

# **IMAGE BASED VISUAL SERVOING (IBVS)**

Summer Internship Report

By

**Dhruv Gupta**

Narsee Monjee Institute of Management and Studies (NMIMS)

Under the Supervision of

**Dr. AMIT SHUKLA**

**(Chairperson)**

**CAIR, IIT Mandi**



**CENTRE FOR ARTIFICIAL INTELLIGENCE AND  
ROBOTICS (CAIR)**

**JUNE 2024**

# Abstract

This report explores the field of image-based visual servoing (IBVS), a subset of visual servo control where the objective is to control the motion of a robotic system using feedback from visual data. The study provides a comprehensive overview of the principles underlying IBVS, including the extraction and processing of visual features from images and the use of these features to guide the robot's movements. We delve into the mathematical frameworks that support IBVS, highlighting key techniques such as image Jacobians and interaction matrices. Various algorithms and control strategies are evaluated, demonstrating their effectiveness in different scenarios, such as dynamic and static environments. The report also examines the practical applications of IBVS in areas like industrial automation, autonomous navigation, and medical robotics, illustrating its potential to enhance precision and efficiency. Through simulations and real-world experiments, we assess the performance of IBVS systems, addressing challenges such as robustness to environmental changes and computational efficiency. Finally, we discuss future directions and advancements in the field, emphasizing the integration of artificial intelligence and machine learning to further improve the adaptability and intelligence of IBVS systems.

## Abbreviations:

<b>IBVS</b>	Image-Based Visual Servoing
<b>VS</b>	Visual Servoing
<b>DOF</b>	Degree of Freedom
<b>ROS</b>	Robot operating system
<b>UAV</b>	Unmanned Aerial Vehicle
<b>AGV</b>	Automated Guided vehicle

# Acknowledgement

I would like to express my sincere gratitude to everyone who contributed to the completion of this report on image-based visual servoing (IBVS). First and foremost, I extend my deepest appreciation to my advisor, Karan Raj Pant, for their invaluable guidance, support, and encouragement throughout this research. Their expertise and insights were instrumental in shaping the direction and scope of this study. I would also like to thank the Chairperson of the Center in Artificial Intelligence and Robotics (CAIR) Mr. Amit Shukla for providing the necessary resources and facilities to conduct this Internship. Special thanks to my colleagues and peers for their collaborative spirit and for offering helpful feedback during our discussions. Additionally, I am grateful to the authors and researchers whose work in the field of visual servoing and robotics provided a foundation for my study. Their published papers and contributions were essential in advancing my understanding of the subject. I am particularly thankful to Indian Institute of Technology (IIT) Mandi for their support, which made this Internship possible for me. Finally, I would like to acknowledge the unwavering support of my family and friends. Their encouragement and belief in my abilities provided the motivation I needed to persevere through the challenges of this research. Thank you all for your support and contributions.

Dhruv Gupta

## Contents

<b>Chapter 1</b> .....	<b>9</b>
Introduction.....	9
<b>Chapter 2</b> .....	<b>10</b>
Introduction to Robot Kinematics .....	10
<b>Chapter 3</b> .....	<b>14</b>
Aruco Markers .....	14
3.1 Introduction.....	14
3.2 Structure and Characteristics .....	14
3.3 Dictionaries and Marker Sets .....	15
3.4 Applications of Aruco Marker.....	15
3.5 Aruco Marker Generation .....	17
3.6 Camera Calibration.....	19
3.7 Aruco Marker Detection .....	24
<b>CHAPTER 4</b> .....	<b>25</b>
ESP 8266 Wi-Fi module in IBVS .....	25
4.1 Introduction to ESP 8266.....	25
4.2 Role of ESP 8266 in IBVS.....	25
4.3 Applications of ESP8266 in IBVS Systems.....	25
4.4 Advantages and Limitations .....	26
<b>CHAPTER 5</b> .....	<b>28</b>
IBVS Simulation .....	28
5.1 Installation of ROS Noetic.....	29
5.2 Installation of Gazebo.....	31
5.3 Importing Husky Vehicle in Gazebo.....	32
<b>Bibliography</b> .....	<b>37</b>

# Chapter 1

## Introduction

In the realm of robotics and autonomous systems, the ability to accurately control the movement of unmanned aerial vehicles (UAVs) and robotic manipulators using visual feedback is a significant advancement. Image-Based Visual Servoing (IBVS) offers a robust approach to achieve this control by leveraging visual data to guide the robot's actions. During my internship, the primary focus was on implementing an IBVS system to control both a UAV and a manipulator based on visual feedback, complemented by the creation of an animation video and system simulations to validate the control strategies. The first objective was to set up a monocular camera and calibrate it for accurate visual data acquisition. This involved ensuring that the camera provided reliable and consistent image data, which is critical for the subsequent steps. Next, the development of algorithms to extract visual features from the camera feed was essential. These algorithms are responsible for identifying and tracking features within the images, providing the necessary information for the IBVS controller.

Designing the IBVS controller constituted the core of this project. The controller needed to process the visual feedback and adjust the movements of the UAV and manipulator accordingly. This required an in-depth understanding of both the kinematics and dynamics of the systems involved, as well as the ability to translate visual information into precise control commands. To visually demonstrate the operation of the IBVS system, an animation video was created. This animation served as a valuable tool to illustrate the functionality and effectiveness of the visual servoing system in a clear and engaging manner. Before real-world implementation, it was crucial to validate the control strategies through simulations. Using tools like Gazebo, the entire UAV system, including its kinematic, dynamic, and visual servoing models, was integrated into a simulation environment. Various scenarios were simulated to test the stability, accuracy, and robustness of the system, ensuring that it could perform reliably under different conditions. This report details the implementation process of the IBVS system, the creation of the animation video, and the results of the system simulations. It provides insights into the challenges encountered and the solutions developed, offering a comprehensive guide for future work in the field of visual servoing for UAVs and manipulators.

# Chapter 2

## Introduction to Robot Kinematics

Robot kinematics is a fundamental aspect of robotics that deals with the study of motion without considering the forces that cause it. It focuses on the geometric aspects of a robot's movement, specifically the relationship between the joint parameters and the position and orientation of the end-effector. Understanding robot kinematics is crucial for designing, controlling, and programming robots, as it enables the precise manipulation of objects and the execution of complex tasks.

### Types of Kinematics

Robot kinematics can be broadly categorized into two types: forward kinematics and inverse kinematics.

1. **Forward Kinematics:** Forward kinematics involves determining the position and orientation of the end-effector given the joint parameters (angles for rotational joints and displacements for prismatic joints). This process uses the known values of joint variables to compute the end-effector's pose using a series of mathematical transformations. The key challenge in forward kinematics is developing the kinematic equations that describe the robot's structure.
2. **Inverse Kinematics:** Inverse kinematics is the reverse process of forward kinematics. It involves finding the joint parameters that achieve a desired position and orientation of the end-effector. This is a more complex problem because multiple joint configurations can result in the same end-effector pose, and finding a unique or optimal solution requires sophisticated algorithms and techniques.

## 2.1 Mechanism and Mobility

**1. Degrees of Freedom (DOF):** Degrees of Freedom refer to the number of independent movements a mechanism or a robot can perform. In robotics, DOF is a critical concept as it defines the robot's capability to move and position its end-effector in 3D space. Each joint in a robot contributes to the overall DOF, and understanding how these combine is essential for designing and controlling robotic systems.

- **Translational DOF:** Movement along the x, y, and z axes.
- **Rotational DOF:** Rotation around the x, y, and z axes.

A robot's total DOF is the sum of all its joints' DOFs, determining how versatile and capable the robot is in performing complex tasks.

**2. Grübler's Law:** Grübler's Law is used to determine the mobility of a mechanism, which is the number of independent parameters that define the configuration of the system. It provides a formula to calculate the DOF of a planar or spatial mechanism based on the number of links and joints.

$$\text{dof} = m(N - 1 - J) + \sum_{i=1}^J f_i$$

- M is the mobility or the number of DOF,
- N is the number of links (including the ground link),
- J is the number of joints,
- H is the number of higher pairs (joints with surface contact).

## 2.2 End – Effector and Robotic Arms

End-effectors and manipulators are critical to the functionality of robotic systems. They enable robots to interact with their environment, performing tasks such as grasping, lifting, manipulating, and assembling objects. Understanding the design, operation, and control of grippers and manipulators is essential for developing versatile and capable robots.

**Grippers:** Grippers are specialized end-effectors designed to grasp and hold objects. They are crucial for applications requiring precise and secure handling of items.

### Types of Grippers:

1. **Mechanical Grippers:** Use mechanical fingers or jaws to grasp objects. They can be further classified based on their actuation mechanisms (e.g., parallel, angular, and adaptive grippers).
2. **Vacuum Grippers:** Utilize suction to pick up objects, commonly used for handling flat or smooth surfaces.
3. **Magnetic Grippers:** Employ magnetic fields to lift and move ferromagnetic materials.
4. **Adhesive Grippers:** Use adhesive materials or microstructures to adhere to objects, suitable for delicate and irregularly shaped items

**Manipulators:** Manipulators are robotic arms that provide the mobility and dexterity needed to position end-effectors accurately. They consist of a series of links and joints, forming a kinematic chain.

### Structure of Manipulators:

1. **Base:** The fixed or mobile part of the manipulator that supports the entire structure
2. **Joints** – Connect the links and provide DOF. Common joint types include revolute (rotational) and prismatic (linear).
3. **Links** – The rigid segments between joints.

## 2.3 Kinematic Models

Kinematic models are essential for describing the motion of robotic systems. They provide a mathematical framework to analyze the position, orientation, and movement of a robot's components. Key components of kinematic modeling include Denavit-Hartenberg (DH) parameters, homogeneous transformation matrices, and the Jacobian matrix.

### 2.3.1 Denavit – Hartenberg (DH) Parameter

The DH convention is a standardized method to systematically describe the geometry of robotic arms. It simplifies the process of defining the relative positions and orientations of links in a kinematic chain.

#### DH Parameter -

1. **Link Length ( $a_i$ ):** The distance between the  $Z_{i-1}$  and  $Z_i$  axes along the  $X_i$  axis.
2. **Link Twist ( $\alpha_i$ ):** The angle between the  $Z_{i-1}$  and  $Z_i$  axes around the  $X_i$  axis.
3. **Link Offset ( $d_i$ ):** The distance between the  $X_{i-1}$  and  $X_i$  axes along the  $Z_{i-1}$  axis.
4. **Joint Angle ( $\theta_i$ ):** The angle between the  $X_{i-1}$  and  $X_i$  axes around the  $Z_{i-1}$  axis.

#### DH Transformation Matrix -

$$A_i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 2.3.2 Homogenous Transformation Matrix

Homogeneous transformation matrices combine rotation and translation into a single 4x4 matrix, facilitating the transformation of coordinates between different frames.

Transformation Matrix (T) :-

$$T = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix}$$

R = 3X3 Rotation Matrix

D = 3X1 Translation Matrix



### 2.3.3 Jacobian Matrix

The Jacobian matrix relates joint velocities to the end-effector's linear and angular velocities. It is a crucial tool in robot kinematics for analyzing the motion capabilities and performing control tasks.

The Jacobian matrix  $J$  maps joint velocities  $\dot{q}$  to end-effector velocities  $v$  and  $\omega$ .

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = J\dot{q}$$

$V$  = linear velocity

$\Omega$  = angular velocity

For a robotic arm, the Jacobian matrix can be derived by differentiating the forward kinematics equations with respect to time

- For Revolute Joint

$$J_i = \begin{bmatrix} \frac{\partial p}{\partial \theta_i} \\ \frac{\partial o}{\partial \theta_i} \end{bmatrix}$$

- For Prismatic Joint

$$J_i = \begin{bmatrix} \frac{\partial p}{\partial d_i} \\ \frac{\partial o}{\partial d_i} \end{bmatrix}$$

# Chapter 3

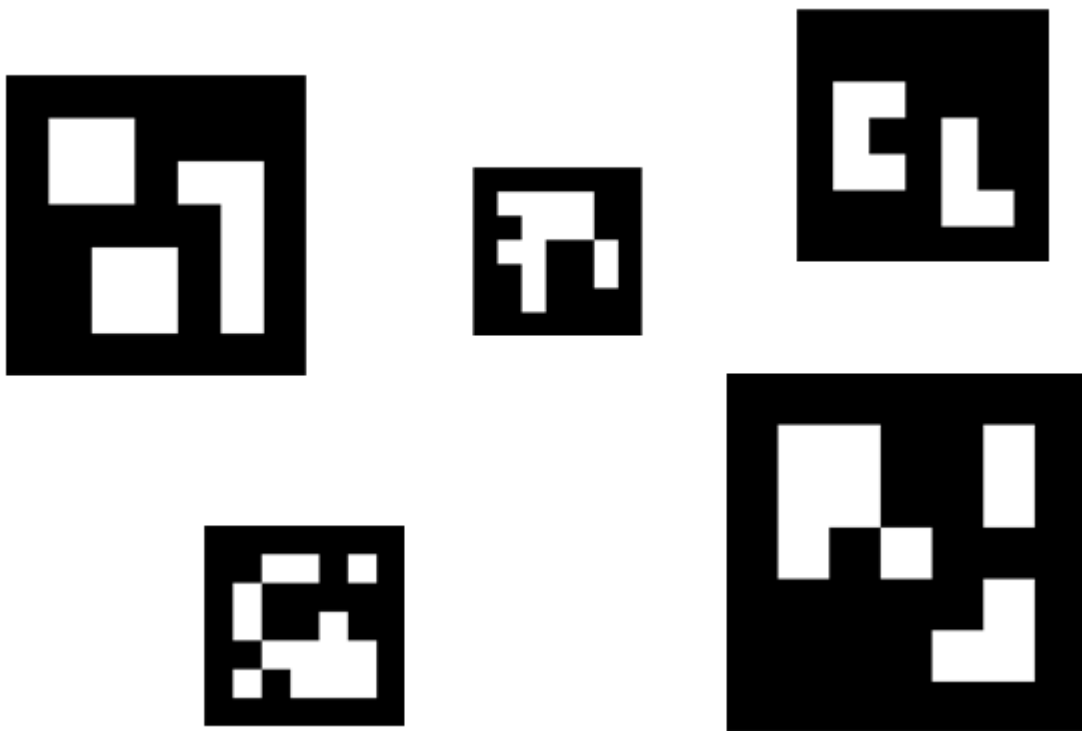
## Aruco Markers

### 3.1 Introduction

Aruco markers are a type of fiducial marker used in computer vision applications for camera pose estimation, object tracking, and augmented reality. They are designed to be easily detectable and identifiable by computer vision algorithms, making them ideal for tasks that require precise and reliable tracking. Aruco markers are particularly popular due to their robustness, ease of use, and the comprehensive support provided by OpenCV's Aruco module.

### 3.2 Structure and Characteristics

Aruco markers are typically square-shaped with a distinctive black border and a unique internal binary pattern. The black border aids in the initial detection of the marker by distinguishing it from the background, while the internal pattern encodes a unique identifier for each marker. This combination of features allows for both reliable detection and identification, even under varying lighting conditions and partial occlusions.



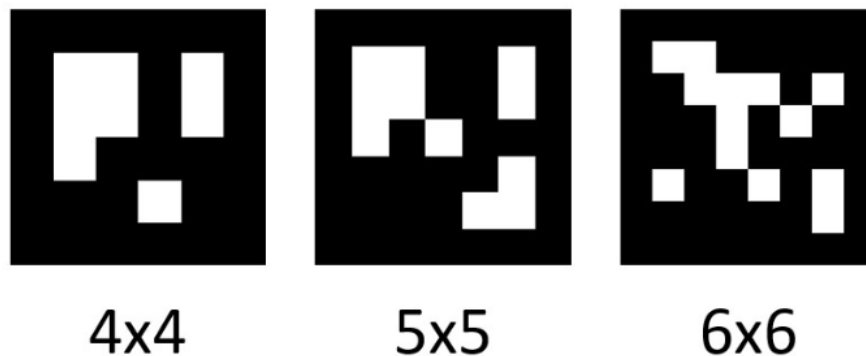
Above figure represent different types of Aruco Marker.

### 3.3 Dictionaries and Marker Sets

Aruco markers are organized into dictionaries, each containing a specific number of unique markers. These dictionaries vary in the size of the internal binary code and the number of markers they include.

For example -

- DICT 4x4 50: Contains 50 markers with a 4x4 internal binary grid.
- DICT 5x5 100: Contains 100 markers with a 5x5 internal binary grid.
- DICT 6x6 250: Contains 250 markers with a 6x6 internal binary grid.



### 3.4 Applications of Aruco Marker

Aruco markers have a wide range of applications due to their ability to be easily detected and uniquely identified in images. Their robust design and ease of use make them ideal for various fields, including augmented reality, robotics, camera calibration, and more. Below are some key applications of Aruco markers:

#### 3.4.1 Augmented Reality (AR)

##### Interactive Experiences:

Markers can be used to trigger interactive content. For example, scanning a marker with a smartphone can display 3D models, animations, or other multimedia content, enhancing user engagement and interaction.

#### 3.4.2 Robotics

##### Navigation and Localization:

Robots use Aruco markers for navigation and localization within an environment. By detecting markers placed in known locations, robots can determine their own position and orientation, enabling precise movements and autonomous navigation.

### **3.4.3 Camera Calibration**

#### **Intrinsic and Extrinsic Parameter Estimation:**

Aruco markers are used to calibrate cameras by determining intrinsic parameters (such as focal length and optical center) and extrinsic parameters (such as position and orientation). Accurate calibration is essential for applications that require precise 3D measurements and reconstructions.

#### **Lens Distortion Correction:**

Markers help in correcting lens distortion, ensuring that images captured by the camera are geometrically accurate. This is crucial for applications in photogrammetry, 3D scanning, and other vision-based measurements.

### **3.4.4 3D Reconstruction**

#### **Multi-View Geometry:**

In 3D reconstruction, Aruco markers are used to align multiple images taken from different viewpoints. The markers provide reference points that help in accurately stitching the images together to create a 3D model.

### **3.4.5 Human-Computer Interaction (HCI)**

#### **Gesture Recognition:**

Aruco markers can be placed on gloves or wearable devices to track hand and finger movements, enabling gesture-based control interfaces for computers and other digital devices.

#### **Virtual Reality (VR):**

Markers can be used to track the position and orientation of VR headsets and controllers, ensuring precise and responsive interactions in virtual environments.

### **3.4.6 Quality Control and Inspection**

#### **Industrial Automation:**

In manufacturing, Aruco markers are used for quality control and inspection processes. They can be placed on products or components to facilitate automated recognition, sorting, and verification tasks.

#### **Assembly Line Monitoring:**

Markers help in monitoring assembly lines by providing reference points that can be used to verify the correct assembly of parts and components, ensuring product quality and consistency.

## 3.5 Aruco Marker Generation

The process of generating Aruco markers involves creating the binary pattern for each marker and ensuring that it is unique within the chosen dictionary. OpenCV's **Aruco** module provides a convenient way to generate these markers programmatically. In this code we can generate single or bulk markers .

### CODE -

```
import cv2
def generate_single_marker(aruco_dict):
    marker_size = int(input("Enter the marker size: "))
    marker_id = int(input("Enter the marker ID: "))

    marker_img = cv2.aruco.generateImageMarker(aruco_dict, marker_id,
        marker_size)

    cv2.imwrite("marker_{}.png".format(marker_id), marker_img)

    marker_img = cv2.imread("marker_{}.png".format(marker_id))

    cv2.imshow("Marker", marker_img)

    print("Dimensions:", marker_img.shape)

    cv2.waitKey(0)
def generate_bulk_markers(aruco_dict):
    marker_size = int(input("Enter the marker size: "))
    num_markers = int(input("Enter the number of markers to generate: "))
    marker_imgs = []

    for marker_id in range(num_markers):
        marker_img = cv2.aruco.generateImageMarker(aruco_dict, marker_id,
            marker_size)

        cv2.imwrite("marker_{}.png".format(marker_id), marker_img)
        marker_imgs.append(cv2.imread("marker_{}.png".format(marker_id)))

    for marker_img in marker_imgs:
        cv2.imshow("Marker", marker_img)
        print("Dimensions:", marker_img.shape)
        cv2.waitKey(0)
def main():
    aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)

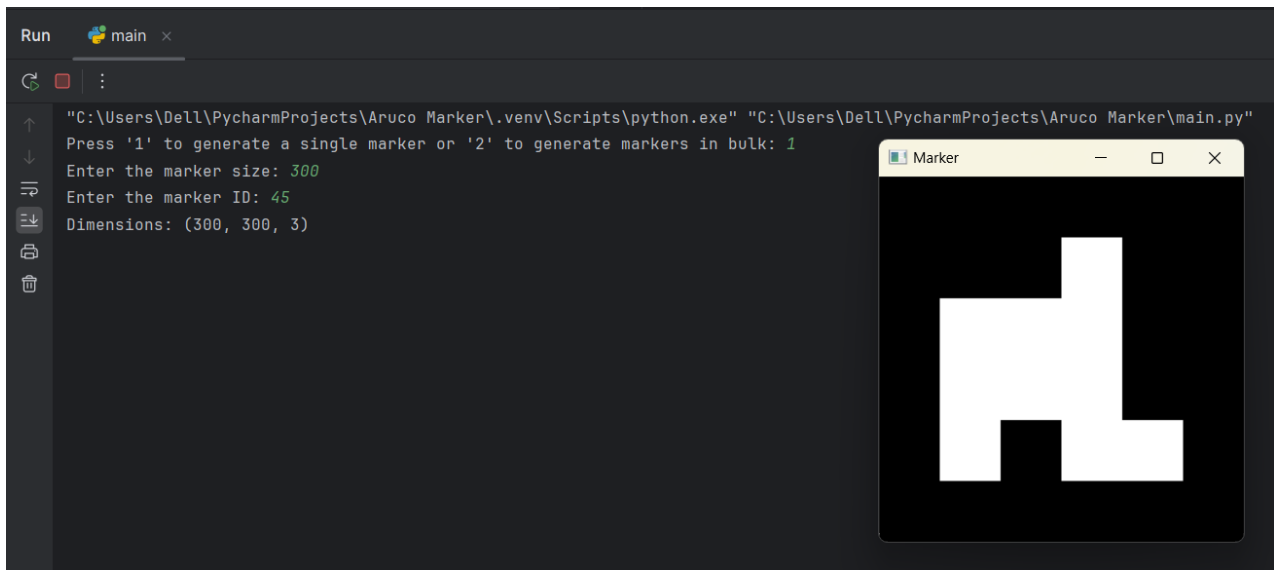
    user_input = input("Press '1' to generate a single marker or "
        "'2' to generate markers in bulk: ")

    if user_input == "1":
        generate_single_marker(aruco_dict)
    elif user_input == "2":
        generate_bulk_markers(aruco_dict)
    else:
        print("Invalid input. Please try again.")

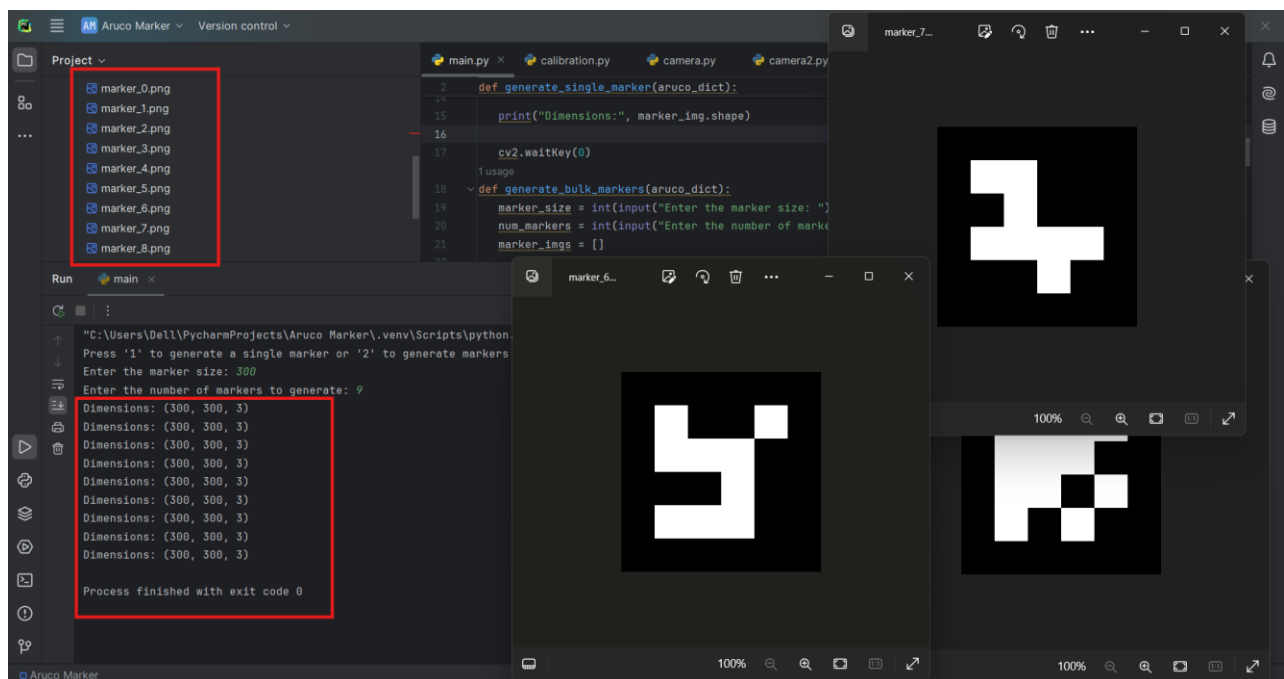
if __name__ == "__main__":
    main()
```

### OUTPUT -

## A. Single marker -



## B. Bulk marker -



## Explanation -

This Python script is designed to generate and display Aruco markers using the OpenCV library, which are widely used in computer vision applications for tasks such as camera calibration, pose estimation, and augmented reality. The script provides functionality to generate either a single marker or multiple markers in bulk, catering to various application needs.

The script begins by importing the necessary OpenCV library, which provides the functions required for generating and detecting Aruco markers. The first function, `generate_single_marker(aruco_dict)`, allows the user to generate a single Aruco marker. The function prompts the user to input the desired marker size and ID. It then uses the `cv2.aruco.generateImageMarker()` function to generate the marker image based on the specified dictionary, ID, and size. The generated marker image is saved to a file, read back, and displayed in a window using OpenCV's `cv2.imshow()` function.

The second function, `generate_bulk_markers(aruco_dict)`, is designed to generate multiple ArUco markers in bulk. The user is prompted to input the marker size and the number of markers to generate. The function iterates over the range of marker IDs, generating and saving each marker image to a file. These images are then read back and stored in a list. The function subsequently displays each marker image in a window and prints their dimensions to the console, waiting for a key press before displaying the next marker.

The `main()` function acts as the entry point of the script. It begins by selecting the predefined dictionary for generating the markers, specifically a 4x4 dictionary with 50 unique markers. The user is then prompted to choose between generating a single marker or multiple markers in bulk. Based on the user's input, the appropriate function (`generate_single_marker` or `generate_bulk_markers`) is called.

Finally, the script includes a standard Python entry point check (`if __name__ == "__main__":`) to ensure that the `main()` function is called when the script is run directly. This user-friendly script provides a straightforward method to generate and display Aruco markers, making it suitable for various computer vision projects. The generated markers can be easily saved as image files and used for tasks that require precise detection and identification.

## 3.6 Camera Calibration

Camera calibration is the process of estimating the parameters of a camera to improve the accuracy of images taken with it. This involves determining the camera's intrinsic and extrinsic parameters. It contains 2 types of parameter Intrinsic parameters and Extrinsic parameters.

### 3.6.1 Intrinsic Parameters

These are the internal characteristics of the camera:

- **Focal length (f):** Determines the field of view.
- **Principal point (cx, cy):** The optical center of the image.
- **Skew coefficient ( $\alpha$ ):** Describes the angle between the x and y pixel axes.
- **Distortion coefficients:** Correct for lens distortion (e.g., radial and tangential distortions)

### 3.6.2 Extrinsic Parameters

These define the camera's position and orientation in the world coordinate system:

- **Rotation matrix ( $R$ ):** Describes the camera's orientation.
- **Translation vector ( $t$ ):** Describes the camera's position.

### 3.6.3 Why Camera Calibration

- **Correcting Lens Distortion:** Lenses, especially wide-angle ones, can cause distortion, making straight lines appear curved. Calibration helps correct this.
- **Accurate 3D Measurements:** For applications like 3D reconstruction, augmented reality, and robotics, knowing the camera parameters is essential for accurate spatial measurements.
- **Image Rectification:** Calibration can correct images for projects involving stitching multiple images together.

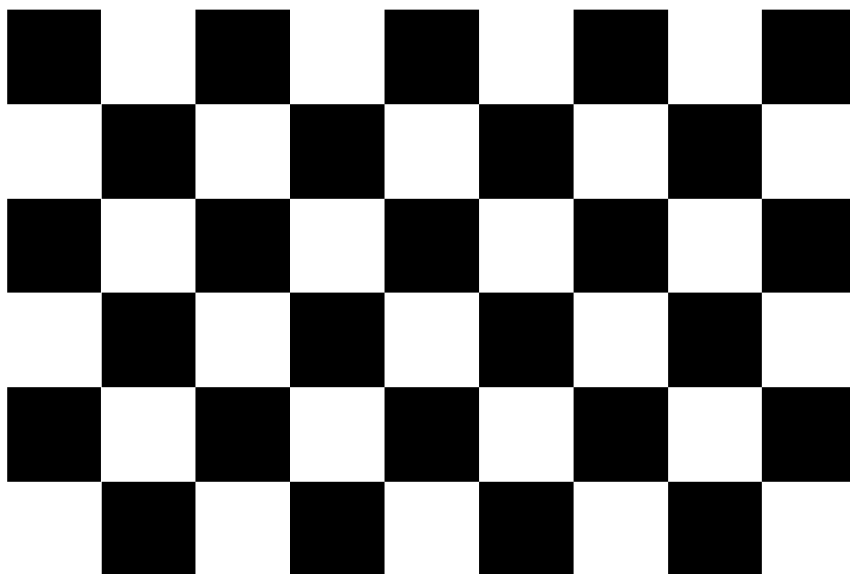
### 3.6.4 Calibration Process

Typically, a known pattern (like a chessboard of size 9\*6) is captured in various orientations and positions. Using algorithms, the observed image points are compared to the known pattern points to estimate the camera parameters.

In OpenCV, for instance, the process involves:

1. Capturing multiple images of a calibration pattern.
2. Detecting feature points in the images.
3. Solving equations that relate the feature points to the known pattern points.
4. Refining the camera parameters to minimize the error between observed and projected points.

#### Chess Board of 9\*6 for Calibration





## Code –

```
import cv2 as cv
import os

CHESS_BOARD_DIM = (9, 6)

n = 0

image_dir_path = "images"

CHECK_DIR = os.path.isdir(image_dir_path)

if not CHECK_DIR:
    os.makedirs(image_dir_path)
    print(f'"{image_dir_path}" Directory is created')
else:
    print(f'"{image_dir_path}" Directory already Exists.')

criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

def detect_checker_board(image, grayImage, criteria, boardDimension):
    ret, corners = cv.findChessboardCorners(grayImage, boardDimension)
    if ret == True:
        corners1 = cv.cornerSubPix(grayImage, corners, (3, 3), (-1, -1),
criteria)
        image = cv.drawChessboardCorners(image, boardDimension, corners1, ret)

    return image, ret

cap = cv.VideoCapture(0)

while True:
    _, frame = cap.read()
    copyFrame = frame.copy()
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    image, board_detected = detect_checker_board(frame, gray, criteria,
CHESS_BOARD_DIM)

    cv.putText(
        frame,
        f"saved_img : {n}",
        (30, 40),
        cv.FONT_HERSHEY_PLAIN,
        1.4,
        (0, 255, 0),
        2,
        cv.LINE_AA,
    )
```

```

cv.imshow("frame", frame)
cv.imshow("copyFrame", copyFrame)

key = cv.waitKey(1)

if key == ord("q"):
    break
if key == ord("s") and board_detected == True:

    cv.imwrite(f"{image_dir_path}/image{n}.png", copyFrame)

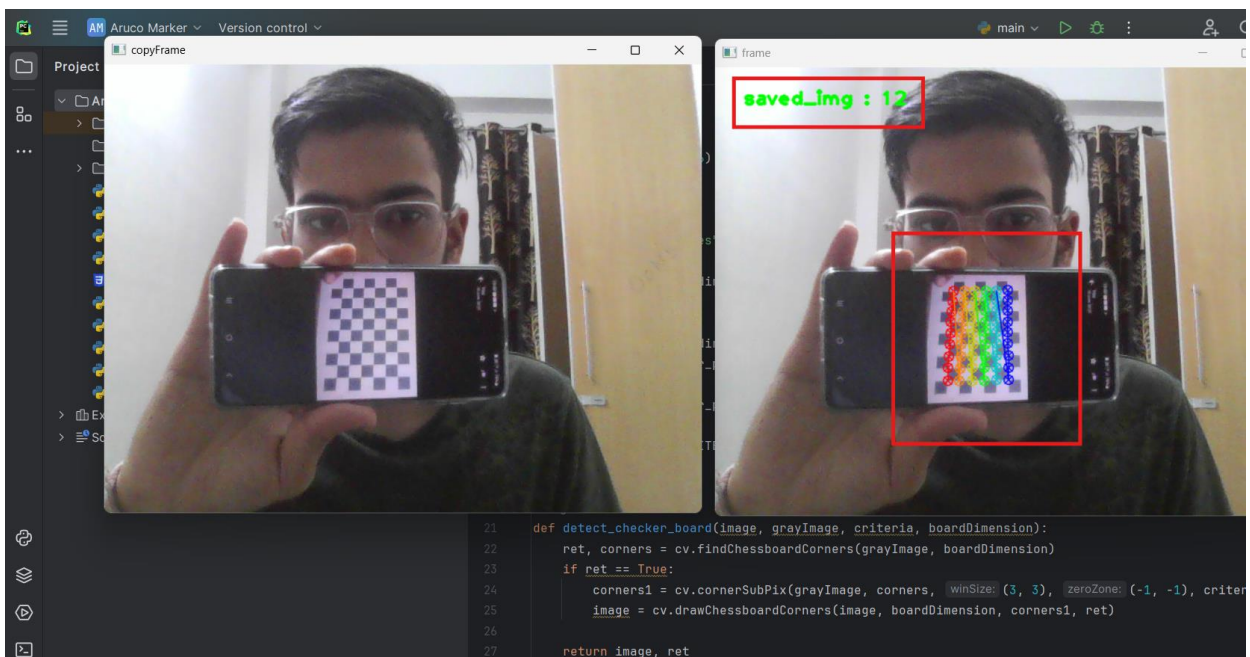
    print(f"saved image number {n}")
    n += 1
cap.release()
cv.destroyAllWindows()

print("Total saved Images:", n)

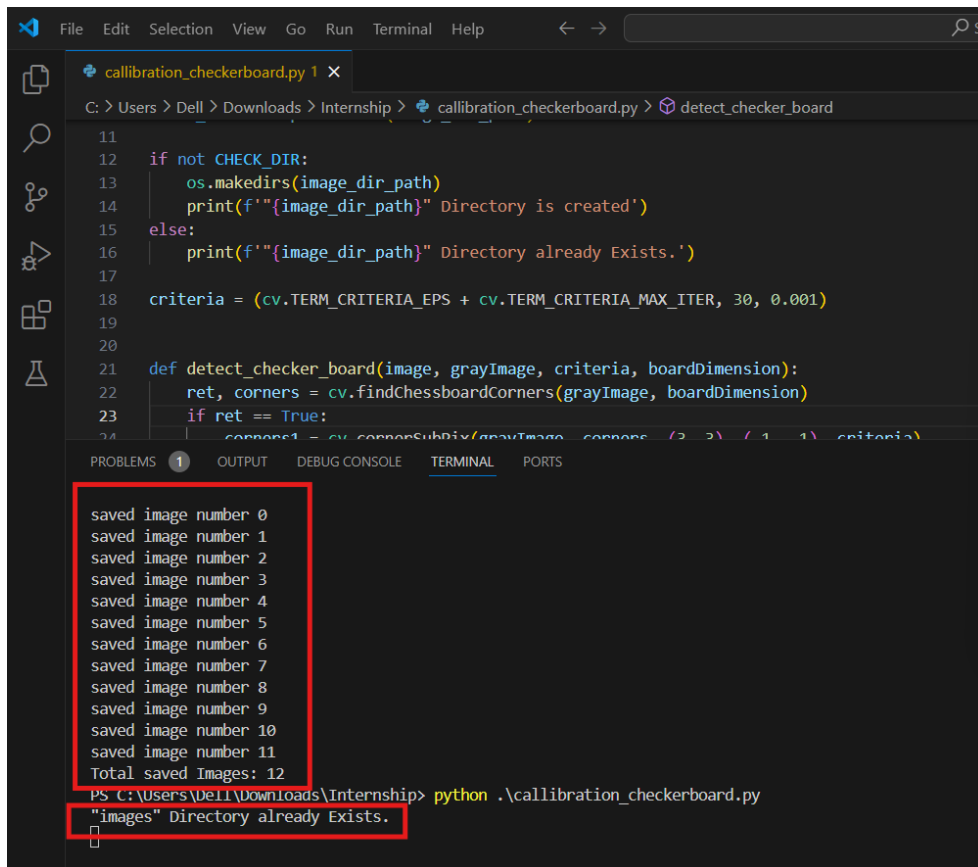
```

### Output -

a. Here Red Box Indicates Calibration of camera using chessboard (9\*6) and another red box shows total number of saved images .



b. In the terminal output number of total images saved in the directory called as images.



The screenshot shows a code editor with a file named `callibration_checkerboard.py`. The code includes a directory check and a function `detect_checker_board` that uses OpenCV's `findChessboardCorners` and `cornerSubPix`. The terminal output, highlighted with a red box, shows the script successfully saving 12 images and reporting the total count.

```
11
12 if not CHECK_DIR:
13     os.makedirs(image_dir_path)
14     print(f"{image_dir_path}" Directory is created')
15 else:
16     print(f"{image_dir_path}" Directory already Exists.')
17
18 criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
19
20
21 def detect_checker_board(image, grayImage, criteria, boardDimension):
22     ret, corners = cv.findChessboardCorners(grayImage, boardDimension)
23     if ret == True:
24         corners1 = cv.cornerSubPix(grayImage, corners, (3, 3), (1, 1), criteria)
```

```
saved image number 0
saved image number 1
saved image number 2
saved image number 3
saved image number 4
saved image number 5
saved image number 6
saved image number 7
saved image number 8
saved image number 9
saved image number 10
saved image number 11
Total saved Images: 12
PS C:\Users\Dell\Downloads\Internship> python .\callibration_checkerboard.py
"images" Directory already Exists.
```

## Explanation –

This code is designed to capture images from a webcam for the purpose of camera calibration using a chessboard pattern. It begins by defining the dimensions of the chessboard, in this case, a 9x6 grid of squares. The script checks if a directory named "images" exists in the current working directory; if not, it creates the directory to store captured images.

The main part of the code is a loop that continuously captures frames from the webcam. For each frame, it makes a copy and converts the original frame to grayscale. The function `detect_checker_board` is then called, which uses OpenCV's `findChessboardCorners` to detect the corners of the chessboard pattern in the grayscale image. If the pattern is found, the corners are refined for better accuracy using `cornerSubPix` and then drawn on the original frame.

The script displays two windows: one showing the original frame with detected corners drawn, and another showing the original unprocessed frame. It also overlays text on the frame displaying the number of saved images.

The loop checks for key presses: pressing 'q' quits the loop, and pressing 's' saves the current frame to the "images" directory if the chessboard pattern is detected. Each saved image is named sequentially (e.g., `image0.png`, `image1.png`, etc.). Finally, it releases the webcam and closes all OpenCV windows, printing the total number of saved images. This process helps in collecting multiple images of the chessboard from different angles and positions, essential for accurate camera calibration. After this our camera is ready to perform Pose estimation of aruco marker.

## 3.7 Aruco Marker Detection

Code –

```
import cv2
from cv2 import aruco
def main():

    aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_5X5_250)
    param_markers = aruco.DetectorParameters()

    cap = cv2.VideoCapture(0)

    while True:

        ret, frame = cap.read()

        if not ret:
            print("Failed to grab frame")
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        marker_corners, marker_IDs, reject = aruco.detectMarkers(gray,
aruco_dict, parameters=param_markers)

        if marker_IDs is not None:
            frame = aruco.drawDetectedMarkers(frame, marker_corners, marker_IDs)

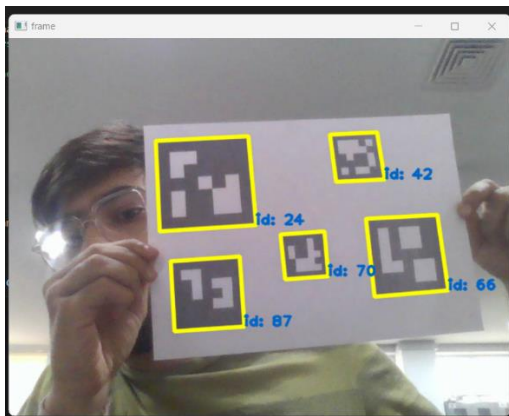
        cv2.imshow('Aruco Marker Detection', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

Output –



# CHAPTER 4

## ESP 8266 Wi-Fi module in IBVS

### 4.1 Introduction to ESP 8266

The ESP8266 is a low-cost, high-performance Wi-Fi module capable of providing full internet connectivity to embedded devices. It comes with a built-in TCP/IP stack, making it ideal for IoT applications. The module can be programmed using the Arduino IDE, making it accessible to hobbyists and professionals alike.

#### Key Features –

- **Wi-Fi Connectivity:** Supports 802.11 b/g/n protocols.
- **Microcontroller Integration:** Can be integrated with various microcontrollers like Arduino and ESP32.
- **Low Power Consumption:** Suitable for battery-powered applications.
- **Compact Size:** Easy to incorporate into small devices.
- **Programming Flexibility:** Can be programmed using Lua, Arduino IDE, or native SDK..

### 4.2 Role of ESP 8266 in IBVS

#### Importance of Wireless Communication in IBVS

Image-Based Visual Servoing (IBVS) systems rely on visual data to control and adjust the position of robots or other devices in real time. Traditionally, these systems may use wired connections to transmit data between the camera, processing unit, and actuators. However, wireless communication offers several advantages:

- **Flexibility and Mobility:** Wireless communication eliminates the need for physical connections, allowing greater freedom of movement for mobile robots, drones, and other devices.
- **Remote Operation:** Operators can control and monitor the system from a remote location, enhancing safety and convenience, especially in hazardous or hard-to-reach environments.
- **Ease of Deployment:** Without the constraints of wires, systems can be deployed and reconfigured more easily, which is particularly useful in dynamic environments or for temporary setups.

### 4.3 Applications of ESP8266 in IBVS Systems

#### 4.3.1 Real-Time Data Transmission:

- The ESP8266 can transmit visual data from a camera to a processing unit over a Wi-Fi network. This real-time data transmission is essential for responsive IBVS systems.

- Example: A drone equipped with a camera uses the ESP8266 to send live video feeds to a ground control station, where the images are processed to adjust the drone's flight path.

#### 4.3.2 Remote Control and Monitoring:

- The ESP8266 allows for remote control of robotic systems via Wi-Fi. Operators can send control commands from a remote location based on the visual feedback received.
- Example: In an industrial setting, a robotic arm can be monitored and controlled from a central control room, reducing the need for human presence on the factory floor.

#### 4.3.3 Integration with Cloud Services:

- The ESP8266 can interface with cloud services to store and process data, enabling advanced analytics and machine learning applications.
- Example: Visual data from a robot can be uploaded to the cloud, where sophisticated algorithms analyze the data and send optimized control commands back to the robot.

#### 4.3.4 Edge Processing:

- While the ESP8266 itself has limited processing power, it can be used in conjunction with more powerful edge computing devices to preprocess visual data before transmitting it.
- Example: A camera module connected to an ESP8266 preprocesses images (e.g., compressing or filtering) before sending them to a server, reducing bandwidth requirements and improving transmission speed.

#### 4.3.5 Networked Robotic Systems:

- Multiple robots equipped with ESP8266 modules can communicate and coordinate their actions over a shared Wi-Fi network.
- Example: In a warehouse, a fleet of autonomous vehicles can communicate with each other to optimize their paths and avoid collisions, all facilitated by the ESP8266.

### 4.4 Advantages and Limitations

#### 4.4.1 Advantages:

- **Cost-Effective:** The ESP8266 is a low-cost module, making it an affordable option for adding Wi-Fi capabilities to IBVS systems.
- **Ease of Use:** It is compatible with various programming environments, including Arduino, making it accessible for both hobbyists and professionals.
- **Robust Connectivity:** It supports multiple Wi-Fi standards and provides reliable connectivity for real-time data transmission.

#### 4.4.2 Limitations:

- **Processing Power:** The ESP8266 has limited processing capabilities, which may not be sufficient for complex image processing tasks. It is typically used to transmit data to more powerful processors.
- **Range and Latency:** The performance of Wi-Fi networks can be affected by range and latency issues, which may impact the responsiveness of IBVS systems in certain environments.

- **Power Consumption:** While suitable for many applications, the power consumption of the ESP8266 may be a concern for battery-operated systems, requiring careful power management.

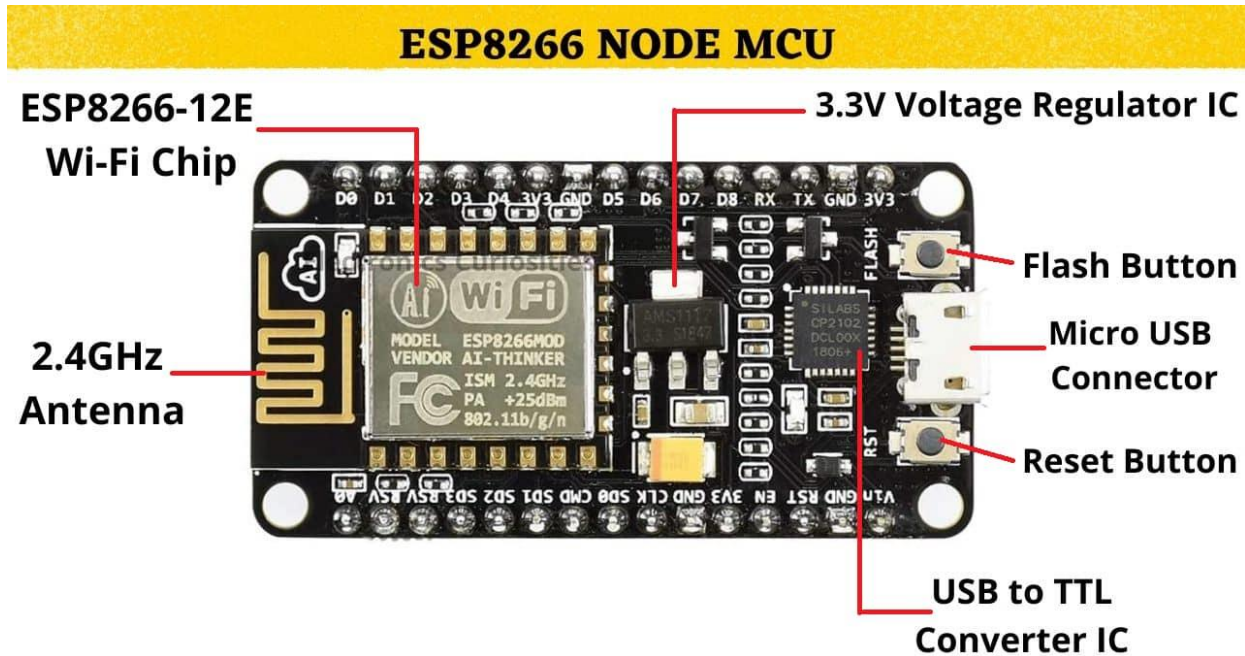


Figure shows ESP8266 Wifi Module

# CHAPTER 5

## IBVS Simulation

For IBVS Stimulation we need to install Ubuntu 20.04, ROS Noetic and Gazebo, firstly we need to install the Oracle's Virtual box 7.0.

### Ubuntu 20.04

Ubuntu 20.04, codenamed "Focal Fossa," is a long-term support (LTS) release of the Ubuntu operating system. Released in April 2020, it is designed to provide a stable and secure platform for users and developers, with updates and support until April 2025. Ubuntu 20.04 is built on the Linux kernel and uses the GNOME desktop environment by default.

- **LTS Release:** Offers five years of support and updates, ensuring stability for long-term projects.
- **Kernel and Performance:** Comes with Linux Kernel 5.4, which includes performance improvements, better hardware support, and enhanced security features.

### Ros Noetic

ROS (Robot Operating System) Noetic Ninjemys is the final LTS release of the ROS 1 distribution. Released in May 2020, it is targeted for Ubuntu 20.04 and provides a mature, stable platform for robotics development. ROS Noetic is designed to be compatible with Python 3, aligning with modern software development practices.

- **Python 3 Support:** ROS Noetic is fully compatible with Python 3, ensuring better performance, security, and future-proofing of ROS applications.
- **Extensive Libraries and Tools:** Includes a wide range of libraries and tools for robot simulation, visualization, navigation, manipulation, and more.

### Gazebo

Gazebo is an open-source robotics simulator that provides an accurate and efficient way to test and develop robotic systems in a virtual environment. It integrates seamlessly with ROS, allowing developers to simulate complex robotic scenarios without the need for physical hardware.

- **Physics Simulation:** Offers robust physics engines (such as ODE, Bullet, and DART) for realistic simulation of robot dynamics, collisions, and environmental interactions.
- **Integration with ROS:** Seamlessly integrates with ROS, allowing the use of ROS topics, services, and messages for controlling and monitoring simulated robots.



## 5.1 Installation of ROS Noetic

After installation of ubuntu and virtual box now we have to install ROS Noetic , following Commands are used to install it on running it on terminal.

### 5.1.1 Setup your sources.list

```
dhruv@dhruv-VirtualBox:~$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
[sudo] password for dhruv:
```

### 5.1.2 Setup up your keys

```
dhruv@dhruv-VirtualBox:~$ sudo apt install curl # if you haven't already installed curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
curl is already the newest version (7.68.0-1ubuntu2.22).
0 upgraded, 0 newly installed, 0 to remove and 60 not upgraded.
dhruv@dhruv-VirtualBox:~$
```

```
dhruv@dhruv-VirtualBox:~$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
OK
```

### 5.1.3 Installation

```
dhruv@dhruv-VirtualBox:~$ sudo apt update
Hit:1 http://in.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://packages.osrfoundation.org/gazebo/ubuntu-stable focal InRelease
Hit:3 http://packages.ros.org/ros/ubuntu focal InRelease
Get:4 http://in.archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
Hit:6 http://in.archive.ubuntu.com/ubuntu focal-backports InRelease
Get:7 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [3,392 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main i386 Packages [769 kB]
Get:9 http://in.archive.ubuntu.com/ubuntu focal-updates/main i386 Packages [993 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3,020 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [452 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [2,924 kB]
Get:13 http://in.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [531 kB]
Get:14 http://security.ubuntu.com/ubuntu focal-security/restricted Translation-en [409 kB]
Get:15 http://security.ubuntu.com/ubuntu focal-security/universe i386 Packages [665 kB]
Get:16 http://in.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3,040 kB]
Get:17 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [980 kB]
Get:18 http://security.ubuntu.com/ubuntu focal-security/universe Translation-en [207 kB]
Get:19 http://in.archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [425 kB]
Get:20 http://in.archive.ubuntu.com/ubuntu focal-updates/universe i386 Packages [791 kB]
Get:21 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1,199 kB]
Get:22 http://in.archive.ubuntu.com/ubuntu focal-updates/universe Translation-en [289 kB]
Fetched 20.3 MB in 12s (1,738 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
73 packages can be upgraded. Run 'apt list --upgradable' to see them.
dhruv@dhruv-VirtualBox:~$
```

### 5.1.4 downloading Desktop full ros noetic

```
dhruv@dhruv-VirtualBox:~$ sudo apt install ros-noetic-desktop-full
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-desktop-full is already the newest version (1.5.0-1focal.20240522.184304).
0 upgraded, 0 newly installed, 0 to remove and 73 not upgraded.
dhruv@dhruv-VirtualBox:~$
```

```
dhruv@dhruv-VirtualBox:~$ sudo apt install ros-noetic-PACKAGE
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

## 5.1.5 Environment setup

```
dhruv@dhruv-VirtualBox:~$ source /opt/ros/noetic/setup.bash
```

```
dhruv@dhruv-VirtualBox:~$ sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.8ubuntu1.1).
The following additional packages will be installed:
  brz bzip2 libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib
  libserf-1-1 libsvn1 libutf8proc2 mercurial mercurial-common python2
  python2-minimal python2.7 python2.7-minimal python3-breezy
  python3-configobj python3-deprecated python3-dulwich python3-fastimport
  python3-github python3-gitlab python3-gpg python3-rosdistro
  python3-vcstools python3-wrapt subversion
Suggested packages:
  brz-doc python3-breezy.tests kdiff3 | kdiff3-qt | kompare | meld | tkcvs
  | mgdiff qct python-mysqldb python-openssl python-pygments python2-doc
  python-tk python2.7-doc python3-breezy-dbg python3-kerberos
  python-configobj-doc python-gitlab-doc db5.3-util libapache2-mod-svn
  subversion-tools
The following NEW packages will be installed:
  brz bzip2 libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib
  libserf-1-1 libsvn1 libutf8proc2 mercurial mercurial-common python2
  python2-minimal python2.7 python2.7-minimal python3-breezy
  python3-configobj python3-deprecated python3-dulwich python3-fastimport
  python3-github python3-gitlab python3-gpg python3-rosdep python3-rosdistro
  python3-rosinstall python3-rosinstall-generator python3-vcstools
  python3-wrapt python3-wstool subversion
0 upgraded, 30 newly installed, 0 to remove and 73 not upgraded.
Need to get 11.4 MB of archives.
After this operation, 56.2 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://packages.ros.org/ros/ubuntu focal/main amd64 python3-rosdep all 0.24.0-1 [3,536 B]
Get:2 http://packages.ros.org/ros/ubuntu focal/main amd64 python3-rosdistro all 0.9.1-100 [6,376 B]
Get:3 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 libpython2.7-minimal amd64 2.7.18-1-20.04.4 [335 kB]
Get:4 http://packages.ros.org/ros/ubuntu focal/main amd64 python3-rosinstall-generator all 0.1.23-1 [11.8 kB]
Get:5 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 python2.7-minimal amd64 2.7.18-1-20.04.4 [1,280 kB]
Get:6 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 python2-minimal amd64 2.7.17-2ubuntu4 [27.5 kB]
Get:7 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 libpython2.7-stdlib amd64 2.7.18-1-20.04.4 [1,887 kB]
Get:8 http://in.archive.ubuntu.com/ubuntu focal-updates/universe amd64 python2.7 amd64 2.7.18-1-20.04.4 [248 kB]
Get:9 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 libpython2-stdlib amd64 2.7.17-2ubuntu4 [7,072 B]
```

## 5.1.6 Dependencies for building packages

```
dhruv@dhruv-VirtualBox:~$ sudo rosdep init
Wrote /etc/ros/rosdep/sources.list.d/20-default.list
Recommended: please run

    rosdep update

dhruv@dhruv-VirtualBox:~$ rosdep update
reading in sources list data from /etc/ros/rosdep/sources.list.d
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/osx-homebrew.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/base.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/python.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/ruby.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/releases/fuerte.yaml
Query rosdistro index https://raw.githubusercontent.com/ros/rosdistro/master/index-v4.yaml
Skip end-of-life distro "ardent"
Skip end-of-life distro "bouncy"
Skip end-of-life distro "crystal"
Skip end-of-life distro "dashing"
Skip end-of-life distro "eloquent"
Skip end-of-life distro "foxy"
Skip end-of-life distro "galactic"
Skip end-of-life distro "groovy"
Add distro "humble"
Skip end-of-life distro "hydro"
Skip end-of-life distro "indigo"
Add distro "iron"
Skip end-of-life distro "jade"
Add distro "jazzy"
Skip end-of-life distro "kinetic"
Skip end-of-life distro "lunar"
Skip end-of-life distro "melodic"
Add distro "noetic"
Add distro "rolling"
updated cache in /home/dhruv/.ros/rosdep/sources.cache
```

## 5.2 Installation of Gazebo

```
dhruv@dhruv-VirtualBox:~$ sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable $(lsb_release -sc) main" > /etc/apt/sources.list.d/gazebo-stable.list'
[sudo] password for dhruv:
dhruv@dhruv-VirtualBox:~$
```

```
dhruv@dhruv-VirtualBox:~$ wget https://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
--2024-06-29 15:18:20-- https://packages.osrfoundation.org/gazebo.key
Resolving packages.osrfoundation.org (packages.osrfoundation.org)... 52.52.171.73
Connecting to packages.osrfoundation.org (packages.osrfoundation.org)|52.52.171.73|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1755 (1.7K) [application/octet-stream]
Saving to: 'STDOUT'

-          100%[=====] 1.71K --.-KB/s in 0s

2024-06-29 15:18:22 (151 MB/s) - written to stdout [1755/1755]

OK
```

```
dhruv@dhruv-VirtualBox:~$ sudo apt-get update
Hit:1 http://packages.osrfoundation.org/gazebo/ubuntu-stable focal InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:4 http://packages.ros.org/ros/ubuntu focal InRelease
Get:5 http://in.archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
Hit:6 http://in.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 128 kB in 3s (45.2 kB/s)
```

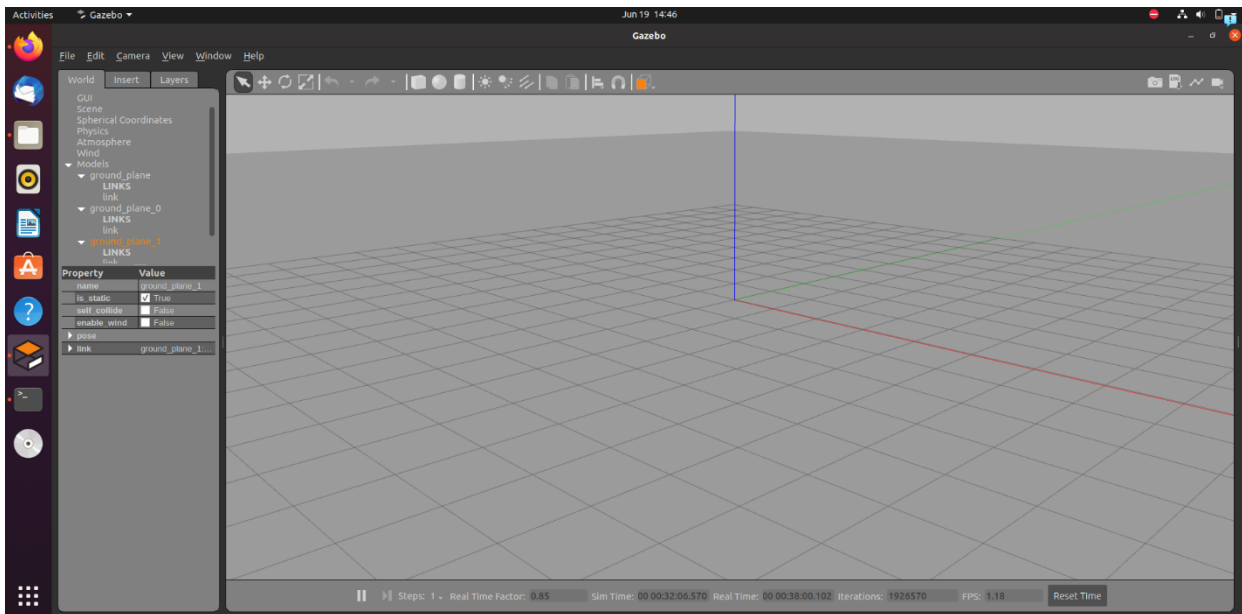
```
dhruv@dhruv-VirtualBox:~$ sudo apt-get install gazebo11
Reading package lists... Done
Building dependency tree
Reading state information... Done
gazebo11 is already the newest version (11.14.0-1~focal).
0 upgraded, 0 newly installed, 0 to remove and 73 not upgraded.
dhruv@dhruv-VirtualBox:~$
```

```
dhruv@dhruv-VirtualBox:~$ sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noetic-gazebo-ros-control
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-ros-control is already the newest version (2.9.2-1focal.20231030.154912).
ros-noetic-gazebo-ros-pkgs is already the newest version (2.9.2-1focal.20240111.185854).
0 upgraded, 0 newly installed, 0 to remove and 73 not upgraded.
dhruv@dhruv-VirtualBox:~$
```

Now run **gazebo** on terminal to open gazebo

Output –

A simple gazebo with empty world will open in gazebo.



## 5.3 Importing Husky Vehicle in Gazebo

To import husky in gazebo firstly need to run some commands on terminal.

### 5.3.1 Install Husky Simulator

```
dhruv@dhruv-VirtualBox:~$ sudo apt install ros-noetic-husky-simulator ros-noetic-husky-desktop
[sudo] password for dhruv:
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-husky-desktop is already the newest version (0.6.10-1focal.20240521.160709).
The following additional packages will be installed:
  ros-noetic- Hector-gazebo-plugins ros-noetic-husky-control
  ros-noetic-husky-gazebo ros-noetic-joint-trajectory-controller
  ros-noetic-pointcloud-to-laserscan ros-noetic-tf2-sensor-msgs
  ros-noetic-velodyne-gazebo-plugins
The following NEW packages will be installed:
  ros-noetic- Hector-gazebo-plugins ros-noetic-husky-control
  ros-noetic-husky-gazebo ros-noetic-husky-simulator
  ros-noetic-joint-trajectory-controller ros-noetic-pointcloud-to-laserscan
  ros-noetic-tf2-sensor-msgs ros-noetic-velodyne-gazebo-plugins
0 upgraded, 8 newly installed, 0 to remove and 73 not upgraded.
Need to get 0 B/1,077 kB of archives.
After this operation, 7,597 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

### 5.3.2 Setup the Workspace

```
dhruv@dhruv-VirtualBox:~$ mkdir -p ~/catkin_ws/src
dhruv@dhruv-VirtualBox:~$ cd ~/catkin_ws/
dhruv@dhruv-VirtualBox:~/catkin_ws$ catkin make
```

### 5.3.3 Source the workspace

```
dhruv@dhruv-VirtualBox:~/catkin_ws$ source /opt/ros/noetic/setup.bash
dhruv@dhruv-VirtualBox:~/catkin_ws$ source ~/catkin_ws/devel/setup.bash
```

### 5.3.4 Adding husky , ground plane , tree in the script file

Now before launching the husky file , we must include one ground plane (Asphalt plane ) where the husky will move and then we include some obstacles in between the plane like Pine trees.

To include this all in our .world file we need to add the code of Husky , ground plane and the tree code inside the file as shown below in SDF world file

**.xml file -**

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://asphalt_plane</uri>
    </include>

    <!-- Husky robot model -->
    <include>
      <uri>model://husky</uri>
      <name>husky</name>
      <pose>0 0 0 0 0 0</pose>
    </include>

    <!-- Tree model -->
    <include>
      <uri>model://tree</uri>
      <name>tree</name>
      <pose>2 2 0 0 0 0</pose>
    </include>

    <physics name='default_physics' default='0' type='ode'>
      <gravity>0 0 -9.8066</gravity>
      <ode>
        <solver>
          <type>quick</type>
          <iters>500</iters>
          <sor>1.3</sor>
          <use_dynamic_moi_rescaling>0</use_dynamic_moi_rescaling>
        </solver>
        <constraints>
          <cfm>0</cfm>
          <erp>0.2</erp>
          <contact_max_correcting_vel>100</contact_max_correcting_vel>
          <contact_surface_layer>0.001</contact_surface_layer>
        </constraints>
      </ode>
    </physics>
  </world>
</sdf>
```



```

    </constraints>
  </ode>
  <max_step_size>0.001</max_step_size>
  <real_time_factor>1</real_time_factor>
  <real_time_update_rate>1000</real_time_update_rate>
  <magnetic_field>6e-06 2.3e-05 -4.2e-05</magnetic_field>
</physics>
</world>
</sdf>

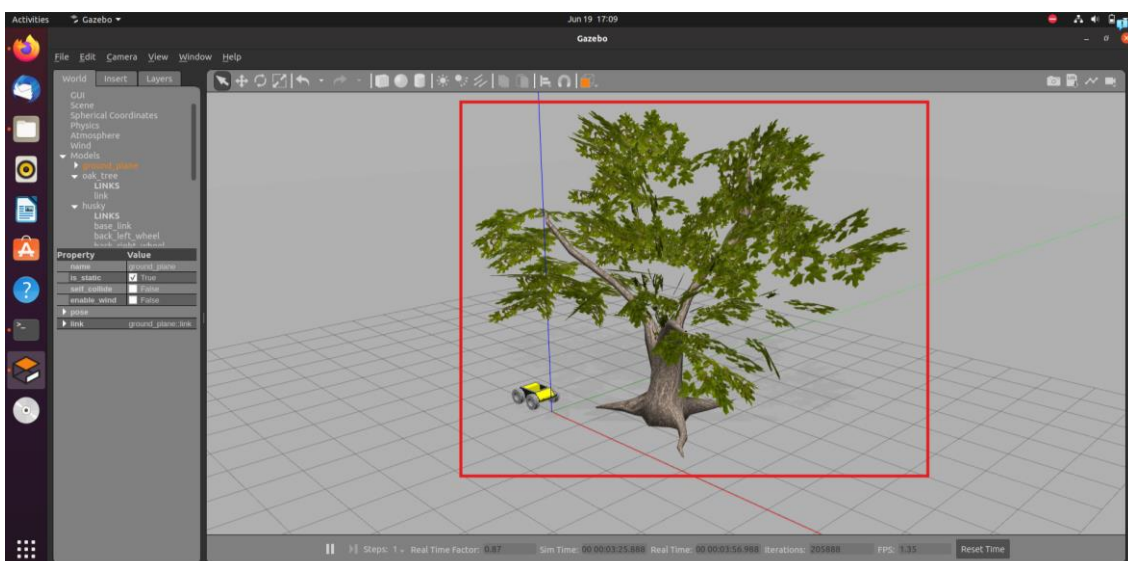
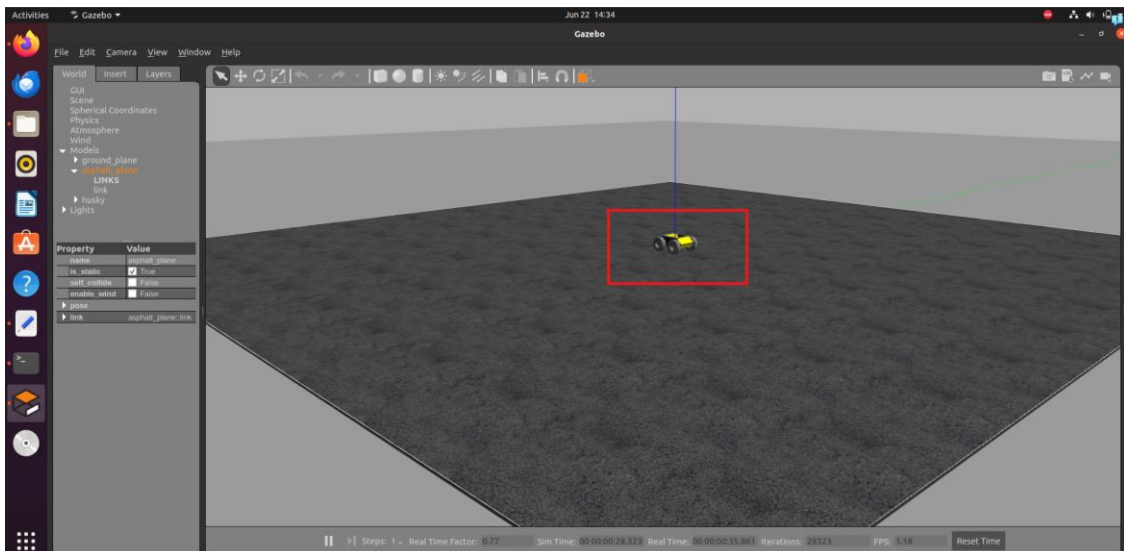
```

### 5.3.5 Launching the Husky Vehicle

```
dhruv@dhruv-VirtualBox:~/catkin_ws$ roslaunch husky_gazebo husky_empty_world.launch
```

Output –

After executing the commands, gazebo opens and we can have the following output –



## 5.4 Teleoperation on Husky Vehicle

Teleoperation, also known as remote operation, involves the control of a machine or vehicle from a distance. In the context of a Husky vehicle, which is a popular all-terrain unmanned ground vehicle (UGV) often used in research and industrial applications, teleoperation allows an operator to navigate and control the vehicle remotely. This capability is crucial in environments that are hazardous, inaccessible, or impractical for human presence.

### 5.4.1 System Architecture

**Control Station:** This is the interface through which the operator sends commands to the vehicle. It can include joysticks, computers, or specialized control panels.

**Sensors:** These provide critical feedback to the operator about the vehicle's environment and state. Common sensors include cameras, LIDAR, GPS, IMUs, and ultrasonic sensors.

### 5.4.2 Communication Challenges

**Latency:** Delays in communication can affect the operator's ability to control the vehicle accurately. Minimizing latency is crucial for effective teleoperation.

**Bandwidth:** Adequate bandwidth is necessary to transmit high-resolution sensor data, particularly video feeds, which are essential for situational awareness.

### 5.4.3 Applications

**Search and Rescue:** Navigating through hazardous environments to locate and assist victims.

**Surveillance and Reconnaissance:** Gathering information in dangerous or inaccessible areas.

#### 5.4.3 Installing Dependencies

We need to install some dependencies to execute teleoperation on husky vehicle .

```
sudo apt-get install ros-noetic-fath-pivot-mount-description
sudo apt-get install ros-noetic-flir-camera-description
sudo apt-get install ros-noetic-velodyne-description
sudo apt install ros-noetic-realsense2-description
sudo apt-get install ros-noetic-lms1xx
sudo apt-get install ros-noetic-robot-localization
sudo apt-get install ros-noetic-interactive-marker-twist-server
sudo apt-get install ros-noetic-twist-mux
sudo apt-get install ros-noetic-teleop-twist-keyboard
sudo apt-get install ros-noetic-teleop-twist-joy
sudo apt-get install ros-noetic-rviz-imu-plugin
sudo apt-get install ros-noetic-gmapping
```

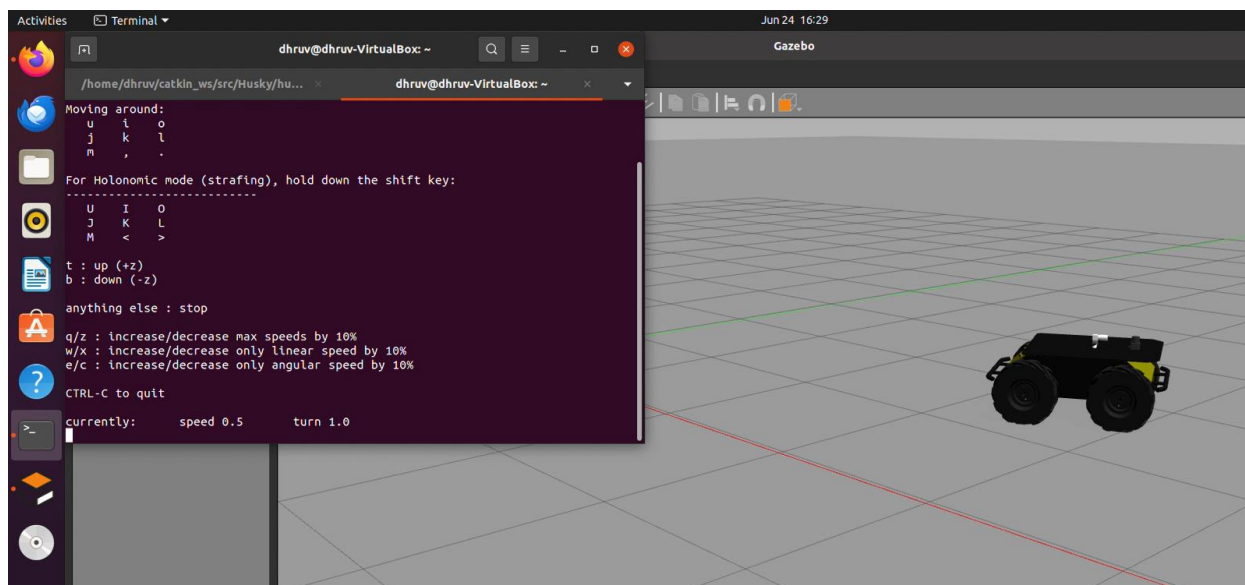
## 5.4.5 Installations

```
$ mkdir -p catkin_ws/src
$ cd catkin_ws/src
$ cd ..
$ catkin_make
```

## 5.4.6 Tele operation Commands

```
$ roslaunch husky_gazebo husky_playpen.launch
$ roslaunch husky_control teleop_keyboard.launch
```

### OUTPUT -





# Bibliography

- [1] <https://www.diva-portal.org/smash/get/diva2:1505194/FULLTEXT01.pdf>
- [2] <https://wiki.ros.org/noetic/Installation/Ubuntu>
- [3] <https://github.com/Varun0157/Image-Based-Visual-Servoing>
- [4] <https://github.com/Tinker-Twins/Husky?tab=readme-ov-file>
- [5] <https://github.com/gazebo-sim/gazebo-classic/tree/gazebo11/worlds>
- [6] [https://docs.opencv.org/4.x/d5/dae/tutorial\\_aruco\\_detection.html](https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html)
- [7] <https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/>
- [8] [https://docs.opencv.org/4.x/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html)
- [9] <https://chev.me/arucogen/>