Artificial Intelligence
& Electronics Society

ArIES

IIT Roorkee

Recruitment Manual 2024

❖ Verticals :

1) Drones,Robotics and Mechatronics
2) AI/ML_DL
3) Dev & AR_VR

1. Drones, Robotics and Mechatronics

“Drone” - Components and Working :

- Unmanned Aerial Vehicle and Model Aircraft

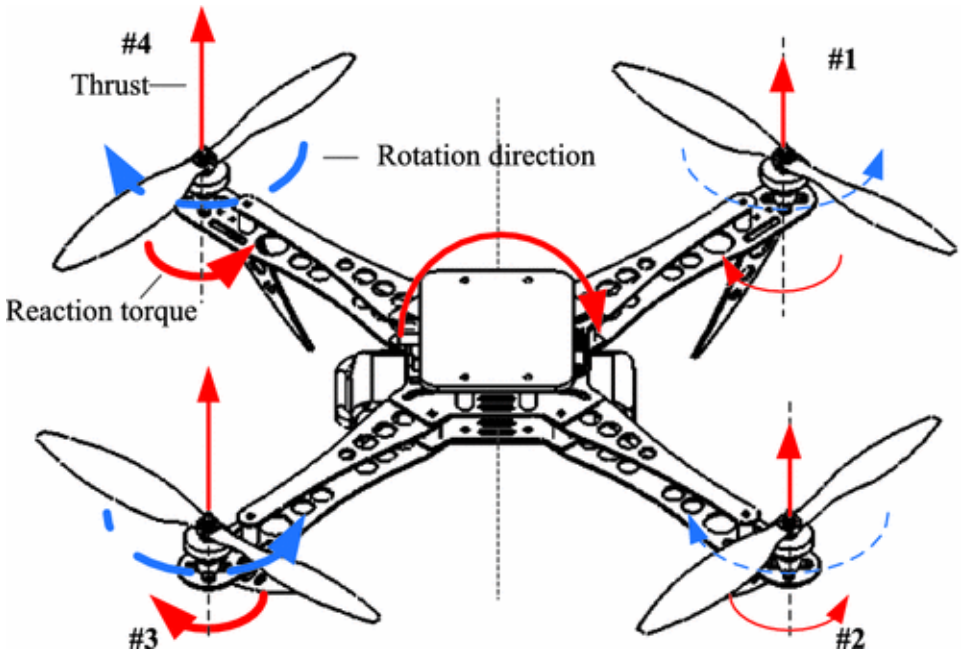
  1). Unmanned Aerial Vehicle or Uninhabited Aerial Vehicle (UAV).The flight of UAVs may be controlled either autonomously by onboard computers or by the remote control from a pilot on the ground or in another vehicle. UAVs are also called drones.

  2). Model Aircraft. “An aircraft of limited dimensions, with or without a propulsion device, not able to carry a human being and to be used for aerial competition, sport or recreational purposes” is called a model aircraft.

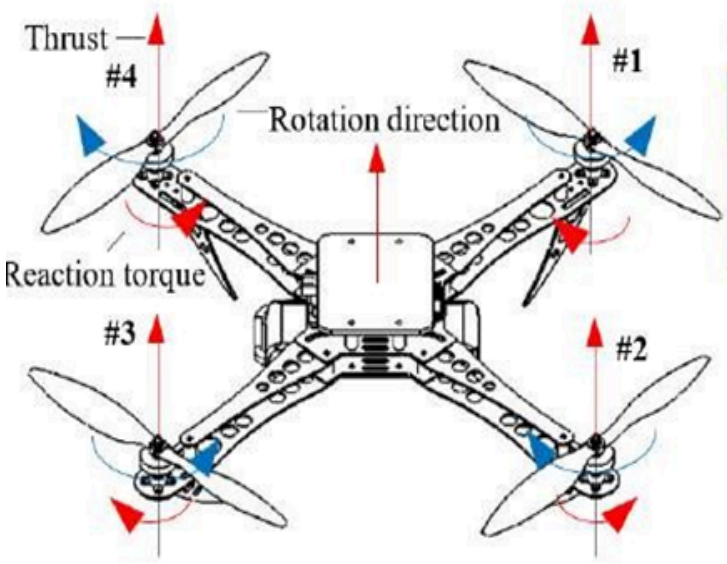
- Remote control of a Quadcopter

1) **At Hover Position :**

- Sum of the four produced thrusts compensates for the weight.
- Thrusts of four rotors are the same and the moments of four Rotors sum to zero.
- About the yaw moment. If the blades are spinning counter Clockwise , then the airframe will start to rotate clockwise due to The reaction torque. This is due to Newton’s Third law.



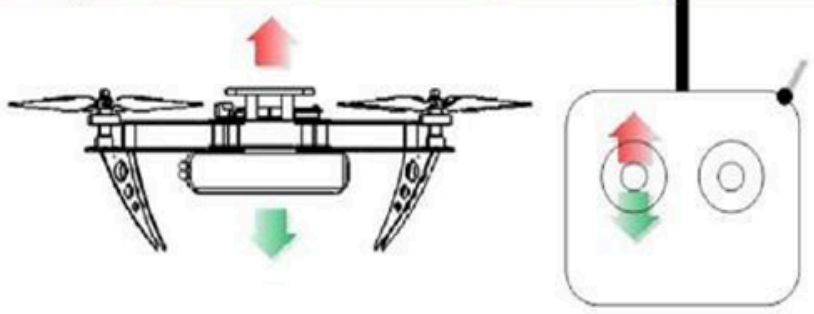
2) Upward - Downward Movement :



Upward movement of a quadcopter

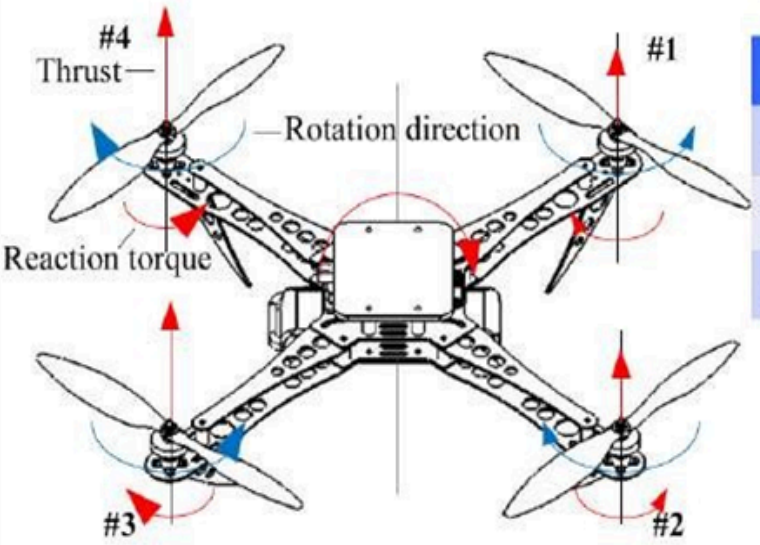
increase, "-" indicates decrease)

	#1	#2	#3	#4
Thrust change	+1	+1	+1	+1



Operation of an RC transmitter for the upward and downward movement

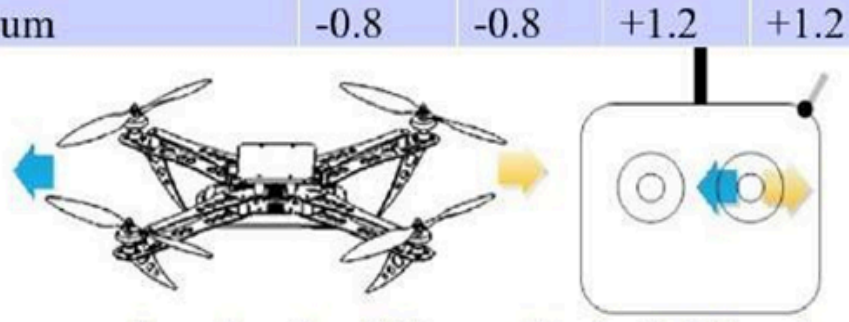
3) Forward - Backward Movement :



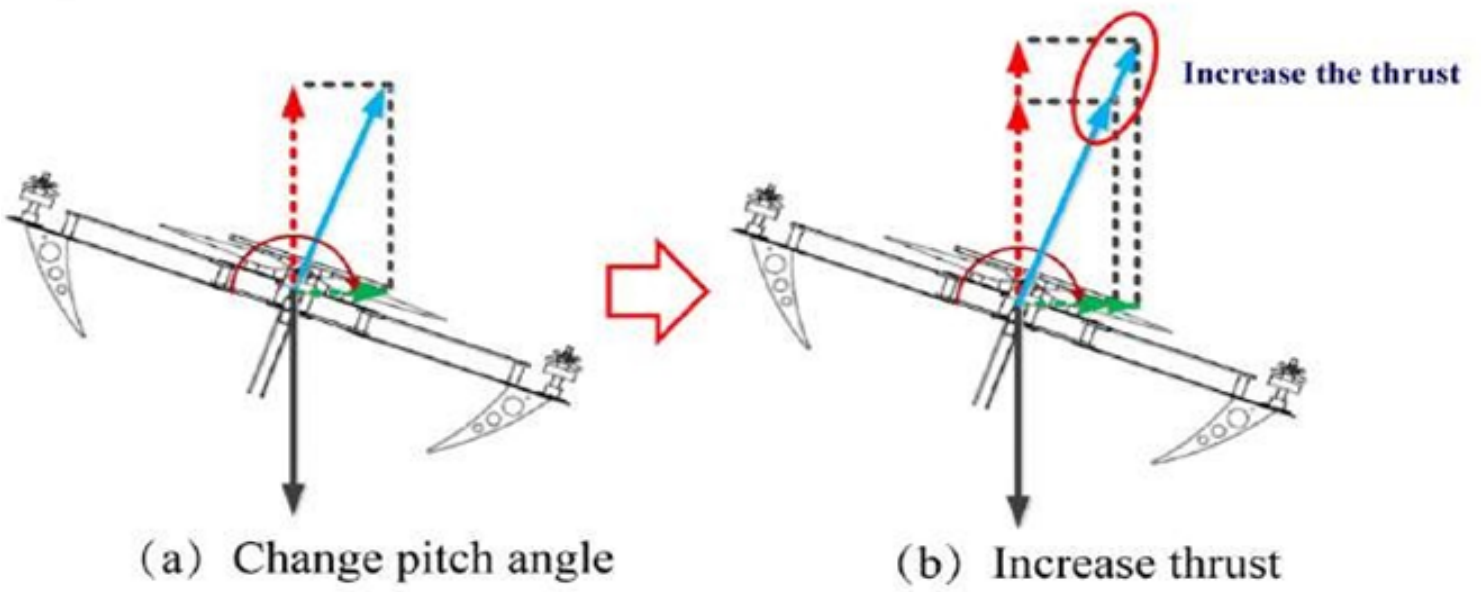
Rightward movement of a quadcopter

Change of propellers' angular velocities ("+"indicates increase, "-" indicates decrease)

	#1	#2	#3	#4
Change roll angle	-1	-1	+1	+1
Increase thrust	+0.2	+0.2	+0.2	+0.2
Sum	-0.8	-0.8	+1.2	+1.2

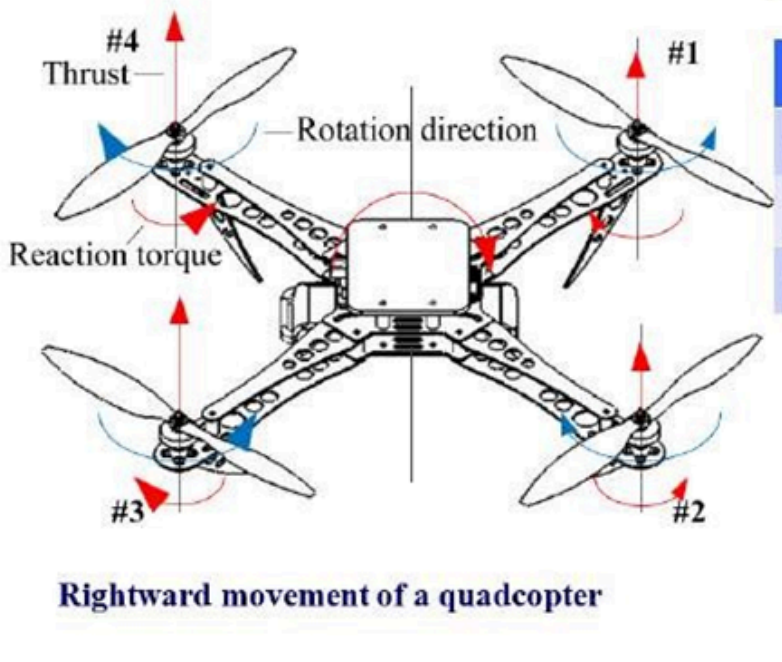


Operation of an RC transmitter for the leftward and rightward movement



After the pitch angle exists, four propellers' angular velocities should be further increased by the same amount to compensate for the weight.

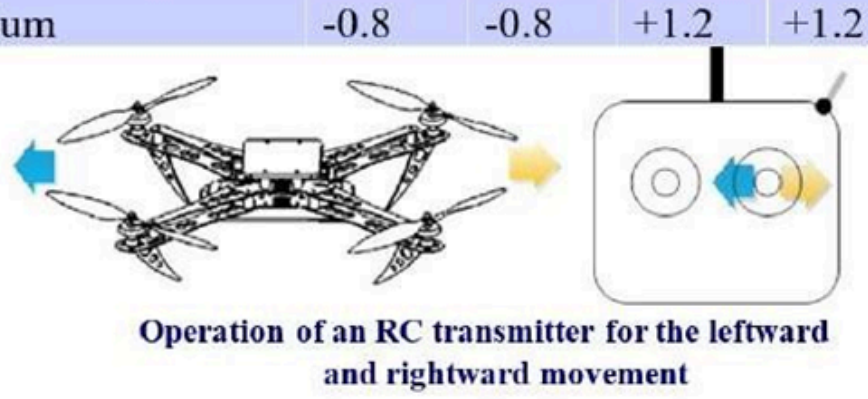
4) Leftward and Rightward Movement :



Rightward movement of a quadcopter

Change of propellers' angular velocities ("+"indicates increase, "-" indicates decrease)

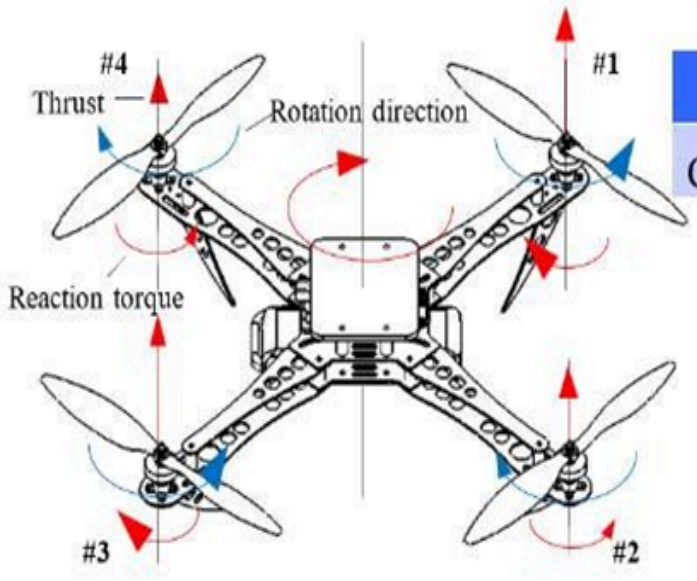
	#1	#2	#3	#4
Change roll angle	-1	-1	+1	+1
Increase thrust	+0.2	+0.2	+0.2	+0.2
Sum	-0.8	-0.8	+1.2	+1.2



Operation of an RC transmitter for the leftward and rightward movement

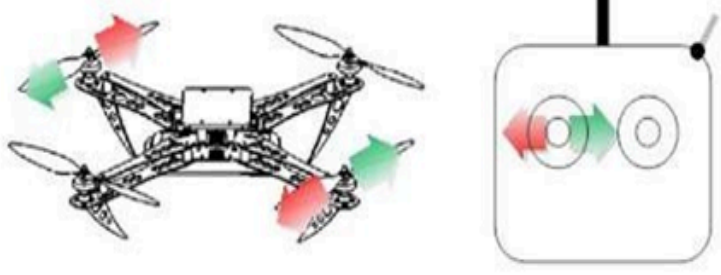
5) Yaw Movement :

Change of propellers' angular velocities ("+"indicates increase, "-" indicates decrease)



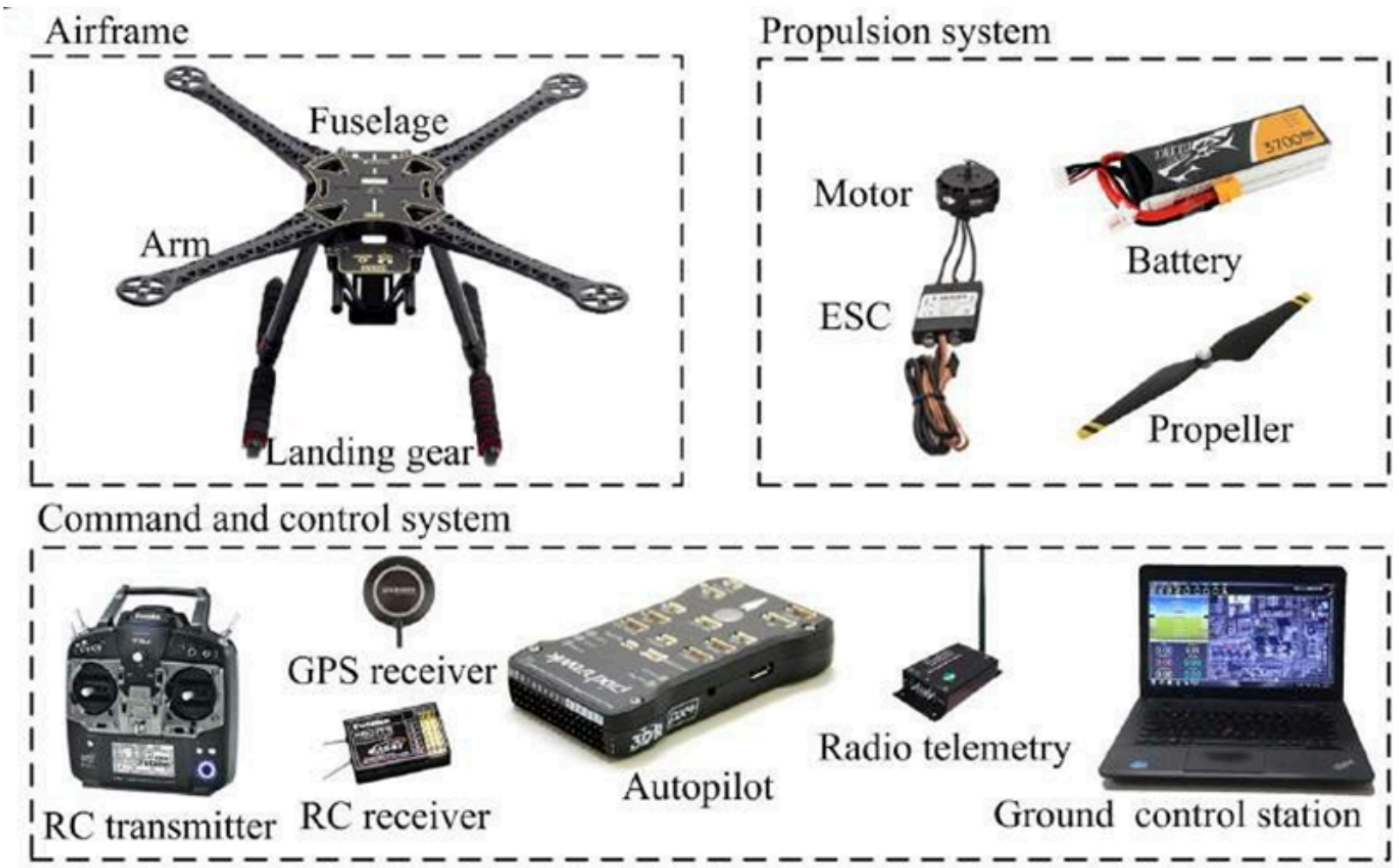
Clockwise yaw movement of a quadcopter

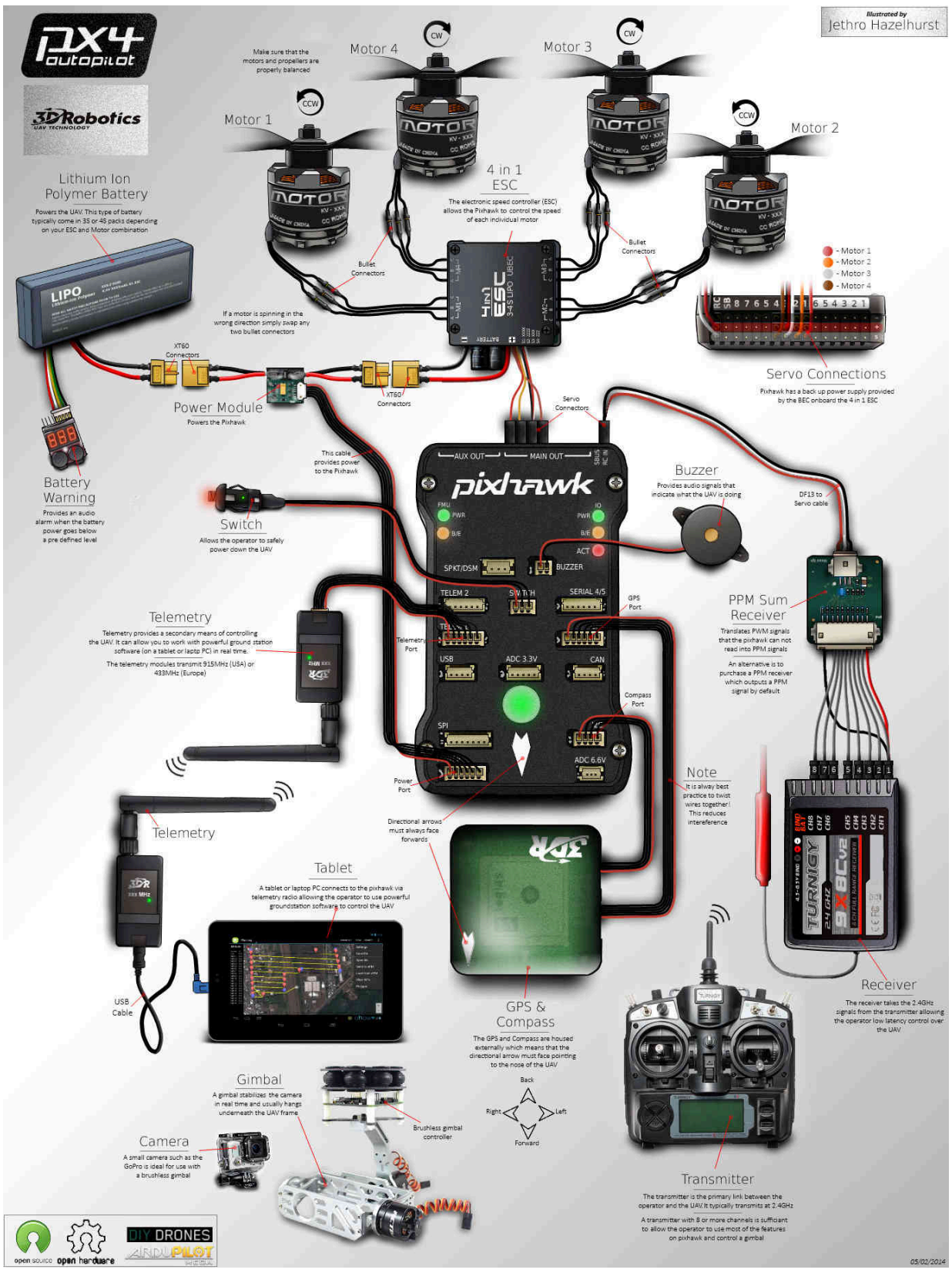
	#1	#2	#3	#4
Change yaw angle	+1	-1	+1	-1

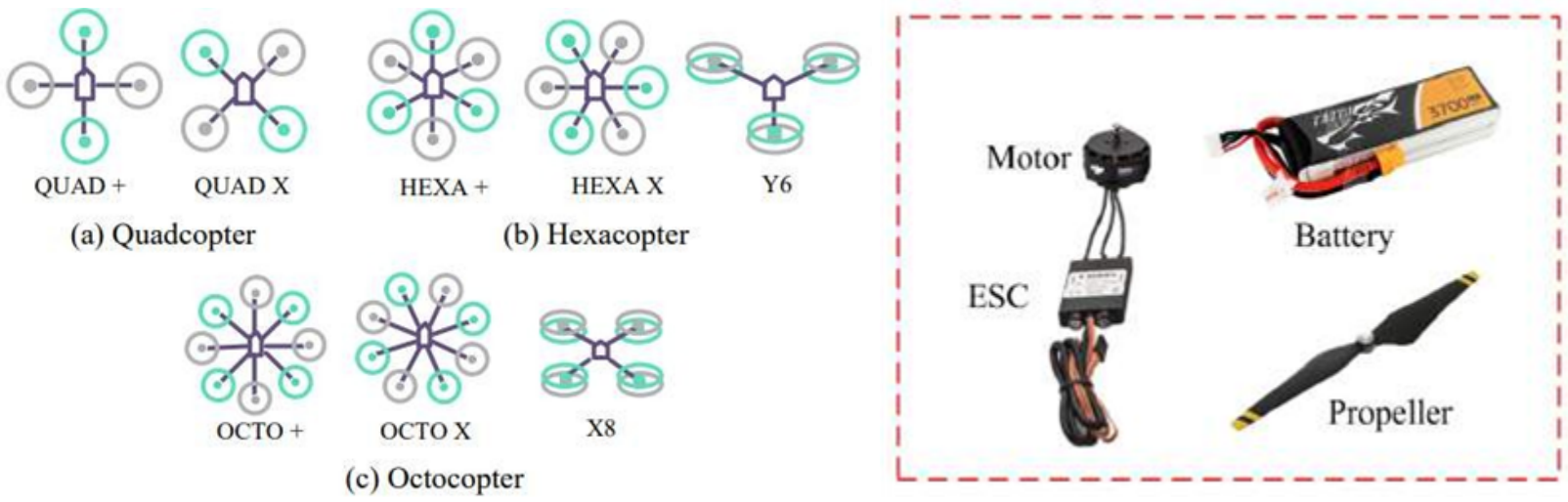


Operation of an RC transmitter for yaw movement

- Basic Components of a multicopter system







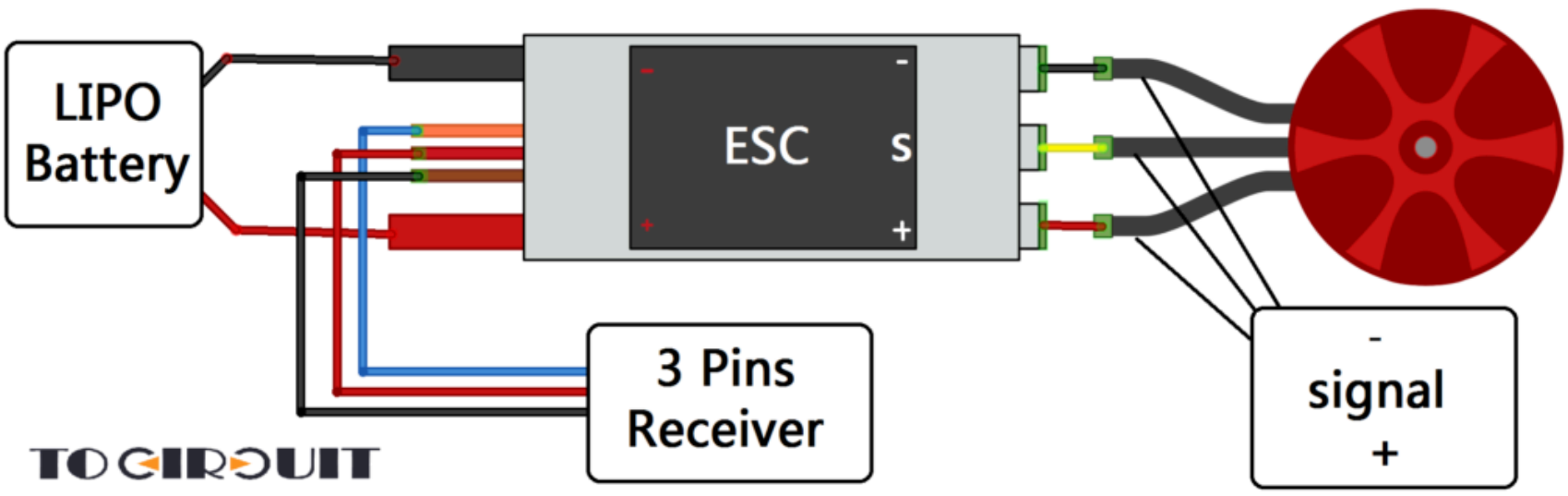
(a) Multicopter configurations (b) Propulsion System

1). A propulsion system includes propellers, motors, ESCs and often a battery.

2). The propulsion system determines the main performances of multicopter like the hovering time, the payload ability, the flying speed and distance.

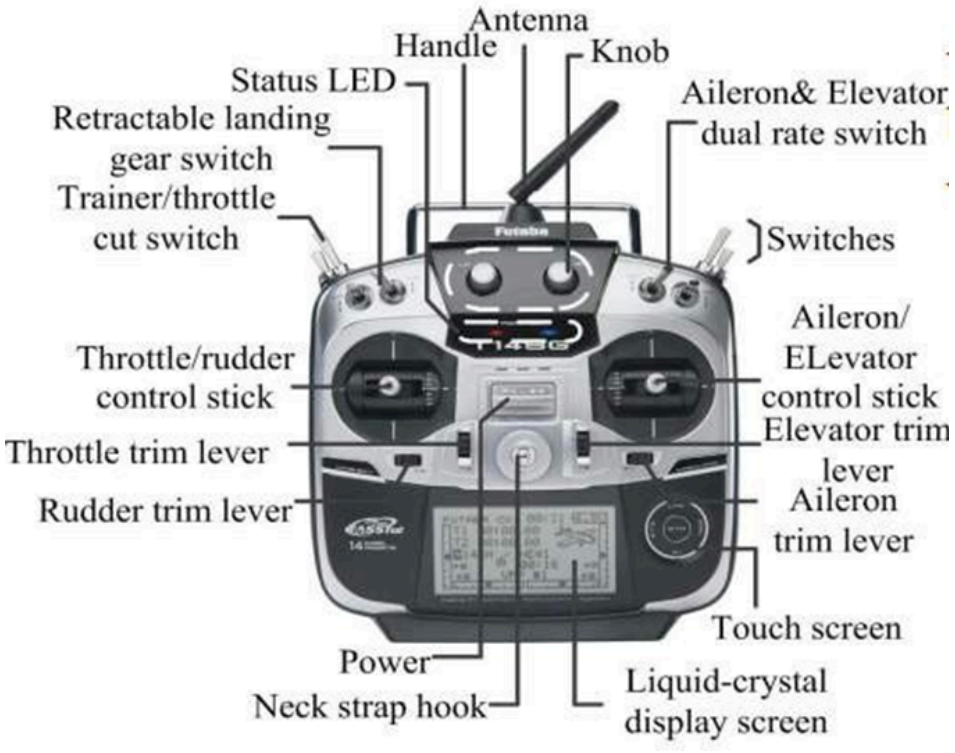
3). Components of the propulsion system have to be compatible with each other, otherwise they cannot work properly or even fails in some extreme cases.

☐ Electronic Speed Controller (ESC)



- Controlling the speed of motors based on the PWM signal that autopilots send.
- Supplying power for Radio Controlled (RC) servo motors.
- Transforming the onboard DC power into the three-phase Alternating Current (AC) power thatcan be applied to brushless DC motors.
- The auxiliary functions include battery protection and starting protection.

☐ RC Transmitter & Receiver



Futaba RC Transmitter

1).The RC transmitter is used to transmit commands from remote pilot to the corresponding receiver, then the receiver passes the commands to the autopilot after decoding them, finally, the multicopter flies according to the commands.

2). Some flight parameters can be set on the transmitter, such as the throttle direction, stick sensitivity, neutral position of RC servo motors, function definitions of channels, record and remind setting of flight time, and lever function setting.

☐ Autopilot

A multicopter autopilot can be divided into the software part and hardware part:

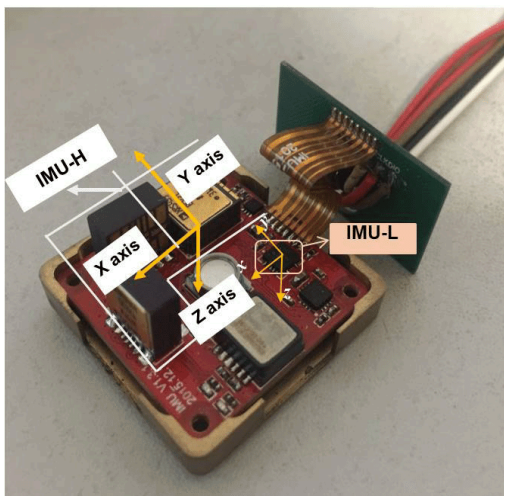
- Global Position System (GPS) receiver. It is used to obtain the location information of multicopters.

- Inertial Measurement Unit (IMU). It includes: three-axis accelerometer, three-axis gyroscope, electronic compass (or three-axis magnetometer). It is used to obtain attitude information of a multicopter.

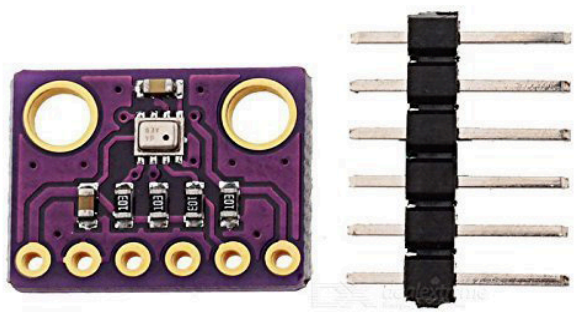
- Height sensor. Height sensor includes the barometer and ultrasonic range finder, which are used to obtain the absolute height and relative height, respectively.

- Microcomputers. It is a platform used to receive information and run algorithms to produce control commands.

- Interface. It acts as a bridge between the microcomputer and the other devices, like the sensors, ESC and RC receiver.



IMU Module



Barometer

AI

Ultrasonic sensor

☐ Ground Control System (GCS)

- Software is an important part of GCS.
- Remote pilots can interact with the software using the mouse, keyboard, button and joystick.
- Remote pilot can plan the way point in advance, therefore the remote pilot can monitor the flight status in real time and set new missions to intervene flight.
- The software can record and playback flight for analysis.

Some Important References :

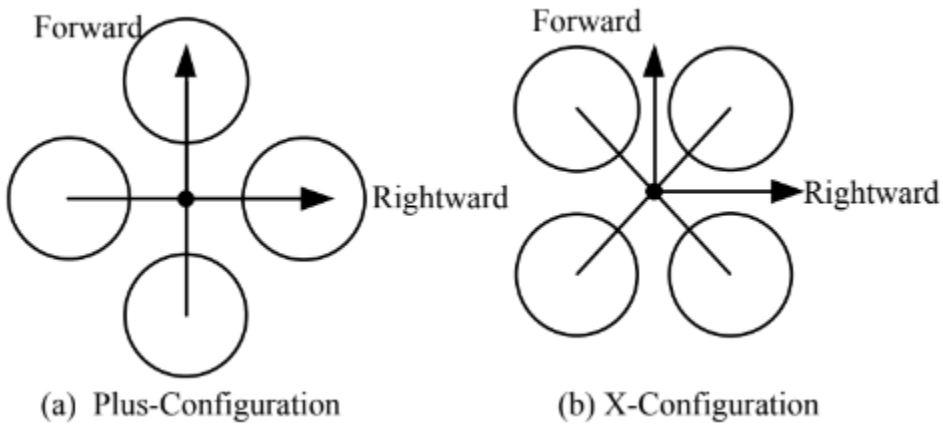
1. https://dronenodes.com/how-to-build-a-drone/_tab-con-12
2. https://catsr.vse.gmu.edu/SYST460/DroneComponents.pdf
3. https://grinddrone.com/drone-features/drone-components
4. (650) How to Make a Drone at Home - For Beginner - YouTube
5. https://www.youtube.com/watch?v=jOugJpQfUDU

Mechanical Design Considerations:

A number of factors need to be considered when designing a drone frame and can be broken down into two aspects.

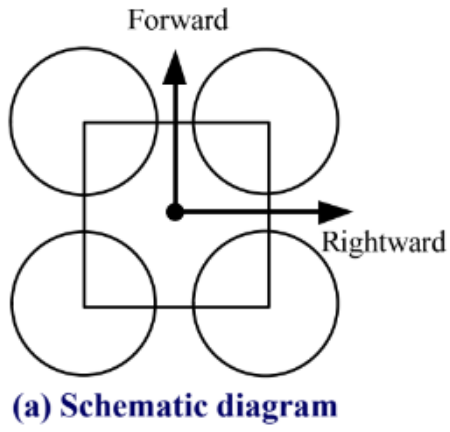
1. Configuration Design
   - Fuselage Configuration

     1.Cross-Configuration: According to the head direction, there are Plus-configuration and X-configuration. X-configuration are more suitable due to their higher maneuverability.



(a) Plus-Configuration (b) X-Configuration

2.Ring-Configuration: The ring configuration can be treated as a whole, which is more rigid than the traditional cross fuselage. It can reduce the vibration produced by the rotors more efficiently.



(a) Schematic diagram

- Propeller Mounting:
  1. Common-Form and Co-Axis Form:

     a. Advantages of co-axis form

     • Increase lift without increasing size

     • Prevent cameras from occlusion

b.. Attentions

• Co-axis form may reduce the efficiency of the single propeller, approximately equivalent to 1.6 propellers in single-propeller mode.

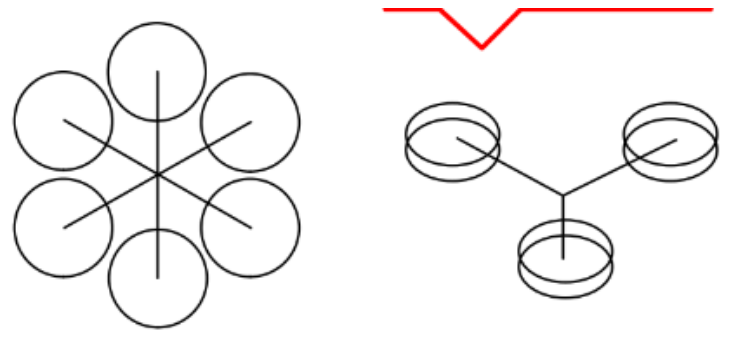


Fig 3.3 Common form and co-axis form

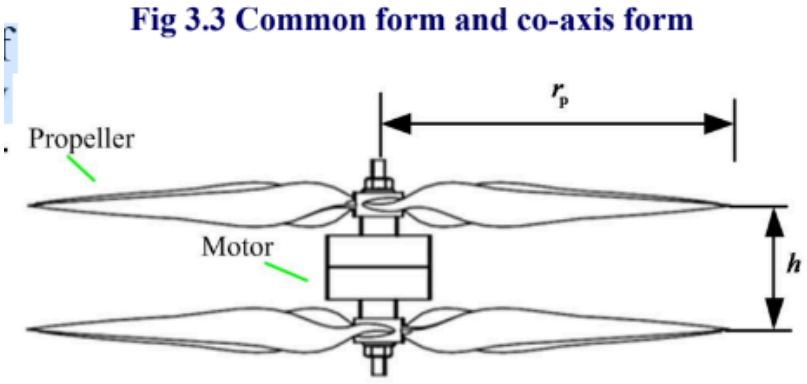
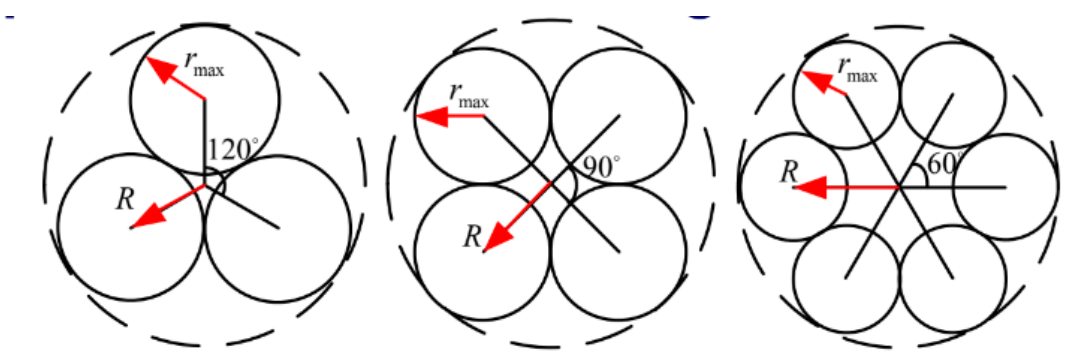


Fig 3.4 connection and geometry a co-axis form

2. Propeller-Radius and Fuselage Radius:



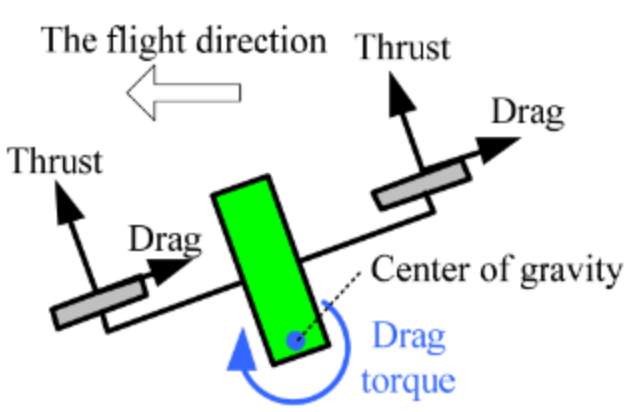
(a) Y6-configuration hexacopter (b) Quadcopter (c) Hexacopter

Fig 3.7 Multicopters with different configurations and their geometry parameters

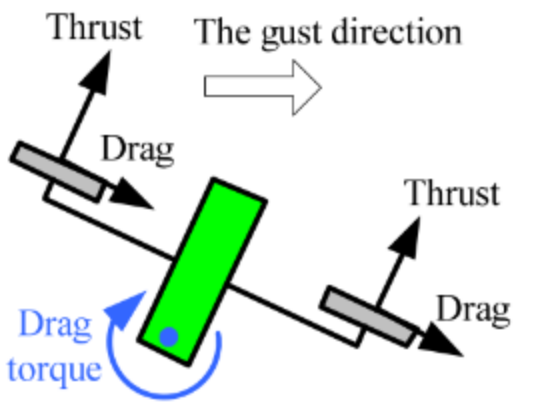
The relationship between airframe radius R and the maximum radius of a propeller r_{max} is denoted by (θ is the angle between two arms, n_r is the number of arms)

$$R = \frac{r_{max}}{\sin\frac{\theta}{2}} = \frac{r_{max}}{\sin\frac{180^\circ}{n_r}}$$

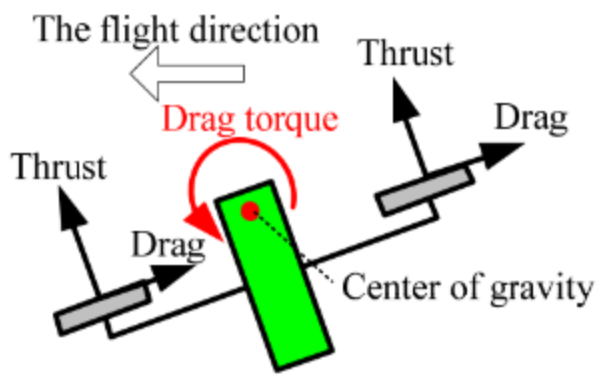
- Position of COG:When designing a drone frame, it is necessary to ensure that after the inclusion of components the COG of the drone lies in the plane of the propeller. In case of high and low COG additional drag moment comes into picture and can affect the drones performance.



(a) Forward flight with low CoG

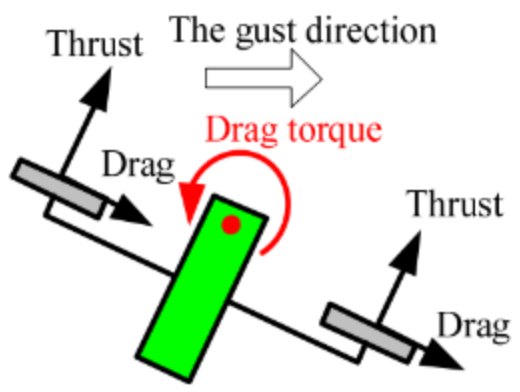


(c) Wind interference with high CoG



(b) Forward flight with high CoG

Fig 3.9 Forces on multicopters



(d) Wind interference with low CoG

Fig 3.9 Forces on multicopters

2. Structural Design: The design of the drone frame should be as rigid and lightweight as possible with proper dimensions and aesthetics.

   Vibrations being a common cause for instabilities and degradation of Performance of drones, it is required that proper measures for vibration mitigation is considered in the design.

- Sources of Vibration:
  - Airframe: Recently, the concept of wearable quadcopters has received more and more attention, such as Nixie, which requires that its airframe is flexible and easily transformed,which creates inflight vibrations.
  - Motor: For a motor having a cog not aligned with the shaft axis, rotation of the shaft creates a sinusoidal force due to the eccentricity of mass.
  - Propeller:Less Rigid propellers vibrate as the air flows around it during rotation due to the force on the propeller due to the pressure-difference which arises due to difference in velocity above and below the propeller airfoil. To counter these vibrations, propellers should be as rigid as possible and carbon fiber propellers are an ideal solution for this, due to their lightweight nature and high rigidity.

Some Important References:

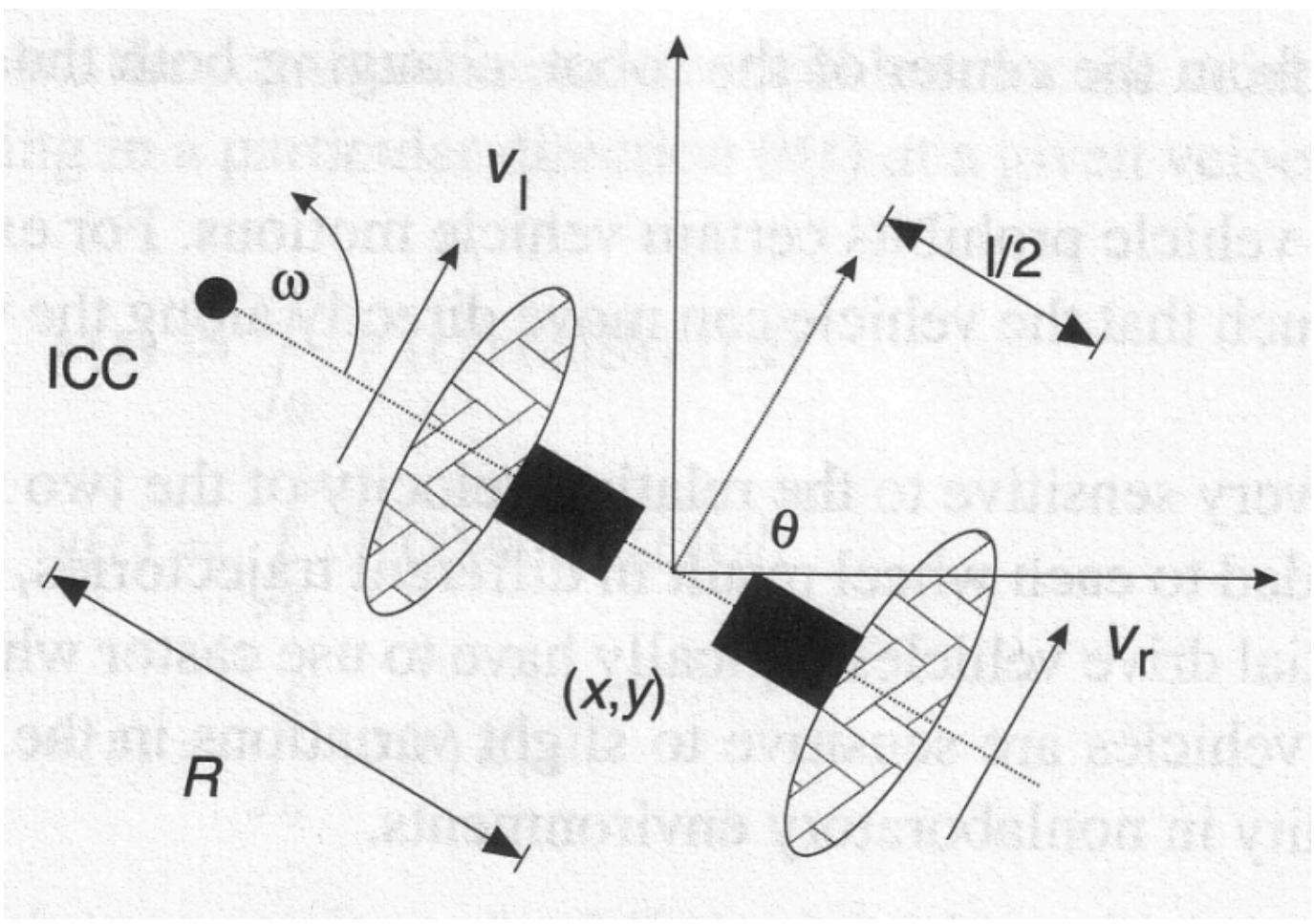
(1)Exploring UAV Design Principles

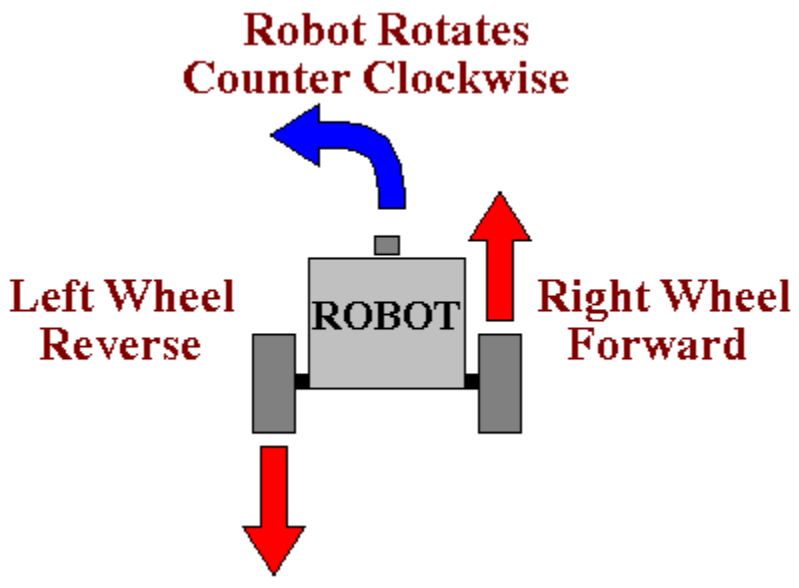
(2)What Are Drones Made Of? (Materials And Components Explained) (dronesgator.com)

(3)Reduce Vibration in Drones: Enhance Flight & Image Quality - TechMeStuff

Differential Drive Model of a robot :

The differential drive is a two-wheeled drive system with independent actuators for each wheel. The name refers to the fact that the motion vector of the robot is the sum of the independent wheel motions. The drive wheels are usually placed on each side of the robot and toward the front.





By varying the velocities of the two wheels, we can vary the trajectories that the robot takes. Because the rate of rotation ω about the ICC must be the same for both wheels, we can write the following equations :

$$\omega (R + l/2) = V_r \quad .. (1)$$

$$\omega (R - l/2) = V_l \quad .. (2)$$

where l is the distance between the centers of the two wheels, Vr , Vl are the right and left wheel velocities along the ground , and R is the signed distance from the ICC to the midpoint between the wheels.

At any instance in time we can solve for R and ω :

$$R = \frac{l}{2}\frac{V_l + V_r}{V_r - V_l};\ \omega = \frac{V_r - V_l}{l}; \tag{3}$$

There are three interesting cases with these kinds of drives.

1. If $V_l = V_r$, then we have forward linear motion in a straight line. R becomes infinite, and there is effectively no rotation - ω is zero.
2. If $V_l = -V_r$, then $R = 0$, and we have rotation about the midpoint of the wheel axis - we rotate in place.
3. If $V_l = 0$, then we have rotation about the left wheel. In this case $R = \frac{l}{2}$. Same is true if $V_r = 0$.

Note that a differential drive robot cannot move in the direction along the axis - this is a singularity. Differential drive vehicles are very sensitive to slight changes in velocity in each of the wheels. Small errors in the relative velocities between the wheels can affect the robot trajectory. They are also very sensitive to small variations in the ground plane, and may need extra wheels (castor wheels) for support.

• Forward Kinematics of differential drive robots :

In figure 1, assume the robot is at some positon (x, y), headed in a direction making an angle θ with the X axis. We assume the robot is centered at a point midway along the wheel axle. By manipulating the control parameters V_l , V_r, we can get the robot to move to different positions and orientations. (note: V_l , V_r) are wheel velocities along the ground).

Knowing velocities V_l, V_r and using equation 3, we can find the ICC location:

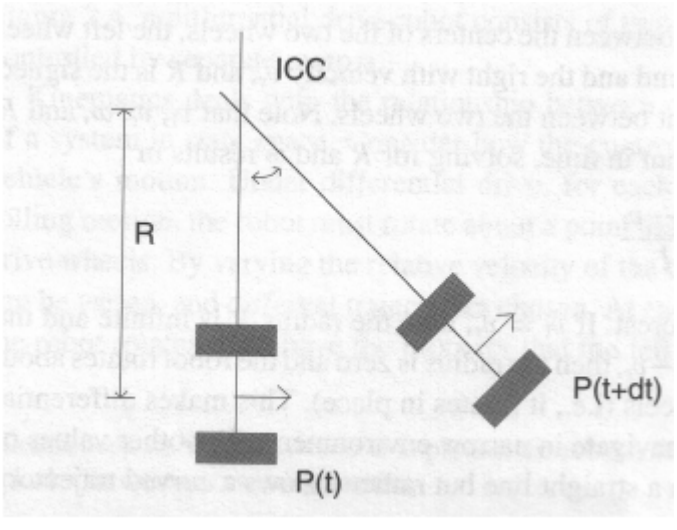
$$ICC = [x - R\,sin(\theta)\,,\ y + R\,cos(\theta)] \tag{4}$$

and at time $t + \delta t$ the robot's pose will be:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} cos(\omega\delta t) & -sin(\omega\delta t) & 0 \\ sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{5}$$

This equation simply describes the motion of a robot rotating a distance R about its ICC with an angular velocity of ω.

Refer to figure 2. Another way to understand this is that the motion of the robot is equivalent to 1) translating the ICC to the origin of the coordinate system, 2) rotating about the origin by an angular amount $\omega\,\delta t$, and 3) translating back to the ICC.



- Inverse Kinematics of differential drive robots :

In general, we can describe the positon of a robot capable of moving in a particular direction $\Theta(t)$ at a given velocity $V(t)$ as:

$$x(t) = \int_0^t V(t)cos[\theta(t)]dt$$

$$y(t) = \int_0^t V(t)sin[\theta(t)]dt$$

$$\Theta(t) = \int_0^t \omega(t)dt$$

For the special case of a differential drive robot such as the GoPiGo, the equations become:

$$x(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]cos[\theta(t)]dt$$

$$y(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]sin[\theta(t)]dt$$

$$\Theta(t) = \frac{1}{l}\int_0^t [v_r(t) - v_l(t)])dt$$

A related question is: How can we control the robot to reach a given configuration (x, y, θ) - this is known as the inverse kinematics problem. Unfortunately, a differential drive robot imposes what are called non-holonomic constraints on establishing its position.

For example, the robot cannot move laterally along its axle. A similar nonholonomic constraint is a car that can only turn its front wheels. It cannot move directly sidewise, as parallel parking a car requires a more complicated set of steering maneuvers. So we cannot simply specify an arbitrary robot pose (x, y, θ) and find the velocities that will get us there.

For the special cases of $v_l = v_r = v$ (robot movng in a straight line) the motion equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v\,cos(\theta)\delta t \\ y + v\,sin(\theta)\delta t \\ \theta \end{bmatrix} \tag{6}$$

If $v_r = -v_l = v$, then the robot rotates in place and the equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + 2v\delta t/l \end{bmatrix} \tag{7}$$

This motivates a strategy of moving the robot in a straight line, then rotating for a turn in place, and then moving straight again as a navigation strategy for differential drive robots.

2R Planar Manipulator Robot :

2D Manipulator **Forward** Kinematics

- Forward Kinematics
  - Given **θ**, find **x**

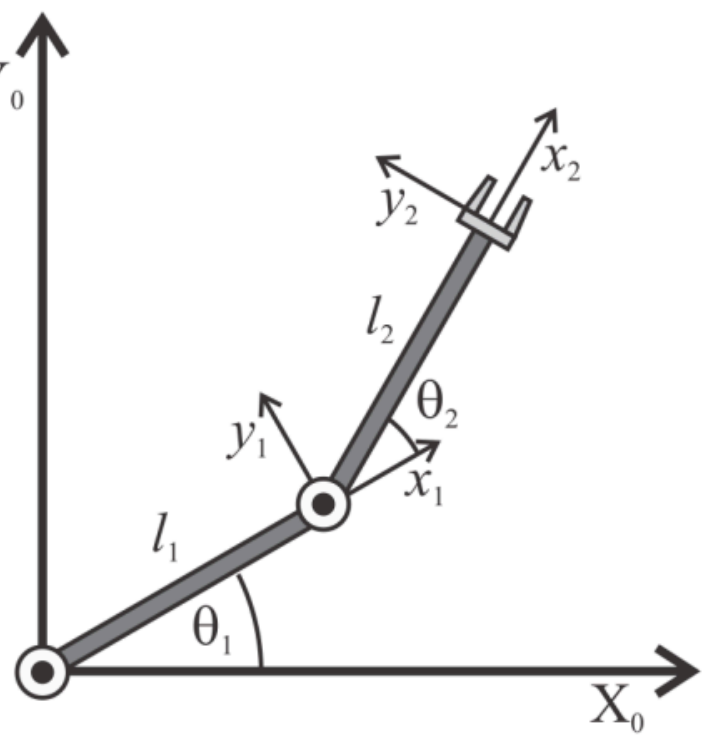
The vector of end effector positions

The vector of joint angles

$$x_2 = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$= l_1 c_1 + l_2 c_{12}$$

Shorthand notation

$$y_2 = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$
$$= l_1 s_1 + l_2 s_{12}$$

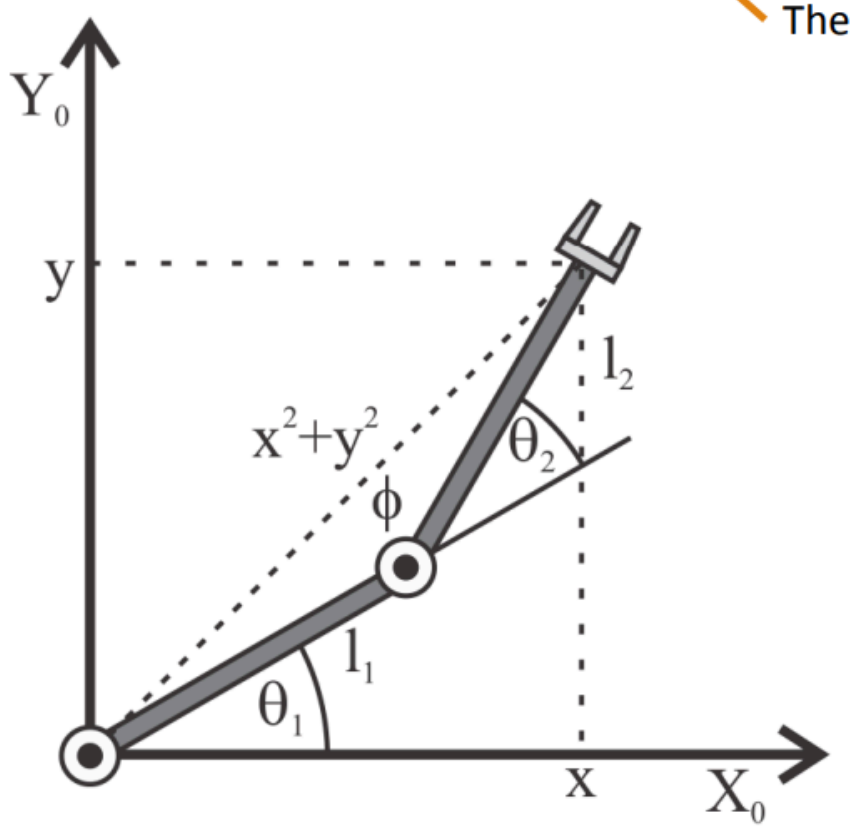


2D Manipulator **Inverse** Kinematics

- Inverse Kinematics = given **x**, find **θ**

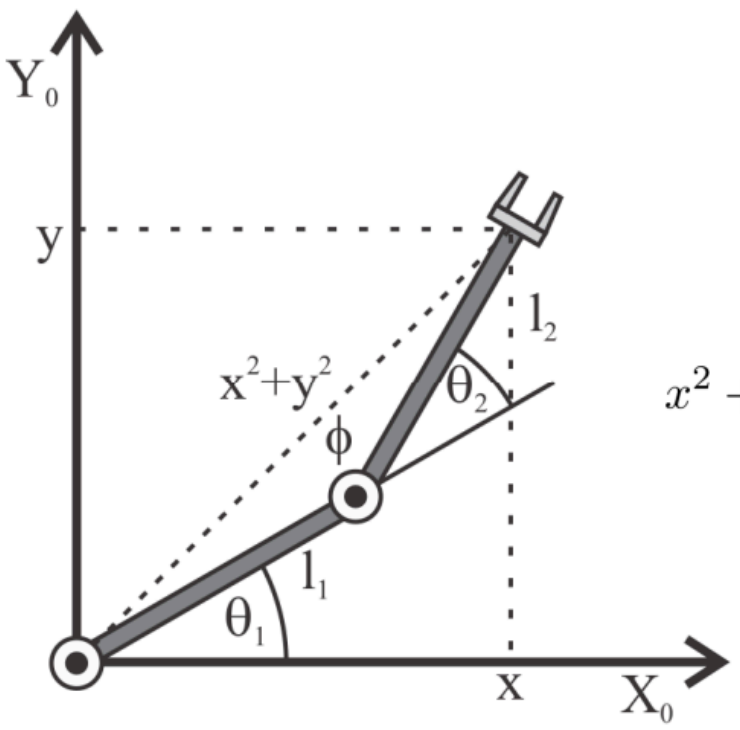
The joint angles

The end effector positions



Problem is more difficult!

2D Manipulator Inverse Kinematics Example



- Use the Law of Cosines:

$$x^2 + y^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos\phi$$

- We also know that:

$$\phi = \pi - \theta_2$$

- Do some algebra to get:

$$\begin{aligned} x^2 + y^2 &= l_1^2 + l_2^2 - 2l_1 l_2 \cos(\pi - \theta_2) \\ &= l_1^2 + l_2^2 - 2l_1 l_2(\cos\pi\cos\theta_2 + \sin\pi\sin\theta_2) \\ &= l_1^2 + l_2^2 - 2l_1 l_2(-\cos\theta_2 + 0) \\ &= l_1^2 + l_2^2 + 2l_1 l_2 c_2 \end{aligned}$$

Again, note the notation short-hand

Inverse Kinematics Example Continued

- Now solve for c2:

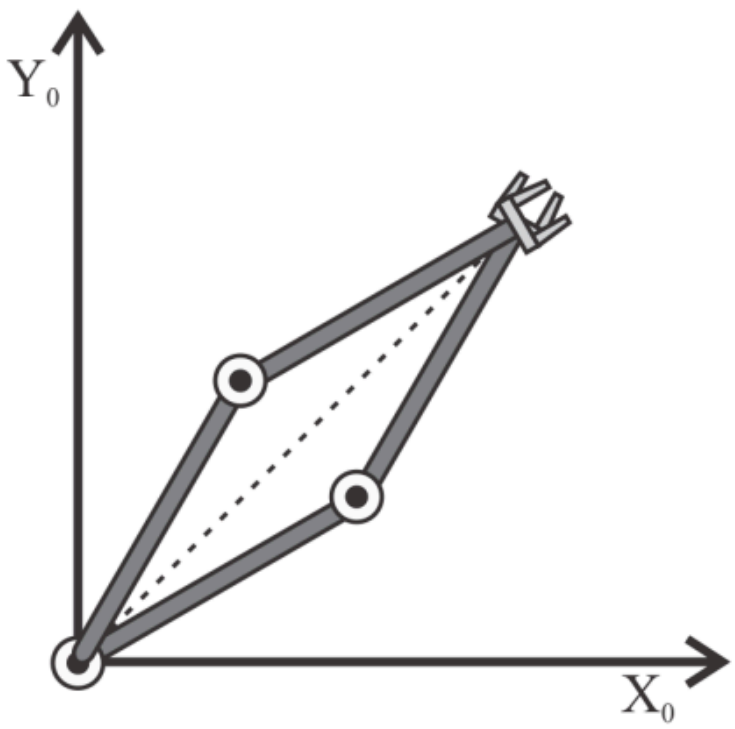
$$x^2 + y^2 = l_1^2 + l_2^2 + 2l_1 l_2 c_2$$

$$c_2 = \frac{x^2+y^2-l_1^2-l_2^2}{2l_1 l_2} \equiv D$$

- One possible solution:

$$\theta_2 = \pm \cos^{-1}\left(\frac{x^2+y^2-l_1^2-l_2^2}{2l_1 l_2}\right)$$

- Elbow up vs elbow down
- May be impossible!
- Multiple solutions may exist!
- Fundamental problem in robotics!



ROS :

ROS, which means **Robot Operating System**, is a set of software libraries and tools to help you build robot applications. The point of ROS is to create a robotics standard, so you don't need to reinvent the wheel anymore when building a new robotic software.

Why should you use ROS for Robotics ?

1). ROS is general

The same base code and knowledge can be applied to many different kinds of robots : robotic arms, drones, mobile bases, etc. Once you've learned about how communication is done between all the nodes of the program, you can set up new parts of an application very easily. In the future, if you need to switch to a totally different robotics project, you won't be lost.

You'll be able to reuse what you know and some parts that you've built, so you'll never really start from scratch again.

2). ROS packages are available for most applications

You need to compute a trajectory for your robot ? There is a package for that. Use your joystick to control the robot ? There is also a package for that. Or you want to map out a room with a drone ? There are many packages to do that.

3). ROS is language - agnostic

You can easily communicate between a Python node and a C++ node. It means a lot of reusability and possibilities of coworking. Many libraries also allow you to use other languages (because ROS has mainly targeted C++ and Python).

4) ROS has great simulation tools

You can't always get your robot running for real, so you need simulation tools. ROS has many great tools, such as Rviz and Gazebo. With Gazebo you can even add some physical constraints to the environment, so when you run the simulation and the real robot, the outcome is pretty much the same. Imagine mapping a room in 3D with a drone directly on your computer, that could save you a huge amount of time.

The simulation tools also allow you to see and use other robots that you don't possess, for educational purposes or to test in a specific environment.

5) you can control Multiple robots with ROS

ROS can work with multiple ROS masters. It means that you can have many independent robots, each with its own ROS system, and all robots can communicate between each other.

6) ROS is light

The core base of ROS doesn't take much space and resources. You can quickly install the core packages and get started in a few minutes. Plus, you can also use ROS on embedded computers, such as Raspberry Pi 3 boards. Thus you can easily start a new project without much trouble.

- Installing ROS : http://wiki.ros.org/ROS/Installation

- Basics of ROS :

1). Creating ROS Workspace

ROS uses catkin to build the ROS Projects. So first we need to create a catkin workspace. Follow the following steps to create your catkin workspace .

First open a terminal in ubuntu. Then navigate to the folder where you want to create a workspace. Then enter the following commands.

```
$ mkdir -p catkin_ws/src
$ cd catkin_ws/
$ catkin_make
$ source devel/setup.bash
```

If you see inside the /catkin_ws you will find three subfolders. They are source space (/src), build space (/build) and development space (/devel).

We are going to work in the source space (/src) where we create packages for our robotics projects which contain the source code files. These packages are catkin packages.

2) Packages

ROS packages are basically folders that contain source code files, libraries, model description files, other important files.

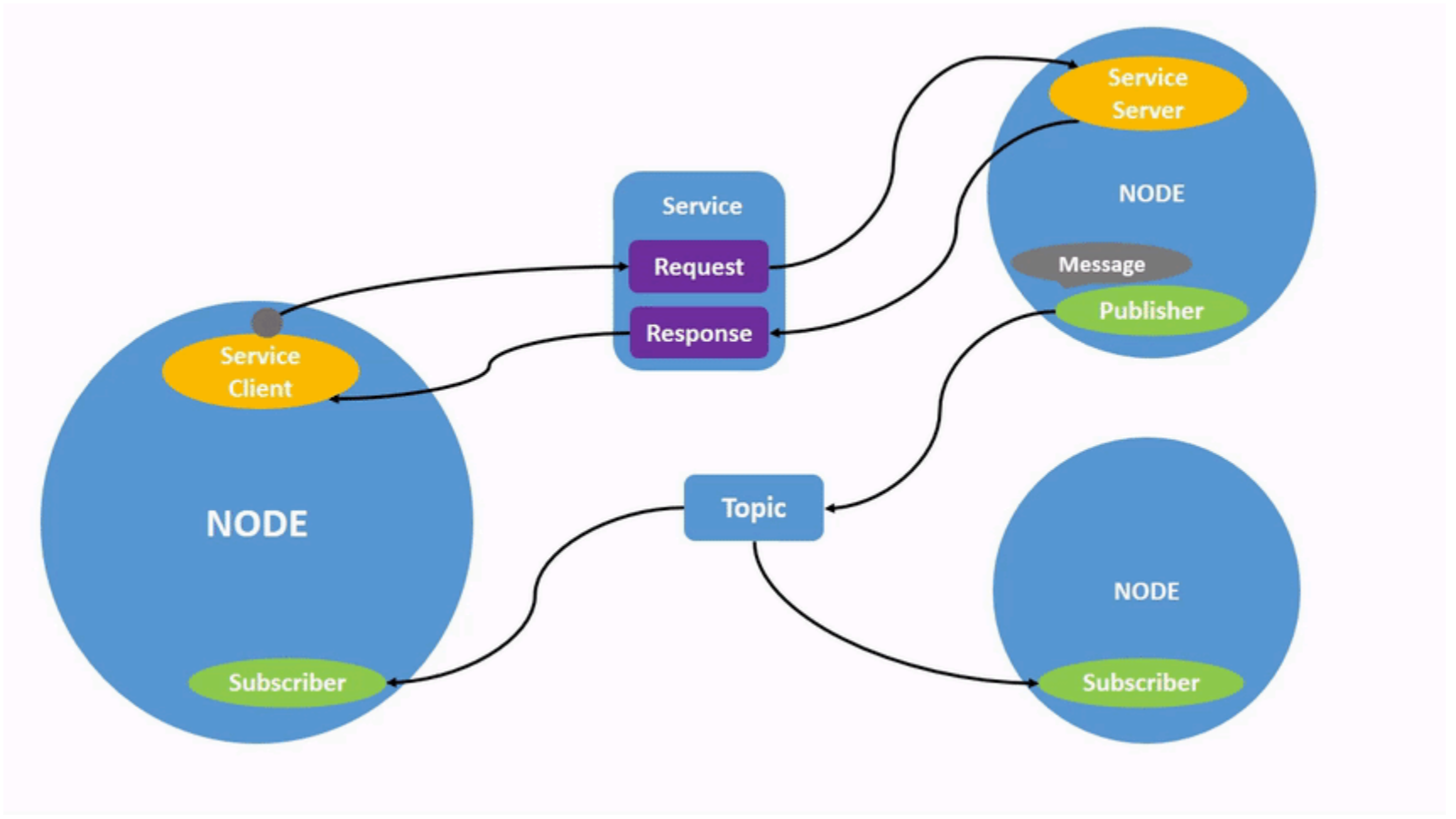
We make our own or import ROS packages in the /src folder.

Follow the instructions to create your own packages.

http://wiki.ros.org/ROS/Tutorials/CreatingPackage

3) ROS software architecture

The diagram below is a simplified representation of ROS architecture. Let us understand the components of this diagram one by one.



Nodes :

A ROS node is nothing but an executable program which contains a part of the code of the robot. A robotic application can have more than one node. For example we can have a node to compute inverse

kinematics, a node to send commands to the actuators, a node to get the joint angles from sensors, etc.

All these nodes communicate with each other through messages. There is a Master node running which will have all the addresses of the nodes running in the system. It is the Master node's responsibility to connect the Nodes with each other to start communication.

There are three modes in which nodes can communicate. These are topics, services and actions.

A ROS can be either written in C++ or Python.

Messages :

ROS message is a kind of data structure using which the topics, services and actions transfer information between nodes.

For topics , ROS messages are present in .msg files and have only one part - <fieldtype> <fieldname> .

For services, ROS messages are present in .srv files are have 2 parts (request and response).

#Request

<fieldtype> <fieldname>

#Response

<fieldtype> <fieldname>

ROS messages are important aspects of ros communication.

http://wiki.ros.org/Messages

Topics :

Topics are named buses over which nodes exchange messages. This is the most commonly used mode of communication for nodes in ROS.

In this mode, a node broadcasts a message with a topic name. This broadcasting node is called **Publisher** node.

A node that wants to receive the message will subscribe to this topic name. This node is known as the **Subscriber** node.

This mode of communication is unidirectional. The nodes are unaware of who they are communicating with i.e A publisher will keep publishing the data even if there are no/multiple subscribers. Similarly subscribers will keep listening to the topic even if there are no/multiple publishers. It is the responsibility of the ROS master to establish the connection between the publisher and subscriber when they are on the same topic.

To make things much clearer we can use the FM radio System as an analogy.

The transmitter (publisher) at the radio station (node) will transmit your favorite channel (message) at some fixed frequency (topic). Suppose you are sitting in your home and wish to listen to your desired channel (message), you will tune the radio receiver (subscriber) to that frequency (topic).

Neither do you know where exactly the radio station is nor the radio station wants to know where you are. Yet, you can enjoy the music on your favorite radio channel.

Topics are most suitable for continuous data streams like sensor data, Robot joint states, etc.

Some of the useful commands for topics are ,

a) To list the current topics subscribed/published by nodes.

   $ rostopic list

b) to get information about a specific topic.

$ rostopic info /topic_name

c) To get the message type of the topic.

$ rostopic type /topic_name

d) To print the data of a topic on the terminal window.

$ rostopic echo /topic_name

e) To publish data on a topic from the terminal window.

$ rostopic pub /topic_name message_type data

Eg: rostopic pub /topic_py std_msgs/Int32 “data: 200”

Launch file :

Till now we have been running each node in a separate terminal. What if we want to run all the nodes in the beginners_tutorials package at once? We may have to open more than ten terminals. That would be a tedious job, wouldn’t it?

In such cases the launch file will come in handy. Launch file is an XML file format which can run multiple nodes, load variables into ROS parameter server, etc. You can even include another launch file in your launch file. If you use roslaunch it will automatically start roscore if it detects that it is not already running, you don’t have to run roscore explicitly (to start ros master). It is good practice to run you nodes using launch files.

So let’s see how to write a launch file.

The description of launch files must be within the <launch> tags.

The node that we want to run must be with the <node>. The format to run the node using launch is given below.

<launch>

<node name=”node_name” pkg=”package_of_node” type=”cpp or py” output=”screen” />

</launch>

URDF :

URDF stands for Universal Robotic Description Format. URDF is an XML file format used in ROS to describe all elements of a robot and their properties. URDF will have the information about the robot's physical construction, geometry, dynamics, etc. In short, URDF will tell ROS how the links and joints are connected, type of joints, the dimensions of links, mass & inertia of links, etc.

Transformation using tf2 :

Almost everything in robotics is concerned with where things are - either relative to the robot itself or relative to other things. Whenever a robot wants to interact with the real world, it will need to know where the things it wants to interact with are.

Each link of a robotic arm is assigned a coordinate frame from the base to the end-effector. Not only the links of the robot but also the objects (like camera fixed to the wall, the ball which robot has to pick, goal pose of the end-effector, etc) around the robot are also assigned with coordinate frames. These frames will tell us the position and orientation of the frame w.r.t to other frames.

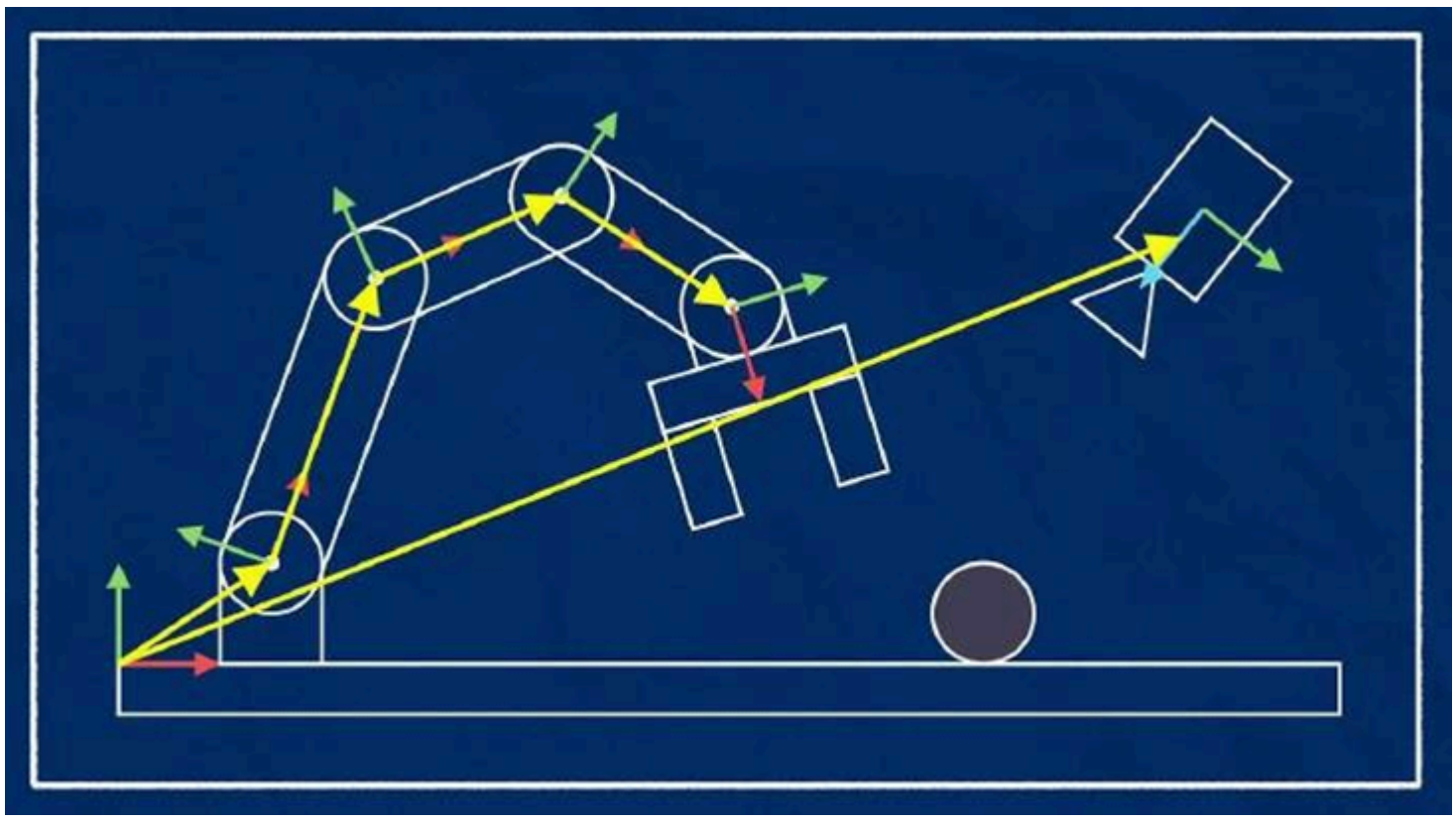
The coordinates of these frames are determined using the transformation matrices.

These frame coordinates need to be updated whenever either the robot moves, or other things move. Computing transformation matrices for all the frames, keeping track of all of those coordinates and updating them becomes a really involved task we move beyond a simple robot or robot application.

So this is where tf2 (tf stands for 'transform') comes in. The **tf2 package** enables ROS nodes to **keep track of coordinate frames and transform data between coordinate frames**.

Tf package relieves us free from deriving the transformation matrices and forward kinematics equations of our robot.

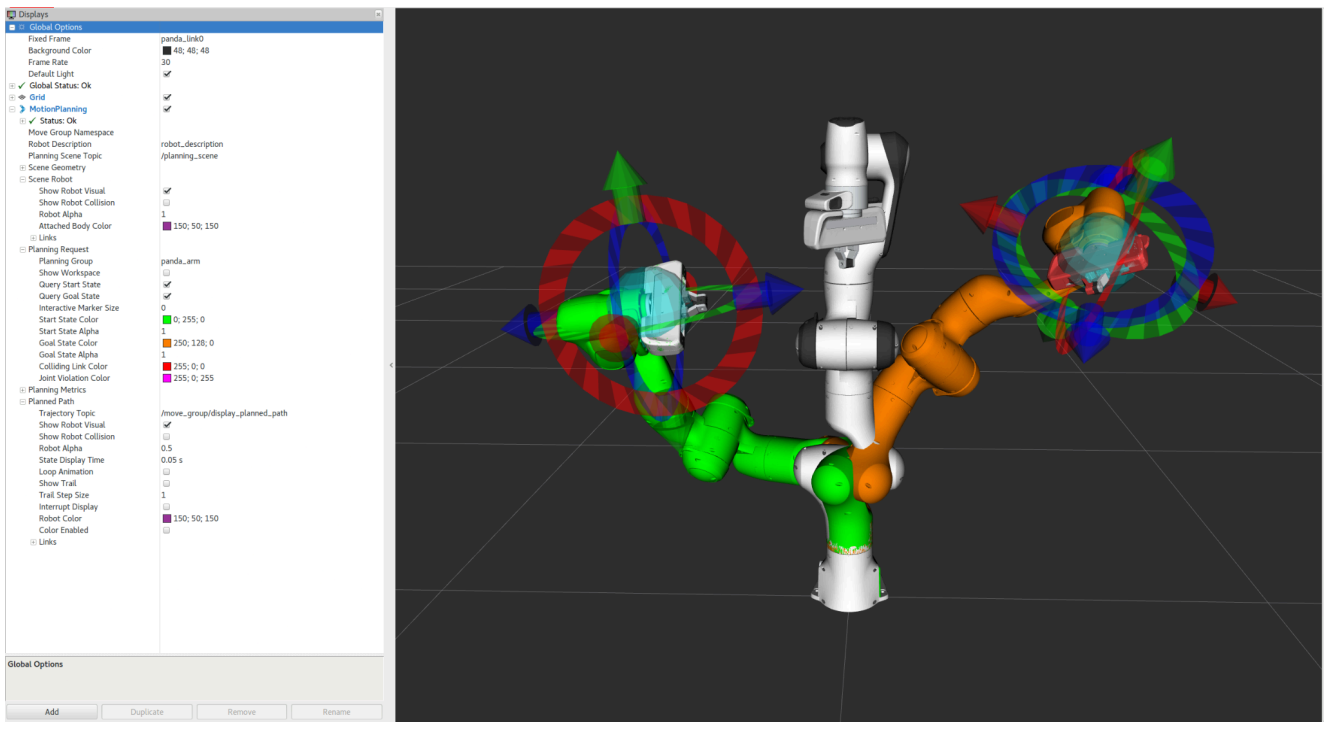
Go through the tf2 tutorials here - http://wiki.ros.org/tf2



RVIZ :

Rviz (ROS visualizer) is a powerful 3d visualization tool that allows you to view the robot's sensors and internal state.

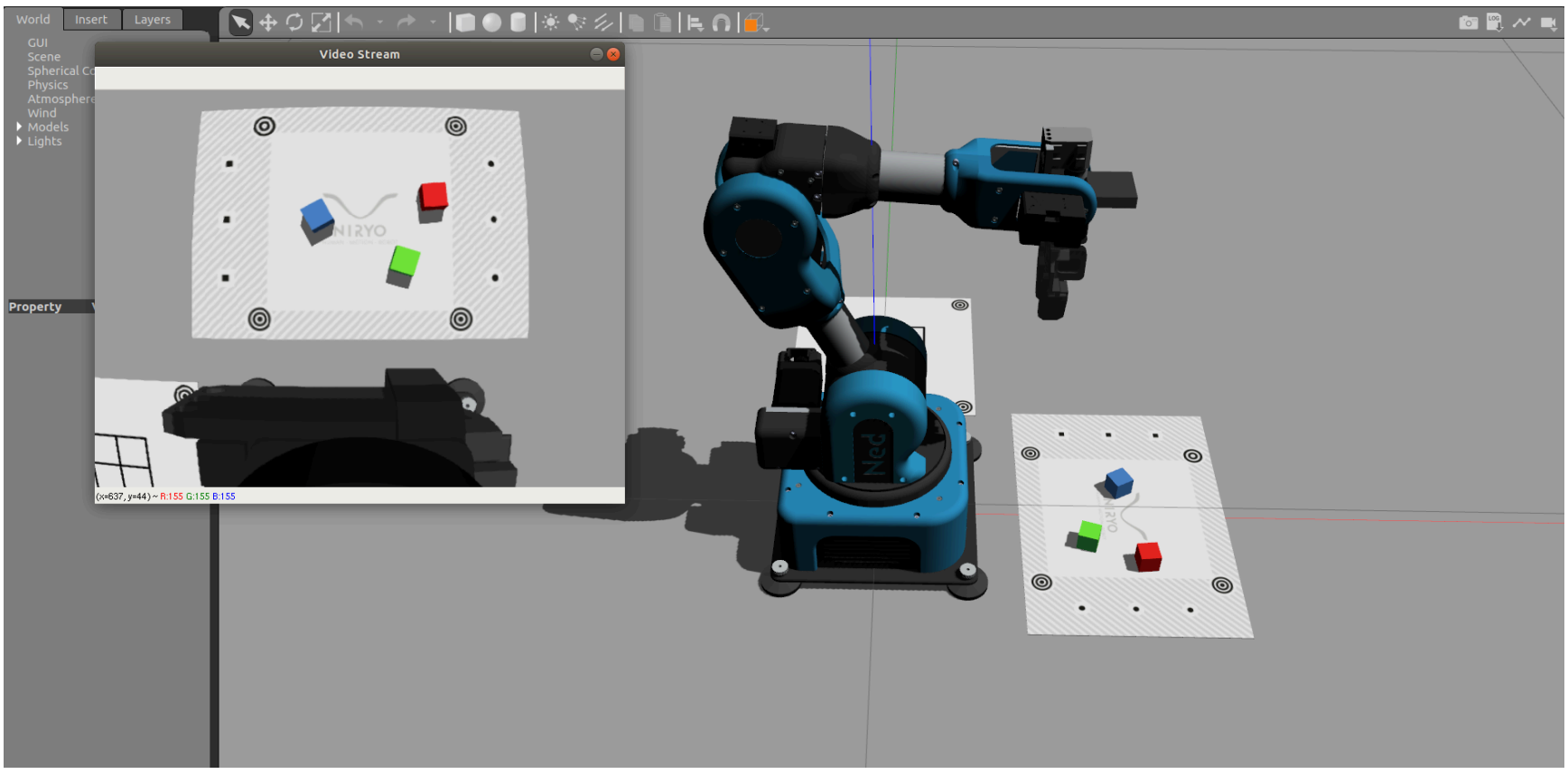
Rviz uses the URDF file to load our robot in the tool. It can also display live representations of sensor values coming over ROS Topics including camera data, point cloud data, infrared distance measurements, sonar data, and more.



Using RVIZ we can check our path planning or collision checking algorithms by inserting objects, trace the motion of any link of the robot, use interactive markers to move the robot & do a lot of other things.

Gazebo :

Gazebo is a robot simulator that performs physical computations, generates synthetic sensor data & offers convenient interfaces. The interfaces range from graphical tools to C++ APIs, allowing users & developers to achieve their goals more quickly. The purpose of Gazebo is to be the best possible software-only substitute for physical robots.



Useful Resources :

1) . http://wiki.ros.org/Documentation

2) . https://www.theconstructsim.com/

3) . https://www.rosroboticslearning.com/basics-of-ros

2. AI/ML_DL

Artificial Intelligence

Artificial Intelligence (AI) is a branch of computer science that focuses on creating systems or machines that can perform tasks that typically require human intelligence. These tasks include learning from data, reasoning, problem-solving, understanding natural language, and perceiving the environment. AI systems are designed to mimic cognitive functions associated with human minds, such as learning from experience, adapting to new situations, and making decisions based on available information. By leveraging algorithms, data, and computing power, AI enables machines to perform tasks with varying degrees of autonomy, ultimately aiming to enhance efficiency, productivity, and decision-making across various domains and industries.

Know more:
https://azure.microsoft.com/en-us/blog/what-is-artificial-intelligence/

Machine Learning (ML)

Machine learning is a subset of artificial intelligence (AI) that focuses on enabling computers to learn and improve from experience without being explicitly programmed. The fundamental idea behind machine learning is to develop algorithms that allow computers to recognize patterns and make decisions based on data.

There are several types of machine learning techniques:

- **Supervised Learning**: In supervised learning, the algorithm is trained on a labeled dataset, where each input is associated with the correct output. The algorithm learns to map inputs to outputs, and once trained, it can make predictions on new data.

- **Unsupervised Learning**: Unsupervised learning involves training algorithms on unlabeled data. The algorithm tries to find patterns or hidden structures in the data without explicit guidance. Clustering and dimensionality reduction are common tasks in unsupervised learning.

- **Semi-Supervised Learning**: This type of learning combines elements of both supervised and unsupervised learning. The algorithm is trained on a dataset that contains both labeled and unlabeled data.

• **Reinforcement Learning**: Reinforcement learning involves training agents to make sequential decisions in an environment to maximize some notion of cumulative reward. The agent learns through trial and error, receiving feedback from the environment in the form of rewards or penalties.

Andrew NG ML Specialization:

Machine Learning Specialization by Andrew NG

Blog for Introduction to ML: https://www.javatpoint.com/machine-learning

(It is highly recommended to understand the basic procedures of ML, and the various algorithms from a mathematical standpoint. (Gradient descent, Loss functions, etc)

ML Algorithms

Machine learning (ML) algorithms are computational techniques that enable computers to learn patterns and relationships from data and make predictions or decisions based on that learned information. These algorithms form the foundation of various ML models, which are trained on datasets to perform specific tasks.

https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

(Familiarizing yourself with Linear Regression, Logistic Regression, K-Nearest Neighbors (KNN), K-Means Clustering, Decision Trees, and Support Vector Machines (SVMs), along with their underlying mathematical concepts is recommended)

Refer to YouTube to visualize and learn.

Deep Learning (DL)

Deep learning is a subfield of machine learning that focuses on training artificial neural networks (ANNs) with multiple layers of abstraction, allowing them to learn complex patterns in large datasets. The term "deep" refers to the multiple layers through which data is transformed as it passes through the network.

Key components of deep learning include:

• **Artificial Neural Networks (ANNs)**: ANNs are computational models inspired by the structure and function of biological neural networks in the human brain. They consist of interconnected nodes (neurons) organized in layers, including input, hidden, and output layers. Each connection between nodes is associated with a weight that is adjusted during the learning process.

- **Deep Neural Networks (DNNs)**: DNNs are ANNs with multiple hidden layers between the input and output layers. These hidden layers enable the network to learn hierarchical representations of the input data, allowing it to capture complex relationships and patterns.

- **Convolutional Neural Networks (CNNs)**: CNNs are a type of deep neural network designed specifically for processing structured grid data, such as images. They consist of layers of convolutional filters that extract features from the input data and layers of pooling operations that reduce the spatial dimensions of the data.

- **Recurrent Neural Networks (RNNs)**: RNNs are another type of deep neural network that is well suited for processing sequential data, such as time series or natural language. They have connections between neurons that form directed cycles, allowing them to maintain internal state and process sequences of inputs. Deep learning has revolutionized many fields, including computer vision, natural language processing, speech recognition, and reinforcement learning. It has led to significant breakthroughs in tasks such as image recognition, machine translation, autonomous driving, and medical diagnosis.

For deeper understanding refer:
https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing3-types-of-neural-networks-in-deep-learning/

Playlist to refer to CNN: https://youtu.be/E5Z7FQp7AQQ?feature=shared

Basic CNN Architecture:

Basics of the Classic CNN. How a classic CNN (Convolutional Neural… | by Chandra Churh Chatterjee | Towards Data Science

(Too much depth isn't required, just try to get a very basic understanding of what the layered structures are trying to achieve)

To get some intuition on DL: 3Blue1Brown

For Deep learning refer:

Deep Learning Specialization [5 courses] (DeepLearning.AI) | Coursera

(For a thorough understanding of Deep Learning. Weeks 1,2 are enough)

Natural Language Processing (NLP)

Natural Language Processing (NLP) is a field of artificial intelligence (AI) that focuses on enabling computers to understand, interpret, and generate human language in a way that is both meaningful and useful. It involves

the interaction between computers and humans through natural language, allowing machines to comprehend and respond to human language data in various forms, such as text, speech, or even gestures. NLP algorithms and techniques enable machines to perform tasks like language translation, sentiment analysis, text summarization, speech recognition, and language generation, among others. Ultimately, NLP aims to bridge the gap between human communication and computational systems, opening a wide range of applications across industries such as healthcare, finance, customer service, education, and more.

Refer following blog:
https://www.deeplearning.ai/resources/natural-language-processing/

Computer Vision

Computer vision is a multidisciplinary field focused on enabling computers to interpret and understand visual information from the world around us. It involves developing algorithms and systems that can extract, analyze, and interpret visual data from images or videos, like how humans perceive and interpret visual information. Computer vision techniques encompass a wide range of tasks, including image recognition, object detection and tracking, scene understanding, image generation, and more. By leveraging machine learning, deep learning, and image processing techniques, computer vision enables applications such as facial recognition, autonomous vehicles, medical image analysis, augmented reality, and robotics, among others, revolutionizing industries and enhancing human-computer interaction.

Link to OpenCV blogs:

https://www.geeksforgeeks.org/opencv-overview/

https://fullscale.io/blog/opencv-overview/

Reinforcement Learning (RL)

Reinforcement learning is a machine learning paradigm inspired by behavioral psychology, where an agent learns to make decisions by interacting with an environment. The agent takes actions in the environment and receives feedback in the form of rewards or penalties, based on the consequences of its actions. The goal of reinforcement learning is to learn a policy or strategy that maximizes cumulative rewards over time. It involves learning through trial and error, where the agent explores different actions and learns from the outcomes to improve its decision-making abilities. Reinforcement learning has applications in various domains, including robotics, game playing, autonomous vehicles, and recommendation systems.

Link to reinforcement learning blog:
https://gordicaleksa.medium.com/how-to-get-started-with-reinforcement-learning-rl-4922fafeaf8c

Probability and Statistics:

Probability and statistics are branches of mathematics that deal with the study of uncertainty, variability, and the analysis of data. Probability focuses on quantifying uncertainty and making predictions about the likelihood of different outcomes occurring, while statistics involves collecting, organizing, analyzing, and interpreting data to make informed decisions or draw conclusions.

In the context of artificial intelligence (AI) and machine learning (ML), probability and statistics play a crucial role in several aspects:

Modeling Uncertainty: Probability theory provides a framework for modeling uncertainty in AI and ML systems. Uncertainty arises from various sources, such as noisy sensor data, incomplete information, or stochastic processes. By quantifying uncertainty using probability distributions, AI and ML models can make robust predictions and decisions in the face of uncertain or incomplete information.

Learning from Data: Statistics is essential for learning from data in AI and ML. Machine learning algorithms learn patterns, relationships, and structures from data, and statistical methods are used to analyze and interpret this data. Techniques such as hypothesis testing, regression analysis, and clustering enable ML models to extract meaningful insights and knowledge from datasets.

Inference and Prediction: Probability theory enables AI and ML models to perform inference and make predictions. Bayesian inference, for example, combines prior knowledge with observed data to update beliefs or make probabilistic predictions about future events. This probabilistic reasoning allows AI systems to make decisions under uncertainty and adapt to new information over time.

Model Evaluation and Validation: Statistics provides tools and techniques for evaluating and validating AI and ML models. Methods such as cross-validation, hypothesis testing, and performance metrics (e.g., accuracy, precision, recall) help assess the performance and generalization ability of machine learning models on unseen data. Statistical significance tests can determine whether observed differences are due to chance or represent true patterns in the data.

Brush up your concepts from the following link:
https://www.mrmacmaths.com/statistics

3. Dev & AR_VR

Augmented reality

Augmented reality is made up of the word “augment” which means to make something great by adding something to it. So basically, augmented reality is a method by which we can alter our real world by adding some digital elements to it. This is done by superimposing a digital image on the person’s current view thus it enhances the experience of reality.

Virtual reality

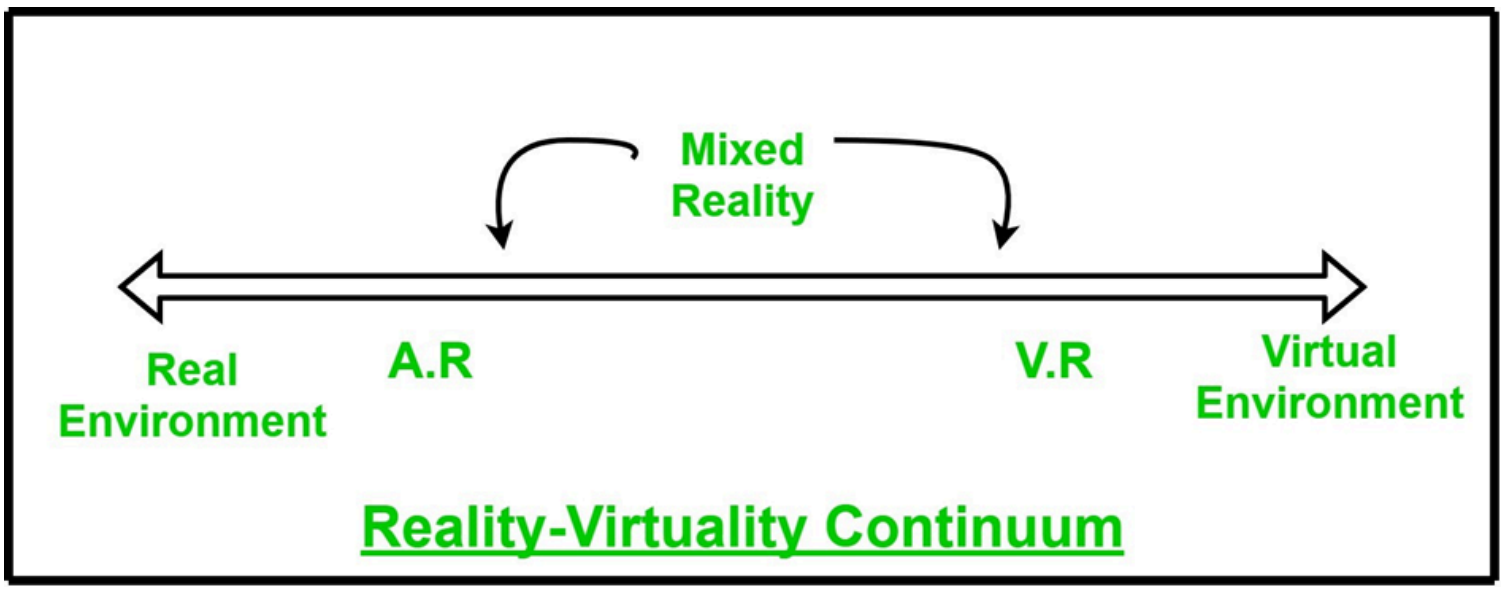
Virtual reality is the use of computer technology to create simulated environments. Virtual reality places the user inside a three-dimensional experience. Instead of viewing a screen in front of them, users are immersed in and interact with the 3D world.

Difference between Augmented Reality and Virtual Reality

Virtual reality makes a virtual environment and puts the user in it whereas Augmented reality just adds the virtual components into the user’s real-world view. Virtual reality sends a person into a virtual place created by a computer whereas augmented reality brings the digital world into our real world.to experience virtual reality the person needs to wear a special VR headset that is connected to a computer like the Oculus Rift or a gaming console like PlayStation VR but there are devices that work with a smartphone-like Google Cardboard.

The Reality-Virtuality Continuum

The reality-virtuality continuum is a scale that was given by Paul Milgram. It is a scale which has two extremes. One part depicts ‘virtuality’ or an environment which is completely virtual and the other part describes a real environment or ‘reality’ and the middle part is termed as “mixed reality”. AR, VR and MR are collectively called “XR”.



How to build AR/VR applications?

- The first stage is to choose the different tools/platforms that will help you build the AR/VR application.
- Once you have chosen the platform, you need to decide on the application mechanism(What will the application Contain)
- Android and iOS both have helpful SDK's (Software Development Kits) like AR Foundation and AR Kit respectively.AR/VR applications can also be developed for web by APIs like WebXR and Three.JS.
- One of the most popular tools used for developing AR applications for android is Unity3D.Android studio can also be used for developing immersive apps for android.

Scripting Languages

C# and C/C++ are commonly regarded as the top programming languages for XR development. Scripts are used to set/change the behavior of game objects in the scene.

C# Inspired by its predecessors C and C++, C-sharp (C#) is a programming language developed by Microsoft, capable of a wide range of tasks, but most importantly, it's the programming language used by the game engine by Unity Technologies.

Some Popular Companies using XR

Companies like Meta,Snapchat,HTC,Microsoft,Niantic are seriously working towards Developing AR applications.Pokemon Go is one of the most popular AR games

Some important Concepts

Occlusion

Occlusion refers to what happens when an image or object is blocked by another. Move your hand in front of your face, you've just occluded the computer screen with your hand. However, imagine if you moved your hand in front of your face and the screen was still visible, you would probably get a little concerned. AR objects have to play by the rules of occlusion if we want them to seem real.

This means that AR hardware has to not only understand where the object is in the room but also its relative distance from the user compared with any other objects physical or digital. Occlusion means hiding virtual objects behind other virtual objects, and ones in the real world.

Say you move behind a wall while using AR. If you still see the AR objects, it will break your sense of immersion, that sense of AR objects actually being in the real world in the app.

Lighting

Just like a real world object, objects in AR need to respond to different patterns of lighting to make sense in our minds. The colors, shading, and shadows cast by these objects all need to behave properly both in the initial lighting of a scene and in the case of a lighting change. For example, if you dim the lights during an AR experience, then the AR objects should change in color and shading appearance. Similarly, if you move an object around, the shadows need to move accordingly just like they would in real life.

Cloud Anchors

Building on this concept and supported by the cloud, Cloud Anchors are a cross-platform feature that allow both iOS and Android device users to share the same AR experience despite using different underlying AR technologies. Where traditionally anchors have been isolated to one device—or one augmented reality—these anchors can be shared by multiple different devices simultaneously.

This makes AR possible for groups of people rather than just individuals, allowing for shared, collaborative experiences like redecorating your home, playing games and making art in 3D space together.

For more Information on Cloud Anchors Visit
https://www.youtube.com/watch?v=MeZcQguH124

Interacting with AR application

1. **Drag and Drop**: This feature lets users drag and drop objects from a menu of 3D digital assets “onto” the screen to place them on a real world surface (that has been spatially mapped by ARCore).

2. **Voice:** Voice is quickly emerging as a powerful interaction tool that creators can build into AR apps. Building in pre-programmed voice commands allows users to execute specific actions within the AR app. This is often best achieved by embedding the Google Assistant SDK to add voice-enabled smart interaction.

3. **Tap:** With the tap mechanic, users can place objects in the real world by tapping on the screen. ARCore uses raycasting (the use of ray-to- surface as an intersection test), and projects a ray to help estimate where the AR object should be placed in order to appear in the real- world surface in a believable way. Another way to use tap is as a mechanic to interact with a digital object that is already placed in the scene. For example, maybe the app allows users to animate a 3D object when users tap on it.

4. **Pinch and Zoom**: Pinch and zoom lets users enlarge or scale down a 3D object—or use the interaction in creative ways to build a game or user experience. For example, this could be used to pull back the string of a bow in a bow-and-arrow target game.

5. **Slide:** Users can interact with 3D objects through sliding, which translates (or moves) objects in-scene, or use it as a game mechanic. For example, say you are creating an AR paper toss game. You could enable a slide interaction to let users project or throw papers into a trash can.

6. **Tilt:** Using the accelerometer and gyroscope, a tilt of the phone can also be used as an input for creative interactions. An easy example would be to make a “steering wheel” mechanic for a racing tabletop AR game.

Glossary

Accelerometer - measures acceleration; for AR Foundation-capable smartphones, it helps enable motion tracking.

Anchors - user-defined points of interest upon which AR objects are placed. Anchors are created and updated relative to geometry (planes, points, etc.) AR Foundation detects in the environment.

Asset - refers to a 3D model.

Augmented Reality (AR) - a direct or indirect live view of a physical, real- world environment whose elements are "augmented" by computer- generated perceptual information.

Computer Vision - a blend of artificial intelligence and computer science that aims to enable computers (like smartphones) to visually understand the surrounding world like human vision does.

Concurrent Odometry and Mapping (COM) - motion tracking process for AR Foundation, and tracks the smartphone’s location in relation to its surrounding world .

Drift - refers to the accumulation of potential motion tracking error. If you walk around digital assets too quickly, eventually the device's pose may not reflect where you actually are. AR Foundation attempts to correct for drift over time and updates Anchors to keep digital objects placed correctly relative to the real world.

Feature Points - are visually distinct features in your environment, like the edge of a chair, a light switch on a wall, the corner of a rug, or anything else that is likely to stay visible and consistently placed in your environment. AR Foundation uses feature points in the captured camera image to compute change in location, further environmental understanding, and place planes in an AR app.

GPS - global navigation satellite system that provides geolocation and time information; for AR Foundation-capable smartphones, it helps enable location-based AR apps.

Gyroscope - measures orientation and angular velocity; for AR Foundation- capable smartphones, it helps enable motion tracking.

Hit-testing - used to take an (x,y) coordinate corresponding to the phone's screen (provided by a tap or whatever other interaction you want your app to support) and project a ray into the camera's view of the world. This returns any planes or feature points that the ray intersects, along with the pose of that intersection in world space. This allows users to select or otherwise interact with objects in the environment.

HMD - Head-Mounted Display.

Immersion - the sense that digital objects belong in the real world. Breaking immersion means that the sense of realism has been broken; in AR this is usually by an object behaving in a way that does not match our expectations.

Inside-Out Tracking - when the device has internal cameras and sensors to detect motion and track positioning.

Light estimation - allows the phone to estimate the environment's current lighting conditions.

Magnetometer - measures cardinal direction and allows AR Foundation- capable smartphones to auto-rotate digital maps depending on physical orientation, which helps enable location-based AR apps.

Motion Tracking - in the basic sense, this means tracking the movement of an object in space. AR Foundation uses your phone's camera, internal gyroscope, and accelerometer to estimate its pose in 3D space in real time.

Multi-plane detection – AR Foundation's ability to detect various surfaces as different heights and depths.

Occlusion - when one 3D object blocks another 3D object. Currently, this can only happen with digital objects. AR Foundation objects cannot be occluded by a real world object.

Outside-In Tracking - when the device uses external cameras or sensors to detect motion and track positioning.

Plane Finding - the smartphone-specific process by which ARCore determines where horizontal and vertical surfaces are in your environment and uses those surfaces to place and orient digital objects.

Pose - the unique position and orientation of any object in relation to the world around it, from your mobile device to the augmented 3D asset that you see on your display.

Raycasting - projecting a ray to help estimate where the AR object should be placed in order to appear in the real-world surface in a believable way; used during hit testing.

Runtime - when edits/changes are made during active gameplay mode or while your app is running. For example, you can download Poly assets while your application is in gameplay mode or running.

Scaling - When a placed AR object changes size and/or dimension relative to the AR device's position; enabled by environment understanding.

SLAM - motion tracking process that tracks the device in relation to its surrounding world.

Spatial mapping - the ability to create a 3D map of the environment and helps establish where assets should be posed.

Standalone headset - VR or AR headset that does not require external processors, memory, or power.

Surface detection - allows AR Foundation to place digital objects on various surface heights, to render different objects at different sizes and positions, and to create more realistic AR experiences in general (Module 4).

Unity - cross-platform game engine and development environment for both 3D and 2D interactive applications.

User interface metaphor - gives the user instantaneous knowledge about how to interact with the user interface, like a QWERTY keyboard or a computer mouse.

Virtual Reality (VR) - the use of computer technology to create a simulated environment, placing the user inside an experience.

Some Important Links

Tutorials for XR dev in Unity:
https://learn.unity.com/search?k=%5B%22tag%3A5900b95a090915001e654b47%22%5D

Beginner Google Collab Tutorial :
https://codelabs.developers.google.com/arcore-unity-ar-foundation?hl=en#0

C# unity Documentation :
https://docs.unity3d.com/Manual/ScriptingSection.html

C# programming guide :
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/

Unity XR manual:
https://docs.unity3d.com/Manual/XR.html

Some Other Links for reference

- **https://www.w3.org/TR/webxr/**
- **https://threejs.org/docs/index.html#manual/en/introduction/Creating-a- scene**
- **https://www.youtube.com/c/DilmerV**
- **https://www.youtube.com/c/CircuitStreamEdu**
- **https://www.youtube.com/c/DineshPunni**
- **https://www.youtube.com/watch?v=gB1F9G0JXOo&t=9177s**

Note: Along with the content given in this doc you are expected to read some basics of getting started with XR in unity and C# language from the links attached.You are also expected to know very basic Programming and some basic Oops Concepts.Some other Links are provided if you are interested do visit these links.