1 = 5:

Joint i	θ_i (deg)	α_i (deg)	r_i (cm)	d_i (cm)
1	θ_1	-90	4	8
2	θ_2-90	0	12	0
3	θ_3	-90	0	0
4	θ_4	90	0	17
5	θ_5	-90	0	0
6	θ_6	0	0	9

Now we can find the Transform Matrices for the 6 joints applying the D-H parameters corresponding to the joints to the homogeneous transformation matrix below:

homgen_n-1_n =

$$\begin{bmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & & & T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T0_1 =

$$\begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 4\cdot\cos(\theta_1) \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 4\cdot\sin(\theta_1) \\ 0 & -1 & 0 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T1_2 =

$$\begin{bmatrix} \cos(\theta_2 - 90°) & -\sin(\theta_2 - 90°) & 0 & 12\cdot\cos(\theta_2 - 90°) \\ \sin(\theta_2 - 90°) & \cos(\theta_2 - 90°) & 0 & 12\cdot\sin(\theta_2 - 90°) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T2_3 =

$$\begin{bmatrix} \cos(\theta_3) & 0 & -\sin(\theta_3) & 0 \\ \sin(\theta_3) & 0 & \cos(\theta_3) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T3_4 =

$$\begin{bmatrix} \cos(\theta_4) & 0 & \sin(\theta_4) & 0 \\ \sin(\theta_4) & 0 & -\cos(\theta_4) & 0 \\ 0 & 1 & 0 & 17 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T4_5 =

$$\begin{bmatrix} \cos(\theta_5) & 0 & -\sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & \cos(\theta_5) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T5_6 =

$$\begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & 0 \\ \sin(\theta_6) & \cos(\theta_6) & 0 & 0 \\ 0 & 0 & 1 & 9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T0_6 = T0_1 . T1_2 . T2_3 . T3_4 . T4_5 . T5_6

Rotation Matrix: We said before that a transform matrix represents a rotation and a translation. In our case we found some 4x4 matrices. The top left 3x3 matrix of each transformation matrices represent the rotation of the corresponding frame in 3D space. These 3x3 matrices are called the rotation matrices.

The top right 1x3 matrices represent the x, y and z coordinates of the frame in space. It is also known as the **translation vector.**

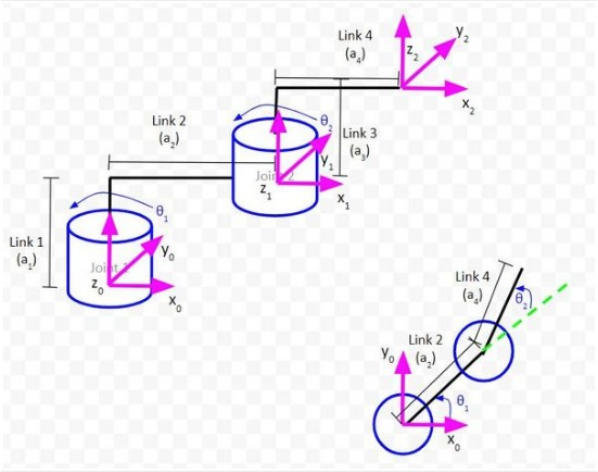
Forward Kinematics: Forward kinematics determines the position and orientation of the robotic arm using some given joint parameters that include joint angles. We find these parameters from the transform matrices. In this project we are not going to use forward kinematics.

Inverse Kinematics: Just like we can use forward kinematics to find the position and orientation of a robotic arm from its joint parameters, Inverse kinematics determines the joint angles from given other joint parameters, orientation and position of the “End effector” (our gripper). The matrices will help to get those parameters. We are going to use Inverse kinematics and give our arm the position and orientation we want the gripper to go. Then we can send the determined joint angles to the servo motors of the arm.

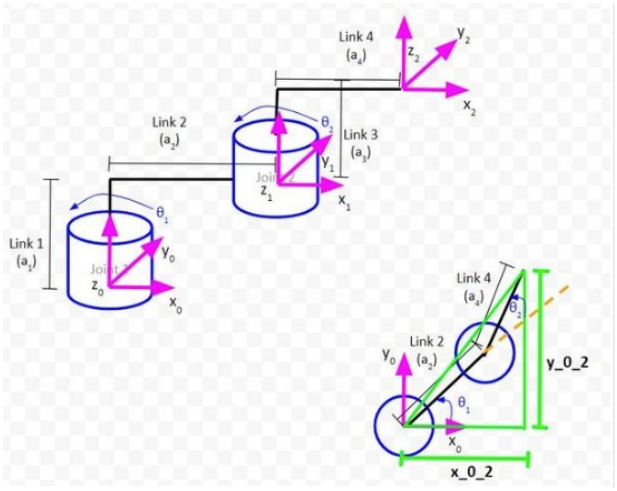
We said previously that we will discuss two different inverse kinematics for our robotic arm. At first we will look at the inverse kinematics for its 2-DOF configuration. This will only help the arm to move in 2 axes but help us understand the method better. Then we will discuss about the Inverse kinematics for the 6-DOF configuration.

I-K for 2-DOF Configuration:

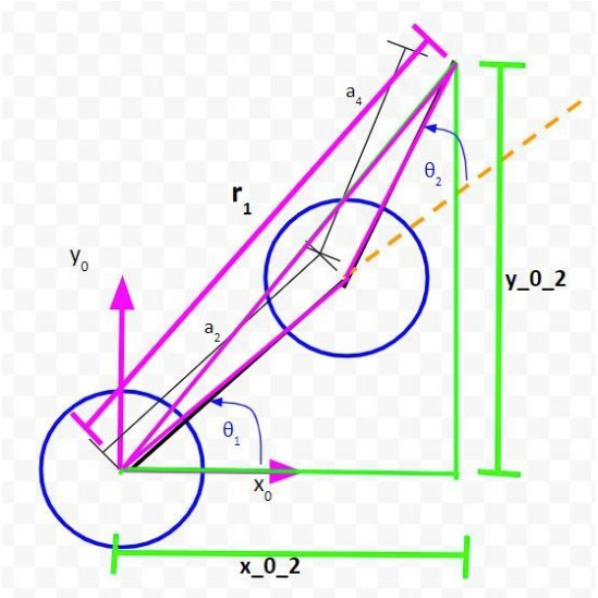
Draw the Kinematic Diagram From Either an Aerial View or Side View:



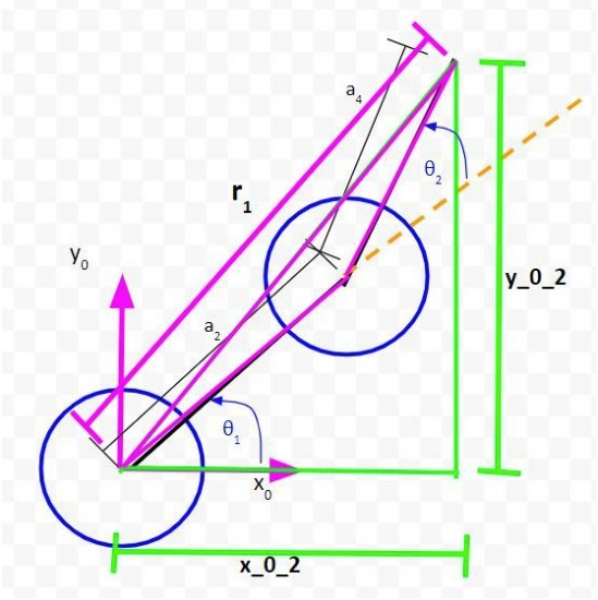
Sketch Triangles on Top of the Kinematic Diagram. Let's draw the first triangle in green:



Let's zoom in and draw another triangle. We'll make this one pink. Let the long side of the triangle we just created be r_1:



Solve for r_1:

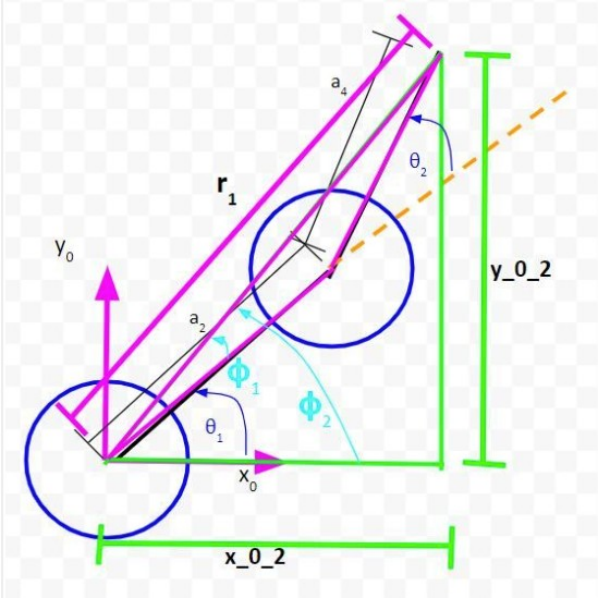


Let's start by looking at the green right triangle above. To solve for r_1, we can use the Pythagorean Theorem.
$r_1^2 = (x\_0\_2)^2 + (y\_0\_2)^2$

Take the square root of both sides. We'll label this as equation 1:
(1) r_1 = square_root($(x\_0\_2)^2 + (y\_0\_2)^2$)

Solve for θ_1:

Now let's look at the pink triangle. Because this isn't a right triangle, we'll have to use the law of cosines. Lets add two new angles, ϕ_1 and ϕ_2:



We can see from the diagram that:
$\theta_1 = \phi_2 - \phi_1$

Using the law of cosines, we have:
$a_4^2 = r_1^2 + a_2^2 - 2r_1a_2\cos(\phi_1)$

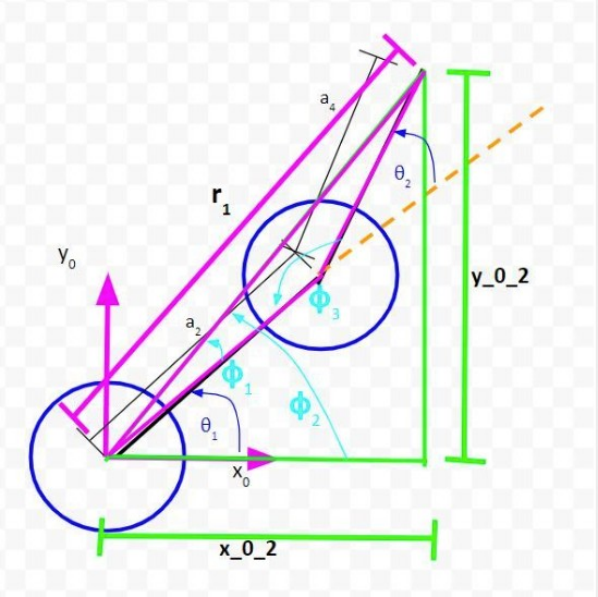
Now, solve for ϕ_1. This will be equation #2:
(2) $\phi_1 = \arccos((a_4^2 - r_1^2 - a_2^2)/(-2r_1a_2))$

Also, we can see from the diagram that:
(3) $\phi_2 = \arctan((y\_0\_2) / (x\_0\_2))$

Now that we have expressions for ϕ_1 and ϕ_2, we can solve for θ_1.
(4) $\theta_1 = \phi_2 - \phi_1$

Solve for θ_2:

To solve for θ_2, we need to label yet another angle. I'll label that obtuse angle in the pink triangle ϕ_3:



Looking at the diagram above we can see that:
$\phi_3 + \theta_2 = 180°$

Therefore,
$\theta_2 = 180° - \phi_3$

To find the value of $\phi 3$, we need to use the law of cosines.
$r_1^2 = a_2^2 + a_4^2 - 2a_2a_4\cos(\phi_3)$

Solving for $\phi 3$, we get:
(5) $\phi_3 = \arccos((r_1^2 - a_2^2 - a_4^2)/(-2a_2a_4))$

Now, we can solve for θ_2:
(6) $\theta_2 = 180° - \phi_3$

θ_1 and **θ_2** are the two angles we can send the shoulder and elbow joint to **x_0_2** and **y_0_2**.

I-K for 6-DOF Configuration: